Symantec C++

Creating an Application with Symantec C++

Part Two

Starting a Project and Defining Workspaces

3

This chapter describes the initial steps involved in writing an application: creating the project that defines your target and then defining workspaces used in working on a project. These topics are covered here in sufficient depth to get you started; more detailed information is presented in Chapter 15, "More about Projects and Workspaces."

What Are Projects and Workspaces?

A project is a collection of files from which an executable or library is generated. The IDDE automatically generates a file (called the makefile) that tracks the dependencies in your project. This makefile is configured using the project option settings you specify. The IDDE executes the makefile when you build your project. File extensions are used to determine which tool is needed to build each component. Project building is discussed in Chapter 8, "Testing an Application."

Workspaces, which are among the IDDE's most useful features, are window configurations used for particular tasks. To create a workspace, you name and save the exact arrangement of windows on your screen. Any time you need to perform a similar task, you can instantly open that workspace, with the windows organized the way you want them. For more information on workspaces, refer to Chapter 15, "More about Projects and Workspaces."

Starting a Project

This section describes how to start a new project, how to open an existing project, and how to edit the project contents.

Purpose of a project

The project is central to building an application with the IDDE. A project is a container for the application you are building. It contains the various components necessary for building an application or a library, as well as information about how to build it.

Projects speed development time because they let you recompile only the source files that have changed, or whose header files have changed, since the last time the project was built. For example, if your program has five source files and you have changed one of them since the last build, only that file is recompiled when you build the project. (You can, however, choose to recompile all the files.) The project management system does this by automatically analyzing the dependencies of the source files and constructing or updating the makefile each time the project is built.

Contents of a project

A project can contain several different types of file, including C and C++ source, assembly language source, resource scripts, object files, libraries, and module definition files. And because a project is built in a hierarchical manner, you can include projects within projects.

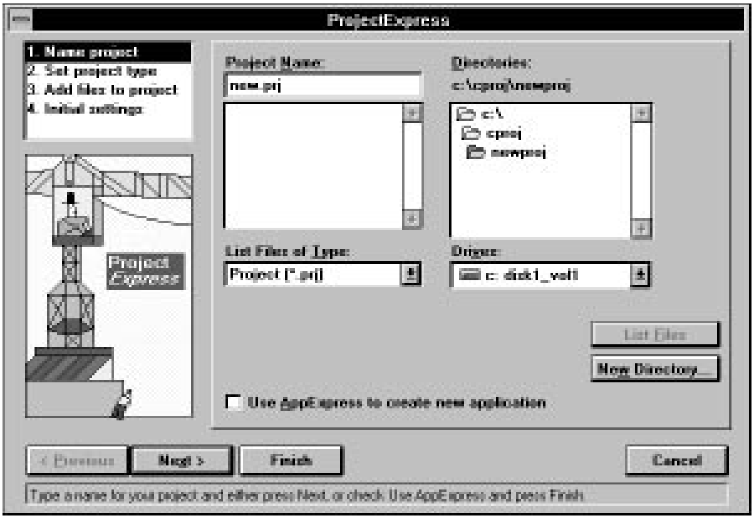
The IDDE stores information about a project on disk as a project file with a `.prj` extension. Among other information, this file includes a list of the source files contained in a project. When you build the project, the IDDE constructs a makefile (`.mak`)—or updates the existing makefile—based on the files the project contains. Project options are stored in an option set file (`.opn`) that is referenced in the project file. The option set file can be loaded into another project, making it easy to transfer all option settings from one project to another.

Creating a new project

To create a new project, choose **New** from the **Project** menu. The **ProjectExpress** dialog box opens. ProjectExpress lets you specify the project name, initial project options, and initial project contents of the new project. The **ProjectExpress** dialog box contains four pages of options, described in the following sections.

Naming the project

Initially ProjectExpress displays the Project Name page (see Figure 3-1).



1. Name project
2. Set project type
3. Add files to project
4. Initial settings

Project Name: new.prj

Directories: c:\cproj\newproj

c:\
cproj
newproj

List Files of Type: Project (*.prj)

Drives: c: disk1_vol1

☐ Use AppExpress to create new application

Type a name for your project and either press Next, or check Use AppExpress and press Finish.

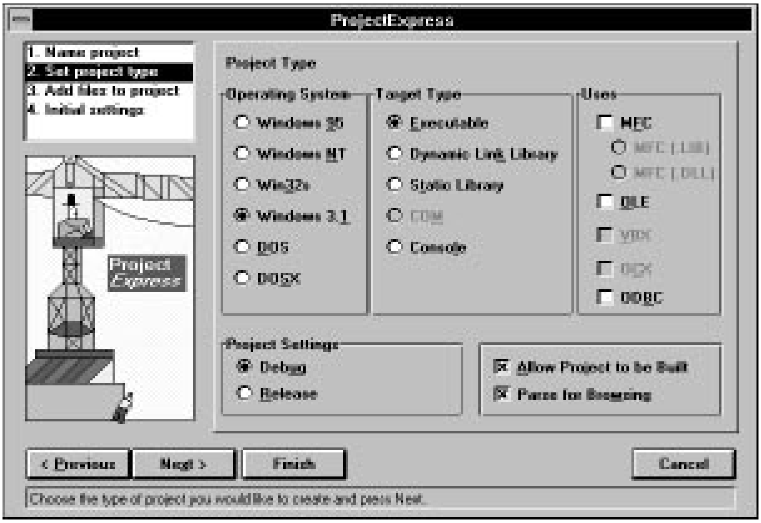
Figure 3-1 ProjectExpress Project Name page

Select the directory in which you want to create the project from the Directories listbox, or click on New Directory to make a new directory for this project. Enter the name of the new project in the Project Name textbox.

If you select Use AppExpress to create new application, then click on Finish, AppExpress will start. AppExpress is discussed in Chapter 4, “Generating an Application Framework.”

Setting the project type

To set the target operating system, target type, and other options, click on Next, or select Set project type from the left listbox, to switch to the Project Type page (Figure 3-2).



Project Type

Operating System
○ Windows 95
○ Windows NT
○ Win32s
● Windows 3.1
○ DOS
○ DOSX

Target Type
● Executable
○ Dynamic Link Library
○ Static Library
○ COM
○ Console

Uses
☐ MFC
○ MFC (.LIB)
○ MFC (.DLL)
☐ OLE
☐ VBX
☐ OCX
☐ ODBC

Project Settings
● Debug
○ Release

☒ Allow Project to be Built
☒ Parse for Browsing

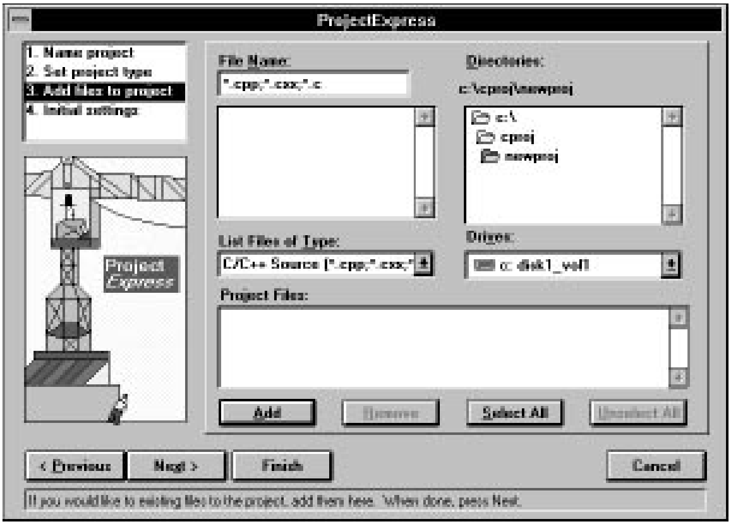
Choose the type of project you would like to create and press Next.

Figure 3-2 ProjectExpress Project Type page

After the project is created, you can modify these settings by choosing **Settings** from the **Project** menu. These options are discussed in more detail in Chapter 15, “More about Projects and Workspaces.”

Adding files to the project

To add pre-existing source, header, or other files to the new project, click on Next, or select Add files to project from the left listbox, to continue to the Project Edit page (Figure 3-3).



File Name: *.cpp;*.cxx;*.c

Directories: c:\cproj\newproj

c:\
cproj
newproj

List Files of Type: C/C++ Source [*.cpp;*.cxx;*

Drives: c: disk1_vol1

Project Files:

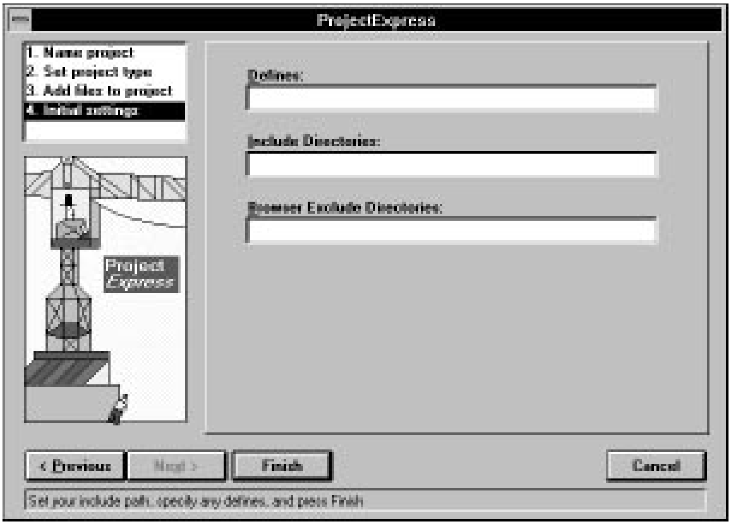
If you would like to existing files to the project, add them here. When done, press Next.

Figure 3-3 ProjectExpress Project Edit page

If you are creating a new project, you do not need to do anything on this page. After the project is created, you can open a similar dialog box by choosing **Edit** from the **Project** menu (see the section “Adding and deleting project files” later in this chapter).

Setting defines and include directories

To define macros, specify search paths, or exclude a directory from parsing, click on Next, or select Initial settings, to continue to the last page of the **ProjectExpress** dialog box (Figure 3-4).



Set your include path, specify any defines, and press Finish

Figure 3-4 ProjectExpress Initial Settings page

To define a macro on the compiler command line, enter the macro in the Defines textbox (for example, `COLOR=1`). Separate multiple macro definitions with semicolons. Type any `#include` file search paths you want on the compiler command line in the Include Directories textbox. Type any directories to be excluded from parsing in the Browser Exclude Directories textbox. (For more information about parsing, see Chapter 5, “Defining Classes and Their Hierarchies.”

In general, you can leave these fields blank. You may change these options later by choosing **Settings** from the **Project** menu.

After the project is set up the way you want, click on Finish to create the new project.

Opening an Existing Project

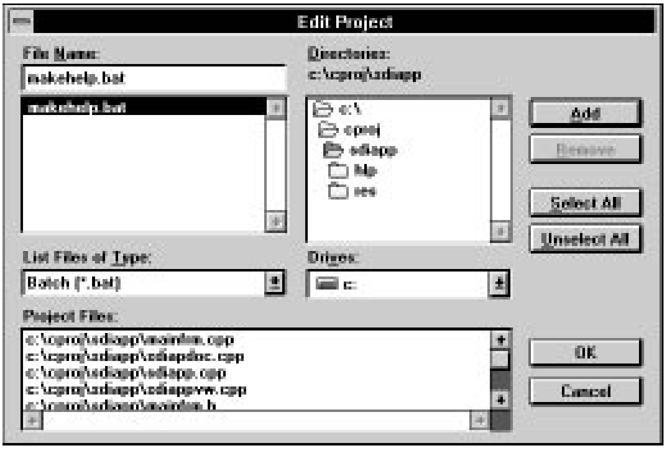
To open a project that already exists, choose **Open** from the **Project** menu. The IDDE displays the **Open Project** dialog box. Select the desired project filename and click OK.

IDDE lets you work with only one project at a time. If you're already working with a project when you open a new one, the IDDE closes the project in process.

An additional method for opening existing projects is to choose one from the list of projects at the bottom of the **Project** menu. Projects are added to this menu as they're opened or created. This makes it easier for you to switch back and forth between projects as you work.

Adding and deleting project files

To add or remove files from your project, choose **Edit** from the **Project** menu. The IDDE opens the **Edit Project** dialog box, shown in Figure 3-5.



File Name: makehelp.bat

makehelp.bat

Directories: c:\cproj\sdiapp

c:\
cproj
sdiapp
hlp
res

List Files of Type: Batch (*.bat)

Drives: c:

Project Files:

c:\cproj\sdiapp\mainfrm.cpp
c:\cproj\sdiapp\sdiapdoc.cpp
c:\cproj\sdiapp\sdiapp.cpp
c:\cproj\sdiapp\sdiappvw.cpp
c:\cproj\sdiapp\mainfrm.h

Figure 3-5 Edit Project dialog box

The Project Files listbox contains the files in your project.

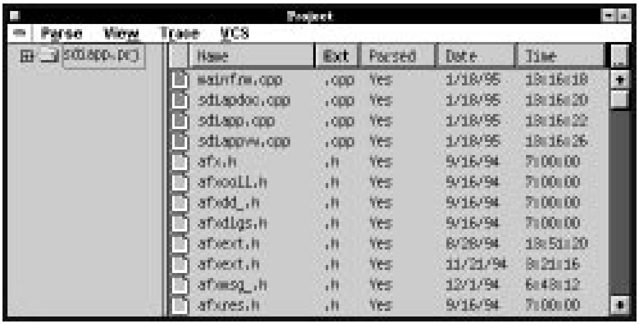
- To add a file to your project, select the file from the File Name listbox and click on Add, or double-click on the name of the file.
- To add all the files in the File Name listbox, click in the listbox, click on Select All, and then click on Add.
- To remove a file from the project, select it from the Project Files listbox and click on Remove, or double-click on the name of the file.
- To remove all the files from the project, click on the Project Files listbox, click on Select All, and then click on Remove.

To make the changes to your project, click OK. To leave your project as it was before, click Cancel.

After you click OK, the IDDE checks your project for dependencies and creates a makefile. While checking for dependencies, the IDDE adds the additional files it needs to build your project. For example, it adds all the header files that your source files reference with the `#include` directive.

The Project window

The Project window, shown in Figure 3-6, displays a list of files in the current project. You can open the Project window by choosing **Project** from the **Goto View** submenu of the IDDE's **Window** menu.



Name	Ext	Parsed	Date	Time
mainfrm.cpp	.cpp	Yes	1/18/95	13:16:18
sdiapdoc.cpp	.cpp	Yes	1/18/95	13:16:20
sdiapp.cpp	.cpp	Yes	1/18/95	13:16:22
sdiappvw.cpp	.cpp	Yes	1/18/95	13:16:26
afx.h	.h	Yes	9/16/94	7:00:00
afxcoll.h	.h	Yes	9/16/94	7:00:00
afxdd_.h	.h	Yes	9/16/94	7:00:00
afxdlgs.h	.h	Yes	9/16/94	7:00:00
afxext.h	.h	Yes	8/28/94	13:51:20
afxext.h	.h	Yes	11/21/94	8:21:16
afxmsg_.h	.h	Yes	12/1/94	6:48:12
afxres.h	.h	Yes	9/16/94	7:00:00

Figure 3-6 Project window

You can double-click on the name of a source file in the right pane to open that file for editing in a Source window. (See Chapter 6, “Editing Program Code,” for a description of text editing functions.)

You can see the current project’s subprojects, or open a subproject, by double-clicking on the project name in the left pane.

The icon to the left of each filename indicates certain properties of the file. If the icon is blue, the file was explicitly added to the project; if the icon is gray, the file is included in the project by a dependency relationship or by parsing.

The icon next to each filename contains different information during debugging. An asterisk indicates that the module contains debug information. A “T” at the right of the icon indicates that tracing is enabled in the module. Dots indicate that the module contains breakpoints: a green dot indicates enabled breakpoints, and a red dot indicates disabled breakpoints.

Closing a project

To close a project that is currently open, choose **Close** from the **Project** menu. The project will be saved automatically.

Importing a Microsoft or Borland project

You can import an existing Microsoft or Borland project into the IDDE project system, by using the other product’s makefile.

First, choose **Open** from the **Project** menu. In the “List files of type” listbox, choose Import Make. When you open a Microsoft or Borland makefile, the IDDE lets you work with it as you would a Symantec C++ project.

To build the project with the original Microsoft or Borland makefile, use the Make page under the Build tab in the **Project Settings** dialog box (see Chapter 16, “More about Project Build Settings”) to call the original makefile or batch file.

Defining Workspaces

This section describes how to set up and save your own workspaces.

The purpose of workspaces

The IDDE’s workspace feature lets you set up multiple screen configurations, each of which is optimized for a specific task. For example, you can have a workspace for editing source files, another for working with project resources, and another for debugging DLLs. You can define up to five different workspaces.

Creating a workspace

To start a new workspace, choose **New** from the **Workspace** submenu of the IDDE's **Environment** menu. Type the name of the new workspace in the **Workspace Name** dialog box. This name appears in the Workspace toolbox, as shown in Figure 3-7. You can then configure the screen as you like by opening the windows you need and positioning and sizing them to suit your requirements. You can refine the workspace as you work; the IDDE automatically saves changes to a workspace configuration when you exit the workspace.

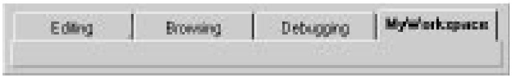


Figure 3-7 Workspace toolbox

Selecting a workspace

To change workspaces, click on a tab in the Workspace toolbox, or choose from the list of workspaces in the **Environment** menu. Changing your workspace does not affect your project; it just changes the way information is presented on the screen.

More options for workspaces

For more information on how to create, edit, clone, and delete workspaces, and to find out how to change workspace options, refer to Chapter 15, "More about Projects and Workspaces."

Generating an Application Framework

4

This chapter introduces application frameworks and the steps necessary to generate and build on such a framework. This process uses two tools: AppExpress and ClassExpress.

Before reading this chapter, you may want to look at the material in Chapter 3 on managing projects and workspaces in the IDDE. AppExpress creates both an application framework (according to specifications you provide) and the project in which that framework resides.

Chapter 17, “More about AppExpress,” and Chapter 18, “More about ClassExpress,” are the reference chapters for the AppExpress and ClassExpress tools. Refer to them when you are ready to explore these tools’ features in more depth.

What Is an Application Framework?

An application framework is a standardized skeleton architecture for an object-oriented application. The framework is composed of C++ classes derived from base classes in the Microsoft Foundation Class (MFC) library.

Using a framework to build an application dramatically shortens its development time. All the files needed to create the application are included in the skeleton program. Standard user interface components such as windows, menus, and toolbars are already defined. Some of the necessary connections between the defined C++ classes are established automatically.

With a framework as a starting point and using the MFC library, you can build many different types of applications by:

- Adding or changing user interface components with ResourceStudio (see Chapter 7, “Adding Look and Feel with Resources”)
- Creating new C++ classes, class methods, and class member variables with ClassExpress
- Writing code

As you build application frameworks with AppExpress and ClassExpress, you will become familiar with message maps. Message maps summarize, within a data structure or table, all of the links between Windows messages and the methods (also called functions) of a particular class that process those messages. Each entry in a message map is a pair, consisting of a message identifier and a method that responds to that message. The method is said to handle the message. Methods referenced in message maps are also called message handlers, or simply handlers.

Because message maps are used by MFC to route Windows messages to the messages’ handlers, they provide the essential translations needed to present the event-driven model of the Windows API in an object-oriented guise. By automating the creation and maintenance of message maps, AppExpress and ClassExpress relieve you of much error-prone drudgery, and allow you to spend more of your development time writing code that implements functionality.

The following sample message map was automatically generated by AppExpress for an MDI-style application framework:

```
BEGIN_MESSAGE_MAP(CMainView, CView)
   //{{AFX_MSG_MAP(CMainView)
   // NOTE - ClassExpress will add and
   //remove mapping macros here.
   // DO NOT EDIT what you see in these
   // blocks of generated code !
   //}}AFX_MSG_MAP
   // Standard printing commands
   ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
   ON_COMMAND(ID_FILE_PRINT_PREVIEW,
              CView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

Creating a Framework with AppExpress

This section describes how to launch AppExpress and use it to generate an application framework.

Launching AppExpress

From the IDDE main window, choose **AppExpress** from the **Tools** menu. This opens the window shown in Figure 4-1.

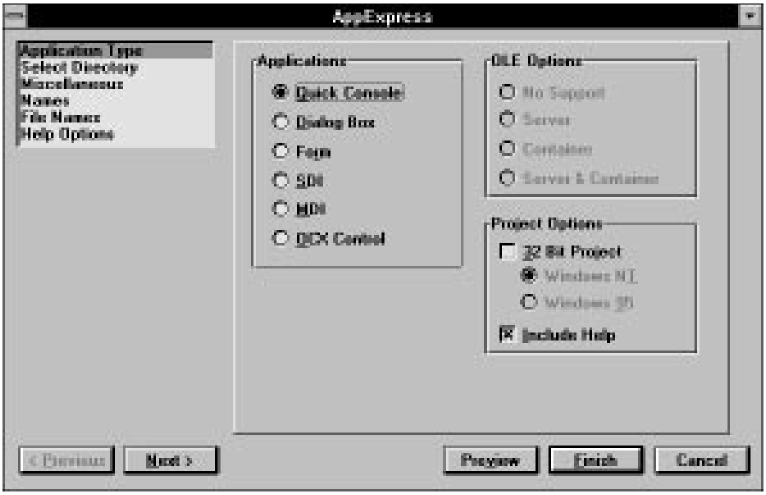


Figure 4-1 AppExpress window

Looking at the AppExpress window

The AppExpress window contains:

- A listbox at the upper left displaying the steps required to create an application framework. The current selection from this list determines which page of options is shown in the larger pane to the right.

- A pane on the right showing the options associated with the currently selected step.

- Buttons below the panes that you can use to navigate among the steps, preview your work, or generate a framework.

To navigate through the six pages of options, click on the name of a step in the steps list, or click on the Next or Previous button. Alternatively, press Control-*n*, where *n* is a number between 1 and 6 representing the selected step's position in the steps list.

Specifying an application framework

AppExpress divides the process of building a framework into six steps:

- Select an application type
- Select a directory for the project
- Provide copyright information and project options
- Specify class names
- Specify source file names
- Specify help file names

These steps, which are briefly outlined here, do not have to be completed in this order. However, you will find it convenient to complete the first step—specifying application type options—before proceeding to any other step. All later steps contain some options that depend upon your choices in this step.

Selecting an application type

To select an application type:

1. Select Application Type from the steps list in the AppExpress window.

2. Select an application type by clicking on a radio button in the Applications group. There are six categories of applications, two of which—SDI and MDI—can be made OLE clients and/or servers.

Check the Include Help box if you want AppExpress to generate files from which a Windows help file can be created. Check the 32-Bit Project box if you are building a 32-bit application. See Chapter 17, "More about AppExpress" for the details of how the 32-Bit Project check box combines with the selected application type to determine what AppExpress generates.

Selecting application type options sets default options for many of the other steps in AppExpress**.** For example, if you select Dialog Box, AppExpress automatically creates three C++ classes with default class names in your skeleton program. You can change these defaults; for example, you can modify the default class names by selecting Names from the steps list.

Selecting a directory for the project

To select a directory for your source files and project file:

1. Select the Select Directory item from the steps list.
   The Select Directory options page opens as shown in Figure 4-2.

2. Select the appropriate drive and directory. Or, click on Create New Directory to create and name a new directory. AppExpress suggests the project name as the new directory name, but you may change that default.

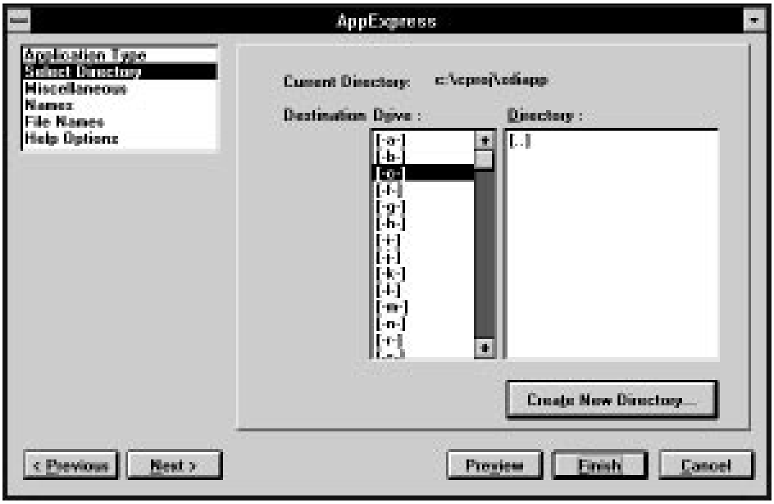
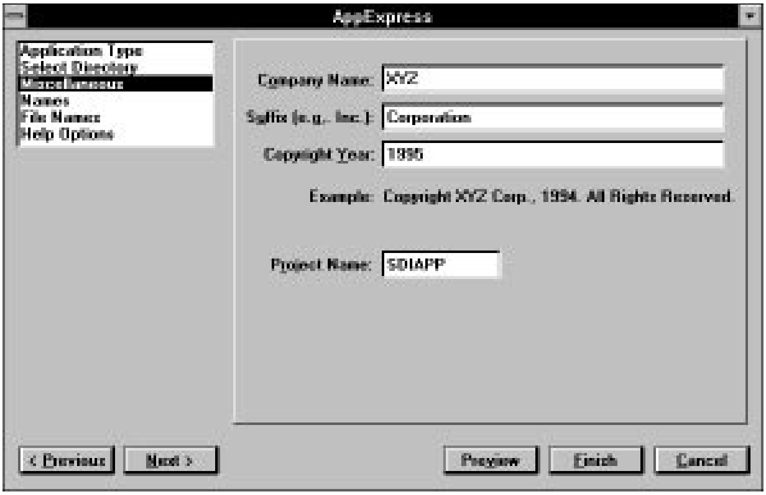


Figure 4-2 Select Directory options

Providing copyright information and project options

You can supply copyright information for your source code, and set the project's name, stack size, and heap size as follows:

1. Select Miscellaneous from the steps list. The Miscellaneous options page opens as shown in Figure 4-3.



Company Name: XYZ
Suffix (e.g., Inc.): Corporation
Copyright Year: 1995
Example: Copyright XYZ Corp., 1994. All Rights Reserved.
Project Name: SDIAPP

Figure 4-3 Miscellaneous options

2. Provide copyright information in the appropriate fields. AppExpress uses this information to write a copyright notice in the comment header of all of your application's source files, and to construct the application's **About** dialog box.

3. Specify the project name in the appropriate field. Stack and heap sizes can be changed later in the **Project Settings** dialog box.

Specifying class names

To view and change the names of C++ classes or their associated source files:

1. Select Names from the steps list. The Names options page opens as shown in Figure 4-4.

2. In the options pane, select a class from the Name drop-down list automatically created by AppExpress. The class names included in this list depend on the application type you have selected.

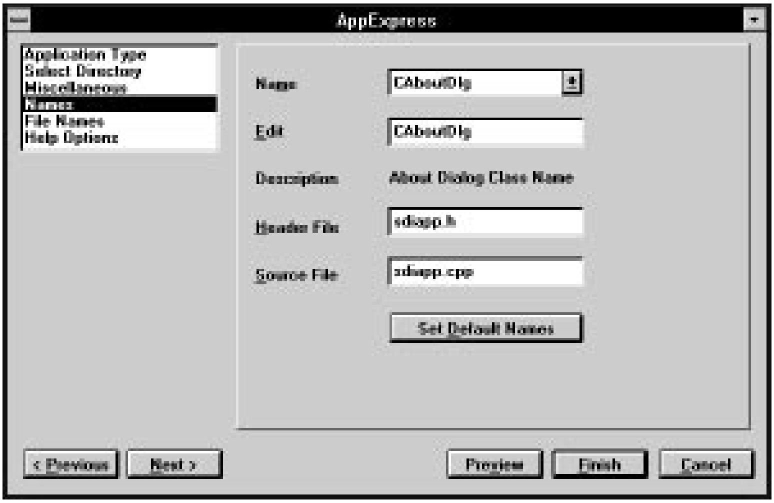
3. If you like, you can edit the selected class name. Note, however, that C++ class names follow a standardized naming convention. Using the default names allows your program's structure to be easily understood by others.

4. In the appropriate field, type the names of the header and source files in which AppExpress should place the class source code.

Note

The CAboutDlg class, which represents your application's About Box class, is always created in the same header and implementation files as the CWinApp-derived class (for example, CSDIAPP).

To discard any changes you have made on this page, click on the Set Default Names button.



Name: CAboutDlg
Edit: CAboutDlg
Description: About Dialog Class Name
Header File: sdiapp.h
Source File: sdiapp.cpp

Figure 4-4 Names options

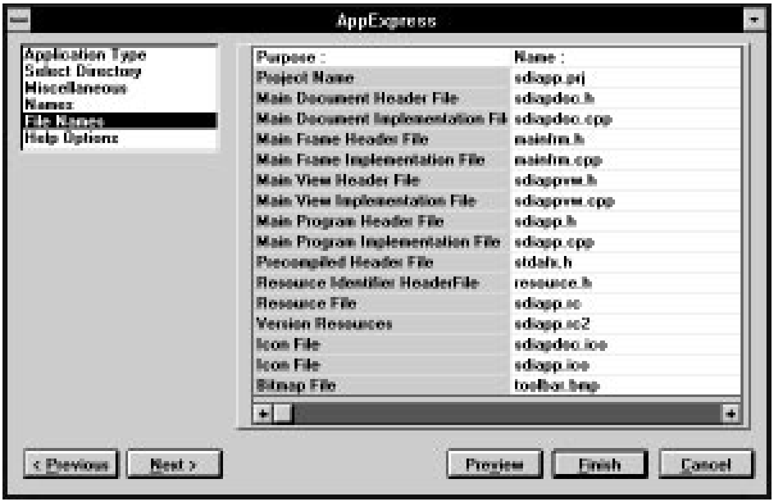
Naming source files

To edit filenames automatically generated by AppExpress:

1. Select File Names from the steps list. The File Names options page opens as shown in Figure 4-5.

2. In the options pane, click on the filename you want to change. The files listed depend on the application type you have selected. Their names correspond to their functionality within the application.

3. Edit the filename, but remember that these changes could make it harder for someone else to identify the purpose of the file.



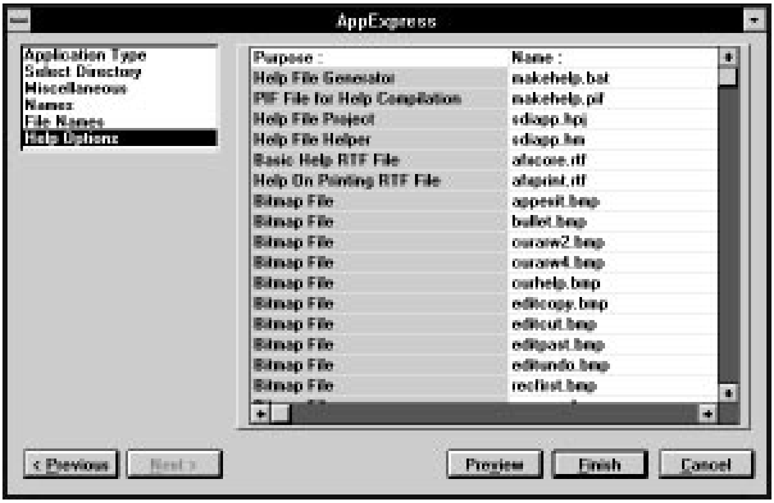
Purpose :	Name :
Project Name	sdiapp.prj
Main Document Header File	sdiapdoc.h
Main Document Implementation Fil	sdiapdoc.cpp
Main Frame Header File	mainfrm.h
Main Frame Implementation File	mainfrm.cpp
Main View Header File	sdiappvw.h
Main View Implementation File	sdiappvw.cpp
Main Program Header File	sdiapp.h
Main Program Implementation File	sdiapp.cpp
Precompiled Header File	stdafx.h
Resource Identifier HeaderFile	resource.h
Resource File	sdiapp.rc
Version Resources	sdiapp.rc2
Icon File	sdiapdoc.ico
Icon File	sdiapp.ico
Bitmap File	toolbar.bmp

Figure 4-5 File Names options

Specifying help file names

To view or change the names of the files that the help compiler uses to generate online help files for your application:

1. Select Help Options from the steps list. The Help Options options page opens as shown in Figure 4-6.



Purpose :	Name :
Help File Generator	makehelp.bat
PIF File for Help Compilation	makehelp.pif
Help File Project	sdiapp.hpj
Help File Helper	sdiapp.hm
Basic Help RTF File	afxcore.rtf
Help On Printing RTF File	afxprint.rtf
Bitmap File	appexit.bmp
Bitmap File	bullet.bmp
Bitmap File	curarw2.bmp
Bitmap File	curarw4.bmp
Bitmap File	curhelp.bmp
Bitmap File	editcopy.bmp
Bitmap File	editcut.bmp
Bitmap File	editpast.bmp
Bitmap File	editundo.bmp
Bitmap File	recfirst.bmp

Figure 4-6 Help Options options

Note

If you selected Quick Console or Dialog Box as the application type, or if Include Help is not selected in the Application Type options page, then you have no help options.

---

2. Edit the filenames if you want. Note, however, that any changes could make it harder for someone else to identify the purpose of the file.

Generating an application framework

At this point, you have selected an application type and the directory in which the application project files will reside, and you may have customized various class names or filenames. AppExpress can now generate your application framework in the form of a skeleton program.

Click on the Finish button in the AppExpress window. AppExpress generates the project and its source files, as you specified, hands it off to the project system, and closes.

Note

> If you checked the Include Help box among the Application Type options, then you will need to build the help file for your application. You do this by running the `makehelp.bat` file that AppExpress creates in your project directory.

Your skeleton program is ready. You can now build the program from within the IDDE, or you can add to your program by using other Symantec C++ tools. The next section shows how to enhance your application's C++ classes by using ClassExpress.

Building on a Framework with ClassExpress

AppExpress cuts the work involved at the beginning of the application-building process. ClassExpress, on the other hand, enhances productivity throughout the rest of the process. You use this tool to flesh out the skeleton application produced by AppExpress.

Specifically, you can use ClassExpress to:

- Write message maps
- Create new classes derived from existing classes
- Create new class methods mapped to specific messages
- Create new class member variables (that is, data variables that are members of a class) mapped to specific user-interface objects
- Create new classes that can be OLE2 automation servers or clients
- Create new classes that act as interfaces to Visual Basic Custom Controls (also known as VBXs)

This chapter describes the first two of these options: writing message maps and creating new classes derived from existing classes.

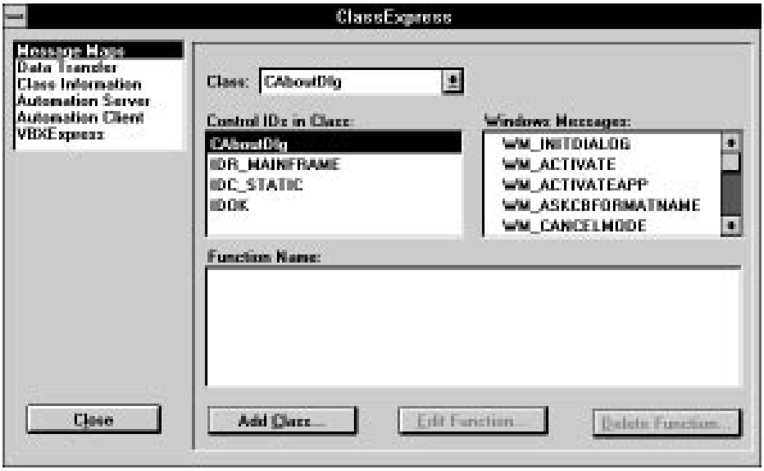
Message maps, defined in “What Is an Application Framework?” in this chapter, are discussed in greater detail in Chapter 17, “More about AppExpress.”

You can use ClassExpress immediately after creating an application framework as well as at later points in development. For example, you probably want to wait until you have created some dialog boxes with ResourceStudio (see Chapter 7, “Adding Look and Feel with Resources”) before mapping class member variables to user interface objects.

Launching ClassExpress

From the IDDE main menu, you launch ClassExpress by choosing **ClassExpress** from the IDDE’s **Tools** menu.

Launching ClassExpress opens the window shown in Figure 4-7.



Class: CAboutDlg

Figure 4-7 ClassExpress window

Looking at the ClassExpress window

The ClassExpress window contains:

- A listbox at the upper left presenting the different kinds of code that ClassExpress can generate
- A pane on the right showing the page of options associated with each selection from the list at the upper left

To navigate through the selections, click on the name of a selection in the list. Alternatively, use the key combination Control-*n*, where *n* is a number between 1 and 6 representing the position in the selections list of the selection to which you want to move.

Writing a message map with ClassExpress

Message maps are tied to particular C++ classes. In ClassExpress, you can add to an existing message map by adding new linkages between messages and class methods. Or you can add a new message map for an entirely new class.

You can launch ClassExpress from within the IDDE or as a stand-alone application. If you launch it from within the IDDE, ClassExpress will be loaded with the IDDE's open project if there is one; otherwise, you are prompted to open an existing project (which also opens the project in the IDDE's project system).

As an example, the following steps demonstrate how to add a message handler to the CAboutDlg class of an SDI application generated by AppExpress:

1. Select Message Maps from the listbox. (If you just started ClassExpress, this should already be selected.) The ClassExpress window is displayed as shown in Figure 4-7.

2. Select the class CAboutDlg from the Class drop-down list. The contents of the three other lists in the options pane are updated to reflect the following selection:

- Control IDs in Class: This list contains all of the menu links and control IDs for which this class can add handlers. For example, IDOK is a universally defined constant for an OK button. Because the CAboutDlg class includes an OK button by default, the IDOK constant is always predefined for this class.

  Because this class represents a dialog box—which itself can respond to many Windows messages—the first item in the Control IDs in Class list is the name of the class itself (CAboutDlg).

- Windows Messages: This list contains all of the Windows messages that apply to the selected item in the Control IDs in Class list. For example, if you highlight IDOK in that list, two applicable messages (button notifications, in this case) are displayed in the Windows Messages list: `BN_CLICKED` and `BN_DOUBLECLICKED.` If a handler is defined for a message, a red box appears before the message name in this list.

- Function Name: This list contains all of the handlers that exist within the selected class. These are the methods that are linked to Windows messages.

3. Select CAboutDlg from the Control IDs in Class list. The list of Windows messages changes, revealing a long list of messages to which your dialog box could respond.

4. Double-click on `WM_ACTIVATE`. Notice that the method OnActivate (in its fully prototyped form) is added to the Function Name list.

Now, whenever your About dialog is activated, the `OnActivate` method is called in response to the dialog window receiving the message `WM_ACTIVATE`. With the four above steps, you have created an association between the message `WM_ACTIVATE` and the method `OnActivate`, and added it to the message map for the CAboutDlg class.

Adding a new class to your application

Adding new classes to your application is as easy as creating a message map. To add a class:

1. Select Message Maps from the listbox. (If you just loaded ClassExpress, then this should already be selected.)
2. Click on the Add Class button. The Add Class dialog box shown in Figure 4-8 opens.

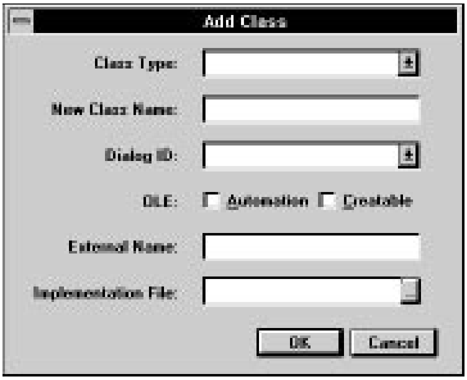


Figure 4-8 Add Class dialog box

3. Select a class type from the Class Type drop-down list. This type specifies the base class from which your new class will be derived. The list displays names of MFC classes without the initial letter ‘C’, thereby allowing you to navigate rapidly within the list by typing the first letter of a class type. For example, typing ‘V’ selects the View class type.
4. The New Class Name field becomes active, displaying a suggested name for the new class. Edit this name as appropriate.
5. If you selected CDialog or CFormView, then the field Dialog ID is displayed. Move to this field and choose the resource ID of the dialog that you want to associate with the new class’s window.

6. Check the OLE Automation box if you want your new class to be a programmable OLE object. If you choose this option, any other Windows application that is an automation client can use the OLE interface you define for this object.

7. If you check the OLE Automation box and you are deriving a new class from the CCmdTarget or CWnd class, the Creatable check box is displayed. Check this box if you want other applications to be able to create the OLE automation object that you are defining.

8. If you check the OLE Automation check box and you are deriving a new class from the CCmdTarget or CWnd class, you must also type a name in the External Name field. This is the name that is exposed to other applications that may want to use your OLE automation object.

9. Decide whether to edit the implementation filename of your new class (shown in the last field in the Add Class dialog box). By default, the base part of the filename is the name of your new class without the initial letter ‘C’, and truncated if necessary.

10. Click OK to add the new C++ class to your application. If you want, you can create message mappings for your new class by using the steps outlined in the previous section.

This section on ClassExpress showed how easy it is to write a message map and to create a new class. By just clicking on a check box, you can also add OLE automation support to your application.

In addition, ClassExpress can bind your classes to your dialog boxes and generate C++ wrapper classes for Visual Basic custom controls (VBXs). It also provides extensive support for building OLE servers and containers. For more information on these features, refer to Chapter 18, “More about ClassExpress.”

Defining Classes and Their Hierarchies

5

Once you have produced an application framework, you can customize the application by adding your own classes and functions. Symantec C++ 7 includes two tools designed specifically for object-programming: the Class Editor and the Hierarchy Editor. These tools simplify the design and maintenance of C++ projects by letting you work directly with the project's class hierarchy and members. The editors themselves take care of many of the mundane details of file-oriented program development, such as opening and closing files and locating source code for particular member functions.

These tools are suited for developing a class hierarchy from scratch, for adding classes to an application framework generated with AppExpress, and for browsing and editing pre-existing class hierarchies. They are especially useful for understanding the architecture of unfamiliar source code.

The Class and Hierarchy Editors can perform the same operations on classes and hierarchies; they differ only in their interface. The Class Editor's interface emphasizes member editing; the Hierarchy Editor's interface, through its graphical display, emphasizes inheritance relationships. You can use one or the other, or both simultaneously, according to preference.

This chapter describes basic operations that may be performed with the Class and Hierarchy Editors. For a complete reference on these two tools, see Chapter 19, "Class Editor Reference," and Chapter 20, "Hierarchy Editor Reference."

Parsing and Browsing

The Class and Hierarchy editors are C++ class browsers. These tools let you work with your source code in an object-oriented manner, rather than in the traditional file-oriented manner. The Class and Hierarchy Editors present you with a list of classes in your project, allow you to view the members of each class, and let you edit the member declarations and definitions directly, without performing the overhead of opening and closing files and locating source code for particular members. Changes made to classes or inheritance relationships in the Class or Hierarchy Editors are automatically changed in the underlying source code.

To present you with a list of classes and members, the IDDE must parse the source code. By default, this is done automatically. When the IDDE detects that source code or project settings have changed since the last parse, it reparses the necessary files. If errors are encountered during parsing, messages are displayed in the Output window. You can double-click on the error message to open for editing the appropriate file at the point at which the error was detected.

How the class browsers expand macros

The parser used by the Class and Hierarchy Editors contains a fully functional C preprocessor. One significant difference between this parser and the Symantec C compiler is the way the parser expands macros.

The parser treats an entire project as one "database" of code. Consequently, preprocessor macros defined in any header will expand in any subsequently parsed file. This can produce unexpected errors. For example:

```
// FOO1.H -----------------------------------
#define pBar (pIndirect->pBar)
// . . .

// FOO1.CPP ---------------------------------
#include <foo1.h>

// FOO2.CPP ---------------------------------
struct A
{
int * pBar; // ERROR! Expands to:  int *
(pIndirect->pBar);
};
```

To avoid errors of this type, undefine the macro at the top of the file where the error occurs (FOO2.CPP).

Browsing library source code

If you are browsing source code that is based on MFC or some other large class library, you may find it convenient to turn off the display of library classes. To do so:

1. Choose **Settings** from the IDDE's **Project** menu.
2. In the **Project Settings** dialog box, click the Directories tab.
3. In the Browser Exclude Directories textbox, type the name of the directory containing the class library headers (for example, `c:\sc\mfc\include`).
4. Click OK.

You can control the parsing of individual files in the Project window, and enable or disable parsing for the entire project in the Target page of the **Project Settings** dialog box. For more information, see Chapter 15, "More about Projects and Workspaces."

Class Editor

The Class Editor provides a list-based view of the class hierarchy. From within the Class Editor you may add classes, modify inheritance relationships, and view and edit class member declarations and definitions.

To open a Class Editor window, do one of the following:

- Choose **Class Editor** from the **Window** menu's **Goto View** submenu.
- Click and drag the Class Editor icon from the Views toolbox to an empty area of the desktop.

The Class Editor window (see Figure 5-1) is divided into three panes:

- The Classes pane, which contains a list of classes.
- The Members pane, which contains a list of members of the current (highlighted) class.
- The Source pane, which displays member source code. You can edit the source code in this pane as in the

Source window (see Chapter 6, “Editing Program Code,” for a description of text editing features).

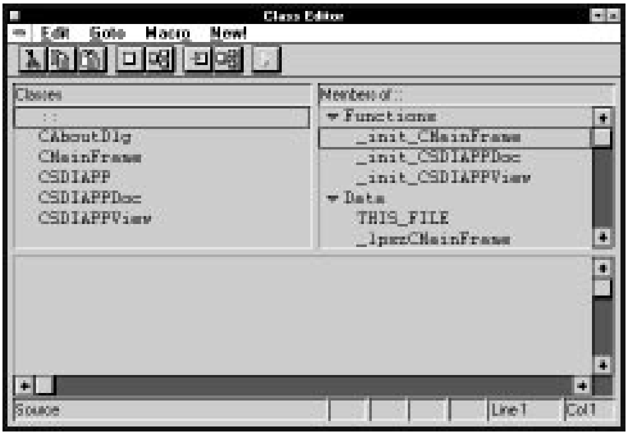


Figure 5-1 Class Editor window

The Class Editor window contains a standard menu bar and toolbar; pop-up menus are available in each pane. You can set options for the grouping, sorting, and display of classes and members in the Classes and Members panes in the **Editing/Browsing Settings** dialog box. For a complete reference on Class Editor menus and options, see Chapter 19, “Class Editor Reference.”

You can change the relative sizes of panes in the Class Editor window by first positioning the cursor over the line separating the panes. The cursor changes to a two-headed arrow. Press the left mouse button and drag the separator to the desired location.

To select a class in the Classes pane, click on it. Additional classes may be selected by holding down the Control key and clicking. By default, classes are displayed hierarchically, with derived classes below and indented relative to their bases. Classes with multiple bases are displayed beneath each of the bases. Triangular buttons to the left of base classes can be used to collapse and expand branches of the hierarchy. You can set an option to display classes alphabetically in the **Editing/Browsing Settings** dialog box.

By default, class members in the Members pane are grouped into Functions, Data, and Typedefs. The tree structure can be expanded or collapsed by clicking on the triangular buttons to the left of the category names. To select a class member, click on it. Additional members can be selected by clicking while holding down the Control key. You can display a member declaration or definition in the Source pane by double-clicking on the member name.

Creating classes

You can create a class hierarchy consisting of new classes related to other classes by inheritance (derived or sibling classes), or create new top-level classes not related to any other class.

Creating a top-level class

To create a new class hierarchy, first create a top-level class to serve as a base for the hierarchy.

To create a new top-level class:

1. Choose **Add Top** from the pop-up menu in the Classes pane. The **Add Class** dialog box is displayed (Figure 5-2).

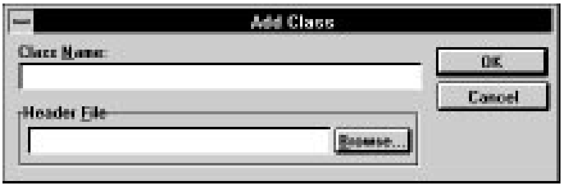


Figure 5-2 Add Class dialog box

2. Type a name for the new class.
3. Click OK.

The class declaration is placed in a new header file, and the header file is added to the project. By default, the first eight letters of the class name are used to derive the header filename. You may change the header filename by typing an alternative name in the Header File textbox of the **Add Class** dialog box.

Creating a derived class

After a base class exists, you may create derived classes—specialized versions of the base class.

To create a new derived class:

1. Select the base class of the new class.

2. Choose **Add Derived** from the pop-up menu in the Classes pane. The **Create Derived Class** dialog box opens. (This dialog box is similar to the **Add Class** dialog box.)

3. Type a name for the new class.

4. Click OK.

Creating a sibling class
You may also create a new derived class as a “sibling” to an existing class. Siblings have the same base class with the same access specifiers.

To create a new sibling class:

1. Select the class whose base class will be the base class of the new class.

2. Choose **Add Sibling** from the pop-up menu in the Classes pane. The **Create Derived Class** dialog box is displayed.

3. Type a name for the new class.

4. Click OK.

Editing inheritance relationships

As your application evolves, you may want to restructure your class hierarchy. The Class Editor lets you add and delete connections between classes, as well as change the inheritance attributes.

When altering inheritance relationships, the Class and Hierarchy Editors change only the class declarations. In particular, they do not change references to base classes and their members in a derived class’s constructors or functions. If such references exist, you must change them manually in the source file.

Connecting to a base class
You can connect a class to a base class to make it a derived class of this base.

To make one existing class derive from another existing class:

1. Select the class that will be the derived class.

2. Choose **Connect Base** from the pop-up menu in the Classes pane. The **Add Base** dialog box opens (Figure 5-3).

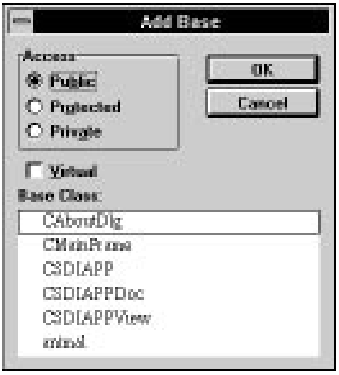


Figure 5-3 Add Base dialog box

3. From the listbox, select the class (or classes) to be this class's base class.

4. Select the desired access specifier. Check Virtual if virtual inheritance is desired.

5. Click OK.

If the classes are displayed hierarchically, the derived class is displayed below and is indented relative to the base class in the Classes pane.

Deleting a connection

You can also disconnect a derived class from a base class.

To delete an inheritance relationship:

1. Select a derived class.

2. Choose **Delete Base Connection** from the pop-up menu in the Classes pane.

3. A message box requests confirmation. Click Yes to delete the base connection.

In cases of multiple inheritance, select one inheritance relationship to delete by clicking on the instance of the derived class below the base to be removed. For classes derived from more than one base class, multiselecting the inheritance relationships to be deleted lets you delete all base connections simultaneously. If a class has all of its base connections severed, it remains in the hierarchy at the top level.

Editing inheritance attributes

You can change a derived class's base class access specifier and virtual inheritance flag.

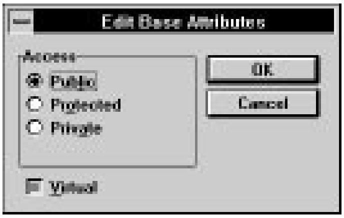
To edit the base class access specifier:

1. In the Classes pane, select the derived class.

   In cases of multiple inheritance, select the base connection to edit by clicking on the instance of the derived class below the appropriate base.

2. Choose **Edit Base Attributes** from the pop-up menu in the Classes pane.

   The **Edit Base Attributes** dialog box opens (Figure 5-4).



● Public
○ Protected
○ Private
▨ Virtual

Figure 5-4 Edit Base Attributes dialog box

3. Select the desired access specifier. Check Virtual if virtual inheritance is desired. Click OK.

If you are editing several inheritance relationships simultaneously and the access specifiers are not identical, a Don't Change option is displayed. This lets you change the Virtual specifier without also affecting the access specifiers. You may, however, select Public, Protected, or Private, and all connections are given that access specifier.

Working with class members

After creating a class, you can implement its functionality through member data and functions. You may add, delete, and edit class members through the Members pane and Source panes.

A list of the members of the currently selected class is shown in the Members pane (Figure 5-5). By default, the member list is sorted into three categories: Data, Functions, and Typedefs. Within each category, items are sorted alphabetically. The colored diamond in front of each member identifies the access as public (green), protected (yellow), or private (red). Names of members defined as macros appear in red.

Note

> Members that appear in red are not syntactically verified before source changes are saved back to the file.

The lists following each category header can be collapsed or expanded by clicking on the triangular button to the left of the category name.

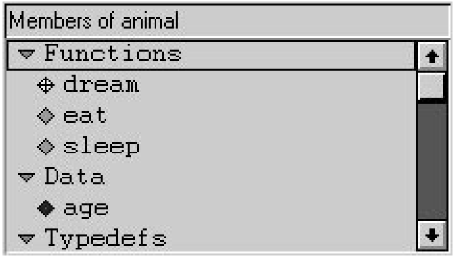


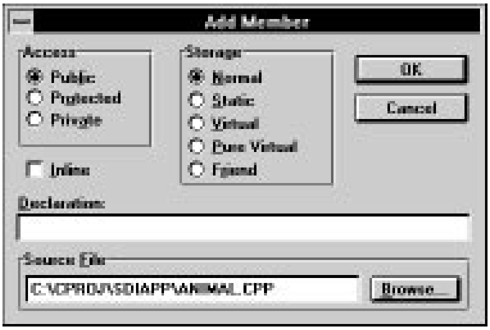
Figure 5-5 Members pane of Class Editor

Adding a class member

The first step in class implementation is to declare the data members and member functions.

To add a class member:

1. Choose **Add** from the pop-up menu in the Members pane. The **Add Member** dialog box opens (Figure 5-6).



Access: ● Public

Storage: ● Normal

☐ Inline

Source File: C:\VCPROJ\SDIAPP\ANIMAL.CPP

Figure 5-6 Add Member dialog box

2. Type the member declaration. The member may be a data item (for example, `int nCats`), a function (for example, `int NumCats()`), or a typedef (for example, `typedef int CATCOUNT`). Do not type any storage specifiers into the declaration textbox; these are added by the Class Editor.
3. Select the desired access and storage specifiers. Check the Inline check box for an inline function.
4. Click OK.

The new member is added to the appropriate group in the Members pane and to the class declaration in the header file. If the new member is a function, an empty function is added to the appropriate source file. If a new source file is created, it is added to the project.

By default, the first eight letters of the class name are used to derive the source filename for new functions. You may change the file name by entering an alternative name in the Source File textbox of the **Add Member** dialog box.

Deleting a class member

Unneeded class members are easily removed.

To delete a class member:

1. Select the class member to be deleted.

2. Choose **Delete** from the pop-up menu in the Members pane.

3. When the message box requests confirmation, click Yes to delete the member.

The declaration is removed from the header file and, if the member was a function, the function definition is removed from the source file.

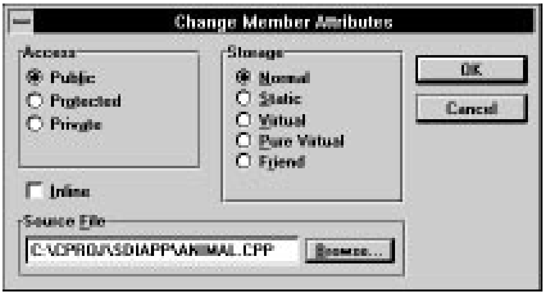
Changing member attributes

As your class and hierarchy evolve, you may decide to change certain member attributes. For example, you may want to make a function virtual, or data private.

To change a member's access and storage specifiers:

1. Select the member you want to change.

2. Choose **Edit Attributes** from the pop-up menu in the Members pane. The **Change Member Attributes** dialog box opens (Figure 5-7).



Access: ● Public

Storage: ● Normal

☐ Inline

Source File: C:\CPROJ\SDIAPP\ANIMAL.CPP

Figure 5-7 Change Member Attributes dialog box

3. Select the desired access and storage specifiers. Check the Inline check box for an inline function. Click OK.

The class declaration in the header file is modified and the Members display is updated to reflect those changes.

If you are editing several members' attributes simultaneously and the access specifiers are not identical, a Don't Change option is displayed in the Access group box. Likewise, if the storage specifiers are not identical, a Don't Change option is displayed in the Storage group box. These options let you change other member attributes without affecting the original access or storage of each member. You may, however, select a particular access or storage specifier, and all members are given that attribute. Also, when you are editing several members' attributes, if the inline attributes are not the same, the Inline check box changes to allow three states (checked, unchecked, and grayed, indicating "Do not change") instead of the normal two-state options.

Viewing and editing member source

After declaring your class members, you can extend your functions by editing the function definitions in the Source pane. You also can change data member types, array dimensions, and so on.

To view and edit member declarations and function definitions:

1. Double-click on the member in the Members pane.

   The member's declaration (for data items and pure virtual functions) or definition (for other functions) is displayed in the Source pane. If the source code is not available (as with MFC libraries, for example), the definition is displayed.

2. Click in the Source pane to begin editing. Editing operations here are identical to editing operations in other Source windows.

3. To save changes, choose **Save** from the pop-up menu in the Source pane.

Changes made to a function's argument and return types in the definition are updated automatically in the class declaration. The Members pane is updated when the source is saved.

Note

> If you close the current project before you save a source file you've modified with the Class Editor, you cannot discard the changes or close the file until you reopen the project.

Viewing and editing source files

At times it is useful to view and edit the entire file containing the class declaration or the class's functions. It is necessary, for example, to add the appropriate `#include` statements to the source files before compiling.

To view and edit the header file containing the class declaration:

1. Select the class in the Classes pane.
2. Choose **Show Header** from the pop-up menu in the Classes pane. A Source window containing the contents of the class header file is displayed.
3. Edit the class declaration as desired.

   You may add or delete class members as you would without the Class Editor. However, if you add functions to the declaration, you have to add the function definitions manually to the source file as well.
4. Save your changes by choosing **Save** from the Source window's **File** menu.
5. To close the Source window, choose **Close** from the **File** menu.
6. Click on the Class Editor window. If you have made changes to the header file, a message box asks whether it is OK to reparse. Click Yes.
7. Click on the class whose declaration you just edited to see the member changes updated in the Members pane.

Note

For non-inline functions, if you change the function's signature (return type, argument types, and so forth) in its declaration, you must make the corresponding changes to its definition by hand, or the Members pane will display two versions of the function.

To view and edit the source file containing member function definitions:

1. Select a member function from the Members pane.
2. Choose **Show Source** from the pop-up menu from the Members pane. (**Show Source** is dimmed if the source code is not available.)

   The file containing the member function's source code is opened for editing in a Source window.
3. Edit the member functions as desired.

   You may edit member functions as you would without the Class Editor. However, if you change function arguments or return types, or add new functions, you must modify or add the function declarations manually in the header file as well.
4. Save your changes by choosing **Save** from the Source window's **File** menu.
5. To close the Source window, choose **Close** from the **File** menu.
6. Click on the Class Editor window. If you have made changes to the source file, a message box asks whether it is OK to reparse. Click Yes.

Hierarchy Editor

The Hierarchy Editor provides a graphical view of the class hierarchy. From within the Hierarchy Editor, you may add classes, modify inheritance relationships, and view and edit class member declarations and definitions. With its graphical view of class relationships, the Hierarchy Editor is an especially useful tool when examining unfamiliar source code for the first time.

To open a Hierarchy Editor window, do one of the following:

- 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 to an empty area of the desktop.

The Hierarchy Editor window opens. By default, only the graphical display of class hierarchies is shown. To provide full functionality, it is necessary to enable the editor's two child windows.

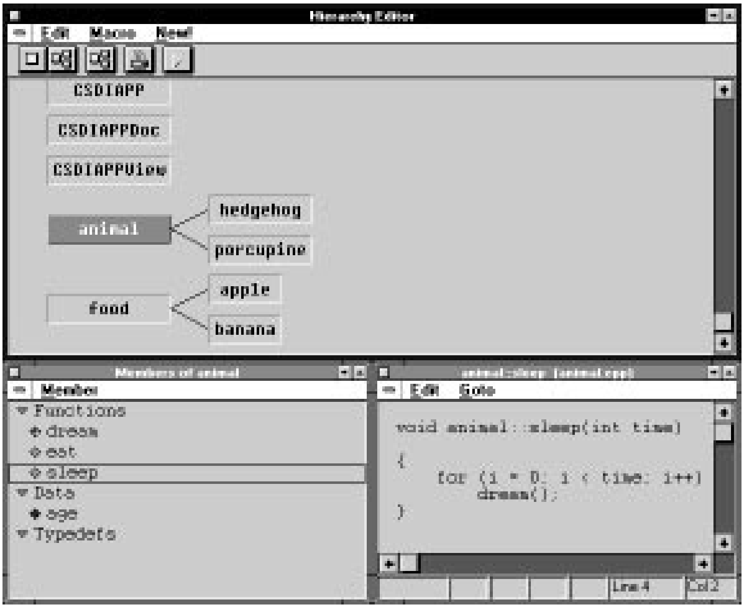
To enable the Hierarchy Editor child windows:

1. Choose **Settings** from the Hierarchy Editor's pop-up menu. The **Editing**/**Browsing Settings** dialog box opens to the Hierarchy page.

2. Check the items labeled Members and Source in the Pop-up Windows group box.

3. Click OK.

The Members and Source child windows open below the Hierarchy Editor window (Figure 5-8).



```
void animal::sleep(int time)
{
    for (i = 0; i < time; i++)
        dream();
}
```

Figure 5-8 Hierarchy Editor window

Unlike the Members and Source panes of the Class Editor, Hierarchy Editor windows are independent. They have their own menus and may be positioned, sized, and closed separately. Otherwise, they behave as do the corresponding panes of the Class Editor. For a complete reference on Hierarchy Editor menus and options, see Chapter 20, “Hierarchy Editor Reference.”

Creating classes

A class hierarchy consists of a number of classes connected by inheritance relationships. You can create new classes related to other classes by inheritance (derived or sibling classes) or new top-level classes not related to any other class.

Operations relating to class creation and the establishment of hierarchical relationships are carried out in the Hierarchy Editor’s graphical display window.

Creating a top-level class

To create a new class hierarchy, first create a top-level class to serve as a base for the hierarchy.

To create a new top-level class:

1. Activate the Hierarchy Editor window.
2. Choose **Add Top** from the pop-up menu. The **Add Class** dialog box opens (Figure 5-9).

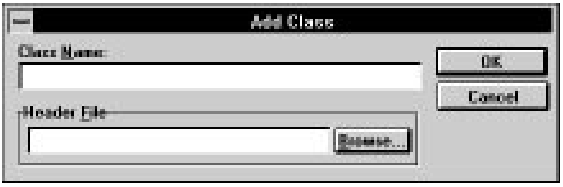


Figure 5-9 Add Class dialog box

3. Type a name for the new class.
4. Click OK.

The class declaration is placed in a new header file, the header file is added to the project, and the new class is selected in the Hierarchy Editor main pane. By default, the first eight letters of the class name are used to derive the header file name. You may change the header file name by typing an alternative name in the Header File textbox of the **Add Class** dialog box.

Creating a derived class

After a base class exists, you may add a specialized version of that class to the hierarchy by creating a derived class.

To create a new derived class:

1. Select the class that will act as the base class of the new class.
2. Choose **Add Derived** from the pop-up menu.

   You can also perform this step with the mouse. Hold the left mouse button down and drag the cursor from the selected class to an empty area of the window. A rubber band line appears as you do this (Figure 5-10). Release the mouse button.

   The **Create Derived Class** dialog box opens (this dialog box is similar to the **Add Class** dialog box.

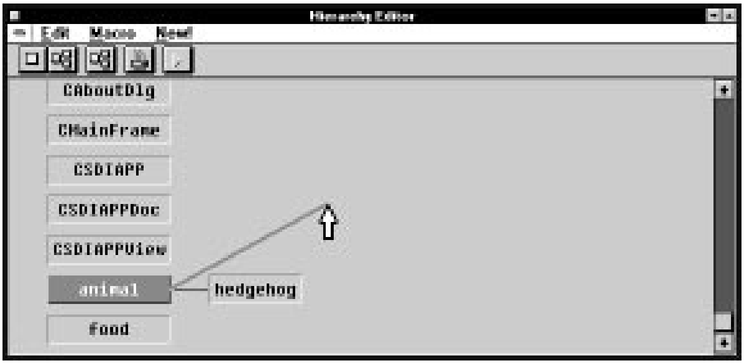


Figure 5-10 Creating a new derived class using the mouse

3. Type a name for the new class.
4. Click OK.

Creating a sibling class

You may also create a new derived class as a “sibling” to an existing class. Sibling classes have the same base class with the same access specifiers.

To create a new sibling class:

1. Click on the class whose base class is to be the base class of the new class.
2. Choose **Add Sibling** from the pop-up menu. The **Create Derived Class** dialog box is displayed.
3. Type a name for the new class.
4. Click OK.

Editing inheritance relationships

As your application evolves, you may want to restructure your class hierarchy. You can add and delete connections between classes, as well as change the inheritance attributes.

In altering inheritance relationships, Class and Hierarchy Editors change only the class declarations, not references to base classes and their members in a derived class's constructors or functions. If such references exist, you must change them manually.

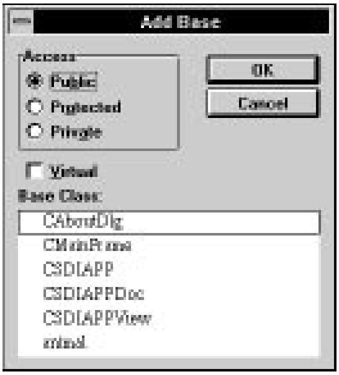
Connecting to a base class

You can connect a class to a base class, to make the former a derived class of the latter.

To make an existing class a base of another existing class:

1. Click on the class that is to be the derived class.

2. Choose **Connect Base** from the pop-up menu. The **Add Base** dialog box opens (Figure 5-11).



Access
● Public
○ Protected
○ Private

☐ Virtual

Base Class:
CAboutDlg
CMainFrame
CSDIAPP
CSDIAPPDoc
CSDIAPPView
[illegible]

Figure 5-11 Add Base dialog box

3. From the listbox, select the class (or classes) that will be the derived class's base class.

4. Select the desired access specifier. Check Virtual if virtual inheritance is desired.

5. Click OK. The display changes to show the new inheritance relationship.

You can also connect to a base by using the mouse in the graphical display. Click on the class that is to be the base class. Holding down the mouse button, drag the rubber-band line to the class that is to be the derived class. (The cursor changes to a universal "No" sign when positioned over a class that cannot become a derived class of this base.) Release the mouse button. The class is publicly derived from the base.

Deleting a connection

Just as you can connect a class to a base class, you also can disconnect a derived class from a base class.

To delete an inheritance relationship:

1. Click on the line connecting the base and derived classes. The line is highlighted to show that it is selected (Figure 5-12).

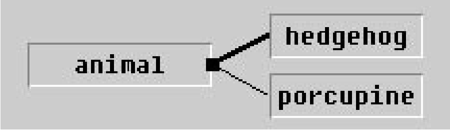


Figure 5-12 Highlighted base-derived connection

2. Choose **Delete Base Connection** from the pop-up menu.

3. When the message box requests confirmation, click Yes to delete the base connection.

Changing base class

You can move a base class connection to change the base of a derived class.

To change a class's base class:

1. Click on the line connecting the derived class and its current base.

   The line is highlighted to indicate that it has been selected. Note that at the end nearest the base class, there is a small black box (referred to as the line's "handle").

2. Click on the line's handle. Holding down the left mouse button, drag the handle over the class that is to be the derived class's new base class.

3. When the message box requests confirmation, click Yes to change the base connection.

Editing inheritance attributes

You can change a derived class's base class access specifier and virtual inheritance flag.

To edit the base class access specifier:

1. Click on the line connecting the base and derived classes. The line is highlighted to show that it is selected.

2. Choose **Edit Base Attributes** from the pop-up menu. The **Edit Base Attributes** dialog box opens (Figure 5-13).

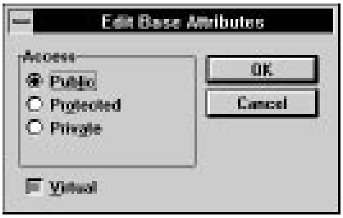


Figure 5-13 Edit Base Attributes dialog box

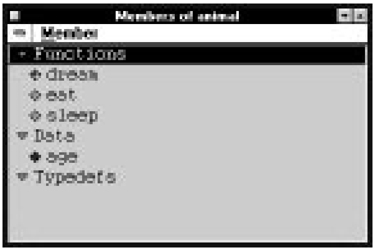
3. Select the desired access specifier. Check Virtual if virtual inheritance is desired. Click OK.

If you are editing several connections simultaneously and the access specifiers are not identical, a Don't Change option becomes available. This lets you change the Virtual specifier without affecting the various access specifiers. You may, however, check Public, Protected, or Private instead, and all connections are given that access specifier.

Working with class members

After creating a class, you can implement its functionality through member data and functions. You may add, delete, and edit class members through the Members and Source child windows.

A list of members of the currently selected class is displayed in the Members child window (Figure 5-14). By default, the member list is sorted into three categories: Data, Functions, and Typedefs. Within each of these categories, items are sorted alphabetically. The colored diamond in front of each member identifies the access as public (green), protected (yellow), or private (red). Lists following each category header can be collapsed or expanded by clicking on the triangular button to the left of the category name.



- Functions
  - dream
  - eat
  - sleep
- Data
  - age
- Typedefs

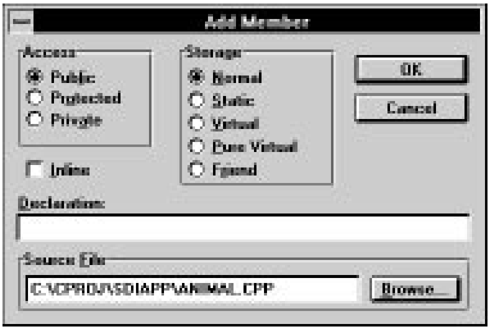
Figure 5-14 Members child window

Adding a class member

The first step in class implementation is to declare the data and function members.

To add a class member:

1. Activate the Members child window.
2. Choose **Add** from the **Member** menu or from the pop-up menu. The **Add Member** dialog box opens (Figure 5-15).



Access: ● Public ○ Protected ○ Private

☐ Inline

Storage: ● Normal ○ Static ○ Virtual ○ Pure Virtual ○ Friend

Declaration:

Source File: C:\VCPROJ\SDIAPP\ANIMAL.CPP

Figure 5-15 Add Member dialog box

3. Type the member declaration. The member may be a data item (for example, `int nDogs`), a function (for example, `int NumDogs()`), or a typedef (for example, `typedef int DOGCOUNT`). Do not type any storage specifiers into the declaration textbox; these are added by the Hierarchy Editor.

4. Select the desired access and storage specifiers. Check the Inline check box for an inline function.

5. Click OK.

The new member is added to the appropriate list in the Members child window and to the class declaration in the header file. If the new member is a function, an empty function is added to the appropriate source file. If a new source file is created, that file is added to the project.

By default, the first eight letters of the class name are used to derive the source file name for new functions. You can change the file name by entering an alternative name in the Source File textbox of the **Add Member** dialog box.

Deleting a class member

Unneeded class members are easily removed.

To delete a class member:

1. Select the class member to be deleted.

2. Choose **Delete** from the **Member** menu or from the pop-up menu.

3. When the message box requests confirmation, click Yes to delete the member.

The declaration is removed from the header file; if the member was a function, the function definition also is removed from the source file.

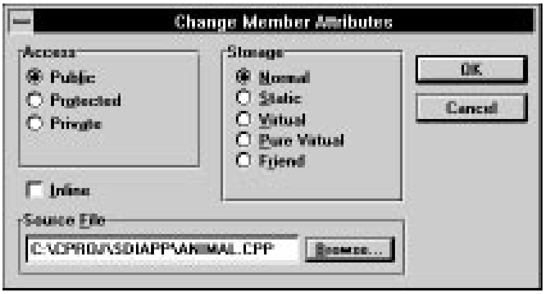
Changing member attributes

As your class and hierarchy evolve, you may need to change certain member attributes. For example, you may want to make a function virtual, or data private.

To change a member's access and storage specifiers:

1. Select the member to be changed.

2. Choose **Edit Attributes** from the **Member** menu or from the pop-up menu. The **Change Member Attributes** dialog box opens (Figure 5-16).



Access
● Public
○ Protected
○ Private

☐ Inline

Storage
● Normal
○ Static
○ Virtual
○ Pure Virtual
○ Friend

Source File: C:\CPROJ\SDIAPP\ANIMAL.CPP

Figure 5-16 Change Member Attributes dialog box

3. Select the desired access and storage specifiers. Check Inline for an inline function. Click OK.

If you are editing several members' attributes simultaneously and their access specifiers are not identical, a Don't Change option becomes available in the Access group box. Similarly, if the storage specifiers are not identical, a Don't Change option is displayed in the Storage group box. These options allow you to change other member attributes without affecting the original access or storage of each member, respectively. You may, however, select a particular access or storage specifier, and all members are given that attribute.

The class declaration in the header file is modified and the Members display is updated to reflect the changes.

Viewing and editing member source

After declaring your class members, you can extend your functions by editing the function definition in the Source child window. Also, you can change data member types, array dimensions, and so on.

To view and edit member declarations and function definitions:

1. Double-click on the member in the Members child window.

   The member's declaration (for data items and pure virtual functions) or definition (for other functions) is shown in the Source child window.

2. Click in the Source child window to begin editing. Editing operations here are identical to those in other Source windows.

3. To save changes, choose **Save** from the pop-up menu.

Changes made to a function's argument and return types are updated automatically in the class declaration. The Members child window is updated when the source is saved.

Viewing and editing source files

At times it is useful to view and edit the entire file containing the class declaration or the class's functions.

To view and edit the header file containing the class declaration:

1. Click on the class in the Hierarchy Editor window.

2. Choose **Show Header** from the pop-up menu.

   A Source window containing the contents of the class header file opens.

3. Edit the class declaration as desired.

   You may add or delete class members as you would without the Hierarchy Editor. However, if you add function declarations, you must add the function definitions to the source file manually as well.

4. Save your changes by choosing **Save** from the Source window's **File** menu.

5. To close the Source window, choose **Close** from the **File** menu.

6. Click on the Hierarchy Editor window. If you have made changes to the header file, a message box asks whether it is OK to reparse. Click Yes.

7. Click on the class whose declaration you just edited to see the member changes updated in the Members window.

To view and edit the source file containing member function definitions:

1. Select a member function from the Members child window.
2. Choose **Show Source** from the **Member** menu or from the pop-up menu.

   The file containing the member function's source code is opened for editing in a Source window.
3. Edit the member functions as desired.

   You may edit member functions as you would without the Hierarchy Editor. However, if you change function arguments or return types, or add new functions, you must modify or add the function declarations manually in the header file as well.
4. Save your changes by choosing **Save** from the Source window's **File** menu.
5. To close the Source window, choose **Close** from the **File** menu.
6. Click on the Hierarchy Editor window. If you have made changes to the source file, a message box asks whether it is OK to reparse. Click Yes.

Editing Program Code

6

Program code is edited in Source windows. The Source window is a full-featured text editor, with many options and features you will find particularly useful when editing C and C++ source code. Subsets of Source window functionality are also available in the Source pane of the Class Editor and in the Source child window of the Hierarchy Editor.

In addition to its role in editing your project's code, you can use the Source window for monitoring program execution while debugging. See Chapter 24, "Commands Available in Debugging Mode," for more information.

Chapter 21, "Text Editor Reference," contains descriptions of all Source window menu commands, toolbar icons, dialog boxes, and options. Refer to it once you've read this chapter and you need more details about specific aspects of the editor.

If you need to know the key combination used to execute a particular editor command, refer to the Symantec C++ IDDE Help. It contains a reference to all key combinations for every key binding set.

Role of the Text Editor

The purpose of the IDDE text editor is to create, examine, and modify your project's source files. Because these files are standard text files, you can, in principle, use any text editor to work with them. However, the IDDE text editor is designed especially to edit C and C++ source files and to work in concert with other IDDE tools.

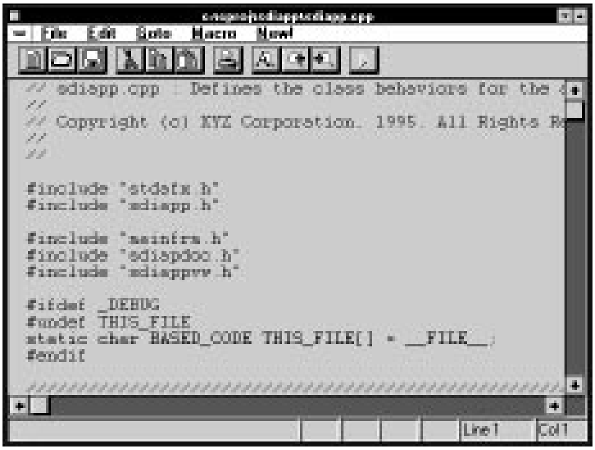
Anyone familiar with Windows can get started quickly with the IDDE text editor because it uses standard Windows editing commands. In addition, the text editor has some features that make working with C and C++ files easier. For example, the editor can automatically indent or unindent after braces and can check delimiters.

In addition, the text editor can display keywords, preprocessor directives, and comments in special font styles and colors. This technique helps track errors in source code while you are editing. For example, an unmatched comment (`/*` without a matching `*/`) turns a large part of the code a different color, making it obvious where the problem lies. Also, keywords and preprocessor directives are easier to spot when they are in a different color or font style. Misspelled keywords can be caught immediately when they remain displayed in the default font.

Source windows are an integral part of the IDDE environment and work together with other IDDE windows to make application development easier. For example, the IDDE automatically saves all files open in Source windows when you rebuild your project. During compilation, error messages are displayed in the Output window; when you double-click on an error message, the IDDE opens a Source window on the corresponding source file, if necessary, and then jumps to the line in the source code that caused the error.

The Source Window

The Source window is shown in Figure 6-1.



```
// sdiapp.cpp : Defines the class behaviors for the
//
// Copyright (c) XYZ Corporation, 1995. All Rights R
//
//

#include "stdafx.h"
#include "sdiapp.h"

#include "mainfrm.h"
#include "sdiapdoc.h"
#include "sdiappvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
```

Figure 6-1 Source window

Most of your work with a Source window is in the editing area. This is where text is displayed and edited, as described later in this chapter. The Source window contains a typical IDDE window menu and toolbar; a pop-up menu is available by right-clicking in the editing area. Complete information about the menus and toolbar is contained in Chapter 21, “Text Editor Reference.”

The status bar, shown in Figure 6-2, provides information about the current state of the file being edited.

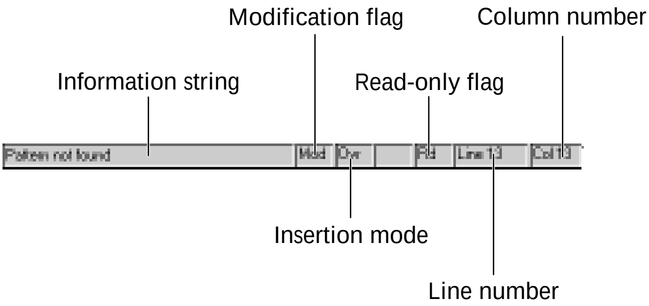


Figure 6-2 Source window status bar

Information string
Displays information about the state of the current function. For example, the message `Pattern not found` is displayed when the **Find** command is unable to locate the pattern you specified.

Modification flag
Displays `Mod` if the file has been modified since it was last saved.

Insertion mode
Displays `Ovr` when the typing mode is set to overtype. When the typing mode is set to insert, nothing is displayed in this field. Toggle the insertion mode with the Insert key.

Read-only flag
Displays `Rd` when the file is set to read-only. When the file is in read/write mode, nothing is displayed in this field. The read-only attribute is set in the **Current Buffer Options** dialog box described later in this chapter.

Line number
Displays the line number of the insertion point.

Column number
Displays the column position of the insertion point.

File Manipulation

The IDDE lets you open as many different files as your computer's memory allows. You can open multiple views of one file; changes made in one window will be made in all.

Creating files

To create a new file, choose **New** from the IDDE's **File** menu. An empty Source window opens, and you can begin typing. Note that the new file is not actually created on disk until you save it.

Opening files

To open a source file (or any text file), choose **Open** from the IDDE's **File** menu, or choose **Open** from a Source window's **File** menu. A standard Windows **File Open** dialog box is displayed. Select the file you want to edit and click OK. The file is opened for editing in a new Source window.

There are additional ways to open a file for editing:

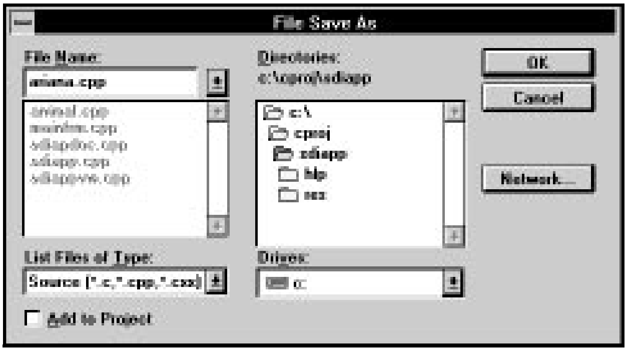
- Choose a file from the list of recently opened files at the bottom of the IDDE's or Source window's **File** menu.
- Double-click on a filename in the Project window.
- Click **New!** in a Source window to open another view of the same file.
- Choose **Load** from the Source window's **File** menu. The file you select opens in the same Source window, replacing the previous file.

Saving files

To save the file in the active window, choose **Save** from the Source window's **File** menu. If the file is untitled, the editor displays the **File Save As** dialog box to let you name it. Otherwise, it saves the file under its current name. This procedure saves only the file in the active Source window.

To save the files in all Source windows, choose **Save All** from the Source window's **File** menu. The editor displays the **Save As** dialog box for any untitled file.

To save an unnamed file or a named file under a new name, choose **Save As** from the **File** menu. The text editor displays the **File Save As** dialog box, as shown in Figure 6-3.



File Name: ariana.cpp
animal.cpp
mainfrm.cpp
sdiapdoc.cpp
sdiapp.cpp
sdiapvw.cpp

Directories: c:\cproj\sdiapp
c:\
cproj
sdiapp
hlp
res

List Files of Type: Source (*.c,*.cpp,*.cxx)

Drives: c:

☐ Add to Project

Figure 6-3 File Save As dialog box

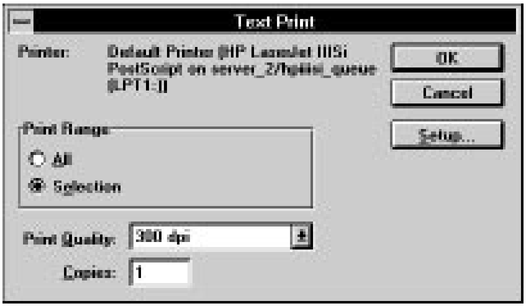
Enter a name for the file and click OK.

Writing blocks of text to files

By choosing **Write Block** from the Source window's pop-up menu, you can write the currently selected text block to a new file or append the block to an existing file. This is useful when, for example, you want to remove some functions from a file and place them in a separate file.

Printing files

To print the file in the active window, choose **Print** from the Source window's **File** menu. The **Text Print** dialog box opens, as shown in Figure 6-4.



Printer: Default Printer (HP LaserJet IIISi PostScript on server_2/hpiiisi_queue (LPT1:))

Print Range
○ All
● Selection

Print Quality: 300 dpi

Copies: 1

Figure 6-4 Text Print dialog box

The contents of the **Text Print** dialog box depend on the default printer you specify in the Windows control panel. Make any necessary changes in the dialog box, then click OK to print.

Closing files

To close a file in a Source window, choose **Close** from the Source window's **File** menu, or click the close button. If you've edited the file since the last time it was saved, the IDDE displays a dialog box asking if you would like to save the file.

Text Editing

This section describes the text editing features available in the Source window.

Typing mode

Source windows support two typing modes: overtype, which replaces characters as you type; and insert, which adds new characters to the file. You can toggle the typing mode between overtype and insert modes by using the Insert key. (Press Alt+I if you selected the Brief key binding set.) The typing mode is displayed in the status bar. A change in typing mode applies to all open Source windows, not just to the active window.

Word wrap

Because the editor is designed for source files, word wrap is not enabled by default. You must press the Enter key to start a new line. When you type past the right edge of the window, the text scrolls horizontally. You can enable word wrap and set a right margin either locally or globally using the text editor options.

Indentation

By default, the editor automatically indents a new line to the same depth as the previous line. You can set several indentation options, such as autoindent, tab width, and indent/unindent after braces.

It also is possible to change the indentation of a selected block of text by choosing **Indent Block** or **Unindent Block** from the **Format Text** submenu of the Source window's pop-up menu.

Moving around in a file

You can move the insertion point in a file by using the mouse or keyboard in the standard Windows text-editing manner. In addition, you can jump to specific points in a file using commands from the Source window's **Goto** menu.

Jumping to a matching delimiter

A common source of problems in C and C++ programming is parentheses (( )), brackets ([ ]), and braces ({}) that don't match. To find the other half of a pair of these delimiters, position the insertion point in front of one of the delimiters and choose **Matching Delimiter** from the **Goto** menu. The insertion point then moves to the front of the other half of the pair.

Delimiter checking can also be done automatically using the text editor options described later in this chapter.

Note

The IDDE editor looks for any parenthesis, bracket, or brace, including text in strings and comments.

Jumping to a specific line

To jump to a specific line, choose **Line** from the **Goto** menu. The **Goto Line** dialog box, shown in Figure 6-5, opens.

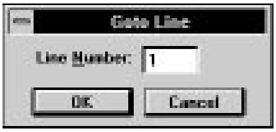
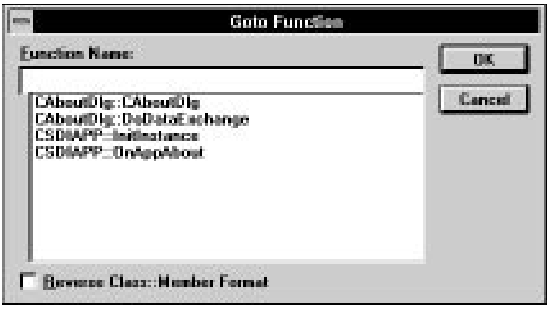


Figure 6-5 Goto Line dialog box

Type the line number in the textbox and click OK. The editor moves the insertion point to the beginning of the specified line.

Jumping to a function

To jump to a specific function, choose **Function** from the **Goto** menu. The **Goto Function** dialog box, shown in Figure 6-6, opens.



CAboutDlg::CAboutDlg
CAboutDlg::DoDataExchange
CSDIAPP::InitInstance
CSDIAPP::OnAppAbout

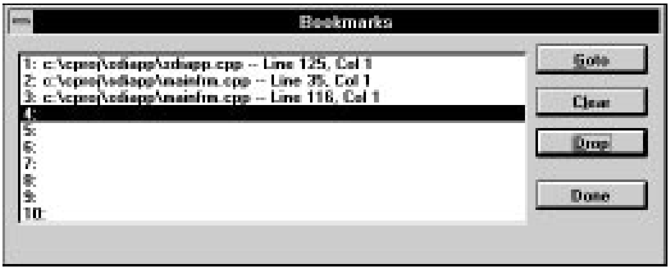
☐ Reverse Class::Member Format

Figure 6-6 Goto Function dialog box

You can select a function name from the scrolling list or type in a function name. The insertion point then moves to the beginning of the specified function.

Jumping to a bookmark

The IDDE text editor lets you position bookmarks anywhere in your files by choosing **Bookmark** from the Source window's **Goto** menu. The **Bookmarks** dialog box, shown in Figure 6-7, opens.



1: c:\cpproj\sdiapp\sdiapp.cpp -- Line 125, Col 1
2: c:\cpproj\sdiapp\mainfrm.cpp -- Line 35, Col 1
3: c:\cpproj\sdiapp\mainfrm.cpp -- Line 118, Col 1
4:
5:
6:
7:
8:
9:
10:

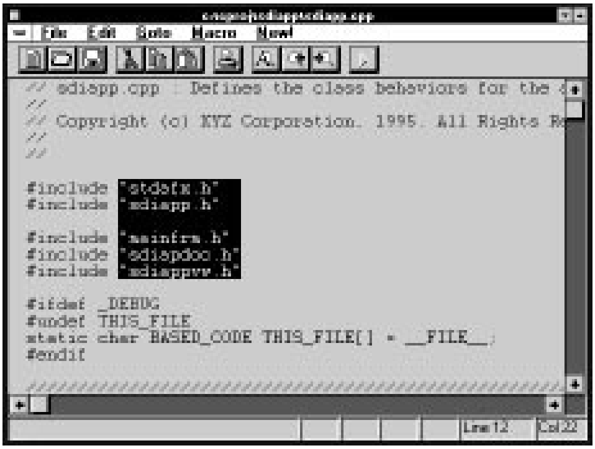
Figure 6-7 Bookmarks dialog box

To use bookmarks, first position the insertion point where you want the bookmark to be located. Then choose **Bookmark** from the **Goto** menu, select a bookmark, and click on the Drop button. When you want to return to this location, click on this entry's name in the list of bookmarks, then click on Goto.

Selecting text

Text selection is accomplished using one of two standard techniques: clicking and dragging the mouse or shift-clicking. The IDDE text editor provides additional modes for text selection using the **Column** and **Line** commands in the pop-up menu's **Select** submenu.

Both commands require that you first select a block of text in the standard manner. **Column** changes the currently selected text block into a column-oriented select block; only the characters in the columns between the start and end of the original text block are selected. This is shown in Figure 6-8.

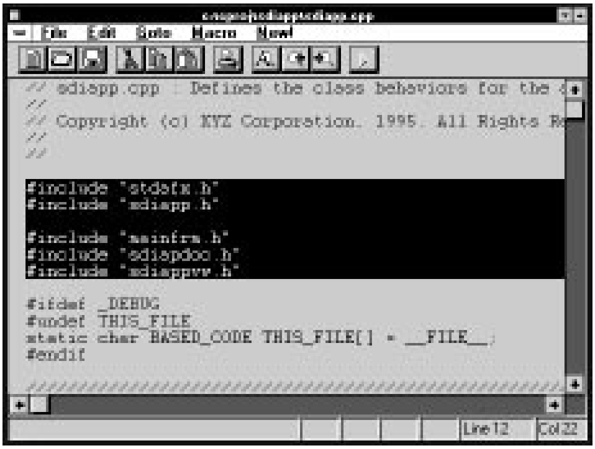


```
// sdiapp.cpp : Defines the class behaviors for the
//
// Copyright (c) XYZ Corporation, 1995. All Rights R
//
//

#include "stdafx.h"
#include "sdiapp.h"

#include "mainfrm.h"
#include "sdiapdoc.h"
#include "sdiappvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
```

Figure 6-8 A column select block

Line changes the currently selected text block into a line-oriented select block; all the lines from the start to the end of the original text block, including the first and last lines, are selected.

This is shown in Figure 6-9.



```
// sdiapp.cpp : Defines the class behaviors for the
//
// Copyright (c) XYZ Corporation. 1995. All Rights R
//
//

#include "stdafx.h"
#include "sdiapp.h"

#include "mainfrm.h"
#include "sdiapdoc.h"
#include "sdiappvw.h"

#ifdef _DEBUG
#undef THIS_FILE
static char BASED_CODE THIS_FILE[] = __FILE__;
#endif
```

Figure 6-9 A line select block

Choosing **Normal** from the **Select** submenu of the pop-up menu changes a column- or line-oriented text block back to the original selection. **Cancel** deselects the current selection block; this can also be accomplished by clicking somewhere outside the selected text.

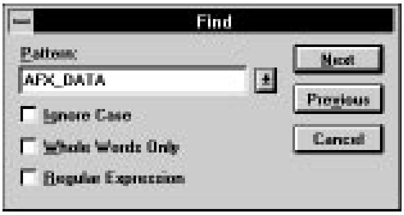
The text editor supplies standard Windows functions for cutting, copying, pasting, and deleting text. These functions can be accessed through either the **Edit** menu or the pop-up menu. In addition, by clicking and dragging a block of selected text, you can reposition the text anywhere in the buffer. Press the Control key while releasing the block to copy rather than move the block. When you are dragging text blocks around in this way, a small outlined box is drawn next to the cursor to indicate this mode.

Searching and replacing

The IDDE editor has powerful text-based search and replace functions that let you search open files for a string and replace one string with another. In addition, you can search through any set of source files, whether or not they are open or in your project.

Finding a string

To search for a string, choose **Find** from the Source window’s **Edit** menu. The dialog box shown in Figure 6-10 opens.



Pattern: AFX_DATA

☐ Ignore Case
☐ Whole Words Only
☐ Regular Expression

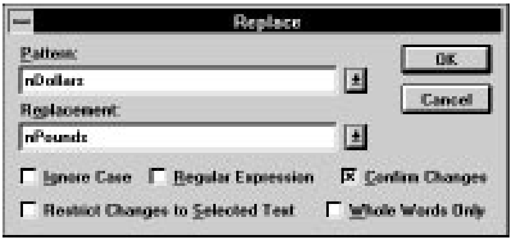
Figure 6-10 Find dialog box

If you select some text before opening the **Find** dialog box, the selected text becomes the default search pattern. You can either use this text as it appears, select a previous string from the drop-down list, or type in a new string. Then click on the Next (or Previous) button. The editor locates and selects the next (or previous) occurrence of the string in the active file. You can repeat the search in the same direction by choosing **Repeat Find** from the **Edit** menu. If the pattern is not found, the editor returns to the current insertion point and displays `Pattern not found` in the status line.

By checking Regular Expression in the **Find** dialog box, you can use the wildcard characters ? and * in your search string. The ? character matches any single character, while the * character matches any string of consecutive characters. Regular expressions permit more powerful text searches.

Replacing a string

To replace an occurrence of a string in your file with another string, choose **Replace** from the Source window’s **Edit** menu. The **Replace** dialog box opens, as shown in Figure 6-11.



Pattern: nDollars

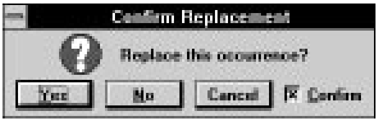
Replacement: nPounds

☐ Ignore Case ☐ Regular Expression ☒ Confirm Changes
☐ Restrict Changes to Selected Text ☐ Whole Words Only

Figure 6-11 Replace dialog box

Type the search string or regular expression in the Pattern textbox, just as you did in the **Find** dialog box described earlier. Then type the replacement string in the Replacement textbox. Note that wildcards cannot be used in the Replacement textbox. You can also select previous search and replacement strings from the drop-down list. Begin the replacement by clicking OK.

If you check the Confirm Changes option, the editor scrolls the display to each occurrence of the search string, selects it, and asks whether you want to replace it by displaying the **Confirm Replacement** dialog box in Figure 6-12.



Replace this occurrence?

Figure 6-12 Confirm Replacement dialog box

Click on Yes to replace the string and continue searching, or click on No to continue searching without replacing. Click on Cancel to stop searching. If you do not check Confirm Changes, the editor replaces all occurrences of the search string without confirmation messages.

You can use the **Undo** command to undo the entire set of replacements of the search string.

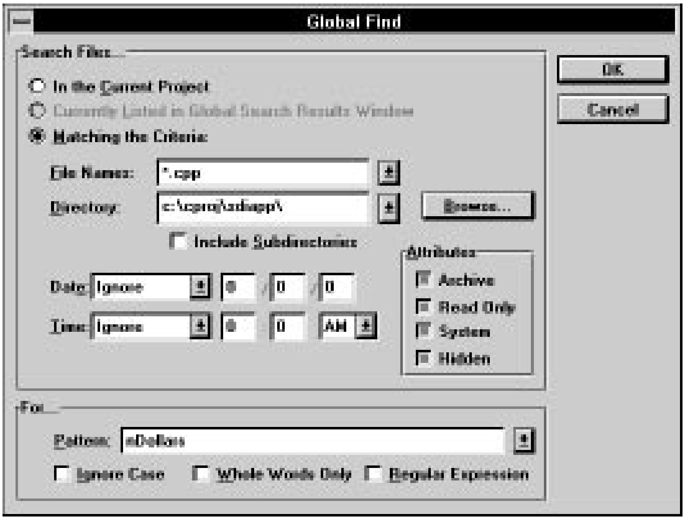
Searching through multiple files

The text editor's global find feature provides a powerful means of locating a string in any set of files. For example, this feature is useful for:

- Locating undefined symbols
- Changing function, variable, or class names
- Fixing references to a function with changed parameters

To search for a string in multiple files, choose **Global Find** from the Source window's **Edit** menu.

The **Global Find** dialog box opens, as shown in Figure 6-13.



○ In the Current Project
○ Currently Listed in Global Search Results Window
● Matching the Criteria:

File Names: *.cpp
Directory: c:\cproj\sdiapp\
☐ Include Subdirectories
Date: Ignore 0 / 0 / 0
Time: Ignore 0 0 AM
☑ Archive
☑ Read Only
☑ System
☑ Hidden

Pattern: nDollars
☐ Ignore Case ☐ Whole Words Only ☐ Regular Expression

Figure 6-13 Global Find dialog box

To perform a global search, first specify the set of source files you want to search. You can choose to search:

- All source files in the current project
- All files listed in the Search window (which opens after the first search)
- All files matching the criteria you specify, including filename, directory, date, time, and file attributes

Next, specify the string or regular expression to search for. (As with the **Find** command, you can preselect the search text before choosing **Global Find**.) When you click OK, the editor opens the Search window. The editor searches through the indicated files and lists the names of files containing a match in the Search window.

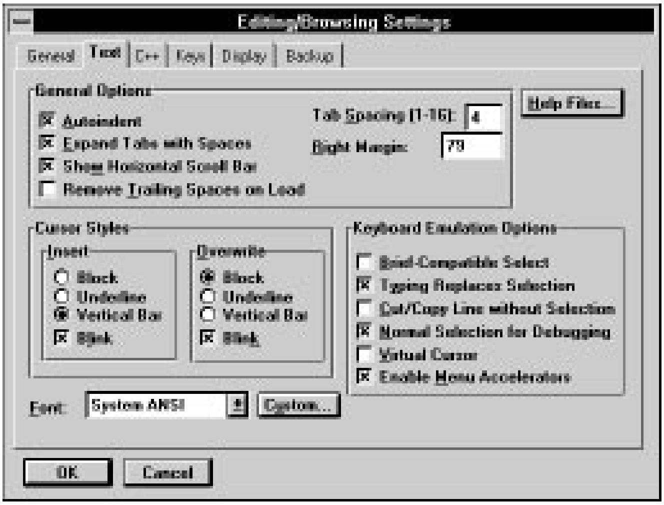
By double-clicking on a filename, you can open the file; the first occurrence of the pattern is highlighted. You can continue searching for your string or expression in that file by choosing **Repeat Find** from the Source window's **Edit** menu. For more information, see Chapter 21, "Text Editor Reference."

Undoing edits

You can undo typing, cutting, pasting, and string-replace operations by choosing **Undo** from the **Edit** menu. Undo multiple edits by repeatedly selecting the **Undo** command. You can set the number of editing operations that can be undone in the text editor options.

Text Editor Options

Global text editor options are set in the **Editing/Browsing Settings** dialog box, shown in Figure 6-14. To open this dialog box, choose **Text Settings** from the **Edit** menu. (You can also choose **Editing/ Browsing Settings** from the **Environment** menu; in this case you will have access to Class and Hierarchy Editor options as well.)



General Options
☒ Autoindent
☒ Expand Tabs with Spaces
☒ Show Horizontal Scroll Bar
☐ Remove Trailing Spaces on Load
Tab Spacing [1-16]: 4
Right Margin: 79

Cursor Styles
Insert
○ Block
○ Underline
● Vertical Bar
☒ Blink
Overwrite
● Block
○ Underline
○ Vertical Bar
☒ Blink

Keyboard Emulation Options
☐ Brief-Compatible Select
☒ Typing Replaces Selection
☐ Cut/Copy Line without Selection
☒ Normal Selection for Debugging
☐ Virtual Cursor
☒ Enable Menu Accelerators

Font: System ANSI
Custom...

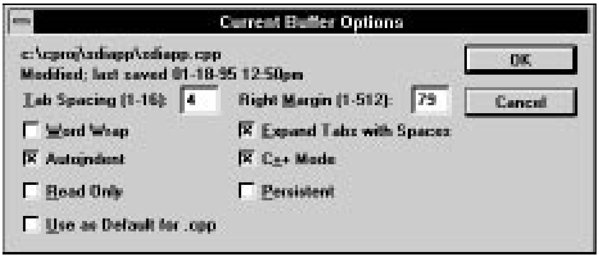
Figure 6-14 Editing/Browsing Settings dialog box

The **Editing/Browsing Settings** dialog box is a tabbed dialog that contains numerous options for different aspects of the text editor:

- The Text page contains options for indentation, cursor styles, keyboard emulation, and text editor font.
- The C++ page lets you set options to enable C++ mode globally, check delimiters, indent after braces, auto-align comments, and add keywords to the keyword dictionary.
- The Keys page lets you customize key bindings and assign key combinations to macros.

- The Display page lets you select special colors and font styles for keywords, comments, and preprocessor symbols.
- The Backup page lets you set options for backing up files.
- The General page lets you set the maximum undo level and select a key binding file.

Several text editor options can be set on a buffer-by-buffer basis. To set these local options, choose **Current Buffer Settings** from the **Edit** menu. The **Current Buffer Options** dialog box, shown in Figure 6-15, opens.



c:\cproj\sdiapp\sdiapp.cpp
Modified; last saved 01-18-95 12:50pm
Tab Spacing (1-16): 4
Right Margin (1-512): 79
☐ Word Wrap
☒ Expand Tabs with Spaces
☒ Autoindent
☒ C++ Mode
☐ Read Only
☐ Persistent
☐ Use as Default for .cpp

Figure 6-15 Current Buffer Options dialog box

This dialog lets you override global indentation options, set word wrap, and set the buffer to read-only.

You can find detailed information about text editor options in Chapter 21, “Text Editor Reference.”

Macros

The text editor has powerful macro facilities. You can record a sequence of keystrokes, save the sequence as a named macro, assign a keyboard shortcut to the macro, place the macro in the **Macro** menu, edit the macro, and so on. To record a macro, choose **Record Macro** from the **Macro** menu; to end the macro, choose **Stop Recording**. You can play back the stored sequence by choosing **Play Macro**.

Compiling Files and Checking Errors

The IDDE makes it easy to iteratively edit and check your source code. To save and compile the current buffer, choose **Compile** from the Source window's **File** menu. The IDDE compiles the file using the current project build settings. If the compilation is successful, the file is marked as compiled. If the compilation generated errors, you can quickly locate the line in your source code that generated the error by double-clicking on the error in the Output window.

For more information about compiling files and building the project, see Chapter 8, "Testing an Application."

Adding Look and Feel with Resources

7

Any application written for the Windows environment needs an interface to the user that gives it the “look and feel” of a Windows application. Various standard mechanisms are available to create this look and feel, and resources are some of the most important of these. Resources such as menus and dialog boxes define the interface to an application.

ResourceStudio contains tools for creating and modifying the resources you will need. This chapter defines the different types of resources and explains how to use ResourceStudio to create three basic resources—menus, dialog boxes, and bitmaps. The final section describes how to work with the identifiers by which resources are referenced in the source code.

For more information about ResourceStudio, see the “ResourceStudio Reference.” For more information on resources and programming with resources, refer to the following texts:

- *Programming Windows 3.1* by Charles Petzold
- *Microsoft Windows Programmer’s Reference, Volume 4: Resources*

What Are Resources?

Resources are structured data objects that define Windows user interface elements. When you choose a command from a menu or select options from a dialog box, for example, you are interacting with an application’s resources. Predefined resource types include bitmaps, cursors, icons, fonts, dialog boxes, menus, accelerators, strings, and version information.

An application’s resources typically are defined in a resource script file (`.rc`). Each resource definition contains information and data relevant to that resource type, as well as the identifier by which the resource is referenced in the source code. Resource script files are created and edited with ResourceStudio.

Resources are compiled by a special resource compiler and are linked to the Windows application separately from the application code. This modular design enables you to modify the appearance of an application without changing, or even accessing, the program source code. Separating resources from code also makes it easier to use the same resource definitions for multiple applications.

Resource Types

You can create and edit the following resource types in ResourceStudio:

Bitmaps
Bitmaps are used by an application to display pictures on the screen. Each bit, or group of bits, in a bitmap represents one pixel of the image. Bitmaps may be any size you specify.

Cursors
Cursors are small bitmaps, usually 32 x 32 pixels in size, that represent the mouse pointer. Cursor images may contain areas that are transparent, allowing the screen background to show through unused parts of the cursor image. Cursors also may contain inverted areas in which the screen background color is reversed.

In addition to image information, a cursor resource also specifies the cursor’s hot spot, the exact pixel in the image that maps to the mouse pointer’s screen location.

Icons
Icons are small bitmaps, usually 32 x 32 or 16 x 32 pixels in size, that represent applications or actions within an application. For example, when you minimize a Windows application, an icon is displayed on the screen to represent the minimized program. Like cursors, icons may contain transparent and inverted areas.

Fonts

A font resource is a collection of up to 256 bitmaps of identical height. A font typically represents a character set, but it also may be used for more efficient management of bitmaps not representing characters.

Dialog boxes

Dialog boxes are windows that communicate information and receive user input. For example, a dialog box may let the user set program options, or alert the user to an error and ask how to proceed. To enter information and make selections in a dialog box, a user manipulates graphical elements called controls. Commonly used controls include buttons, check boxes, listboxes, textboxes, and scroll bars.

A dialog box resource is a complex set of data describing each control in detail, as well as general properties such as the dialog box size and location.

Menus

Menus are hierarchical lists of program commands and options. When a user chooses an item from a menu, either an action is taken or another menu opens to provide additional commands. A menu resource identifies the available commands, specifies their individual styles and attributes, and determines the order in which they appear.

Accelerators

Accelerators are key combinations a user presses to perform a task in an application and are frequently used as shortcuts to menu choices. An accelerator resource contains a collection of key combinations and associated commands.

Strings

Strings are text that an application may use in window titles, menus, dialog boxes, error messages, and so on. A string resource is a table of strings.

Version information

The Version Information resource contains version information for Windows `.exe` and `.dll` files, such as version number, file description, company name, and copyrights.

User-defined resources
User-defined resources contain data that you define and use in your program in any manner you choose. For example, you may want to attach a table of data or a block of text to your executable file. ResourceStudio allows you to include these types of data in your application's resources and to edit the data in hexadecimal format.

Using ResourceStudio

ResourceStudio can be used to:

- Create new resources in script or binary form
- Edit resources, even those already linked to an executable or DLL
- Copy resources between files

This section explains how to start ResourceStudio and use it to create resource files. Detailed steps are provided for creating three basic resources: menus, dialog boxes, and bitmaps. This section also outlines several useful features of ResourceStudio, such as drag and drop.

Starting ResourceStudio

You can start ResourceStudio using any one of several commands available in the IDDE's **Resource** menu:

- **Edit** opens the current project's resource (`.rc`) file for editing.
- **New** opens the current project's resource file. A dialog box then opens, allowing you to create a new resource.
- **Open** opens a dialog box from which you can select for editing any file that contains resources.
- **Settings** opens ResourceStudio to the **Settings** dialog box, but does not load any resource file.

You can also double-click on a resource file in the Project window to open that file for editing with ResourceStudio.

To create a new resource file, choose **Open** from the **Resource** menu, and in the **File Open** dialog box click Cancel. Then, double-click on the ResourceStudio icon on the desktop. The Resource Studio Shell window opens (Figure 7-1).

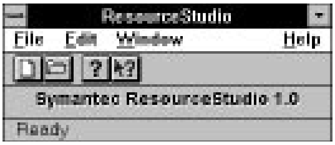


Figure 7-1 Resource Studio Shell window

The Shell window is ResourceStudio's main window and contains the main menu and toolbar. From this window, you can create and open resource files and set ResourceStudio preferences.

Another window that will open as you work is the Property Sheet. The Property Sheet does not open immediately, but rather opens when it is first needed. While creating and editing resources, you can use the Property Sheet to view and edit properties of the currently selected object. (In this context, "object" refers to either a resource as a whole or to an element within a resource, such as a dialog box control or a menu item.) Some types of objects have multiple pages of properties; to switch between pages, simply click on the tabs.

Creating a new resource file

Resources are contained in resource files. You can use ResourceStudio to create and edit most types of files that store Windows resources. For example, you can edit the resources of executable files even if you do not have the original source code.

To create a new resource file, choose **New** from the **File** menu. The **New File** dialog box opens (Figure 7-2), listing the types of resource files that can be created with ResourceStudio.

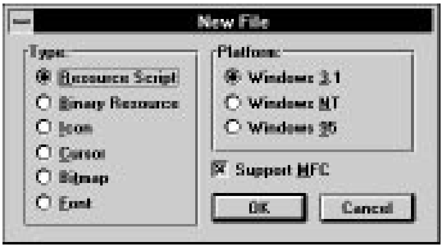


Figure 7-2 New File dialog box

Select the type of resource file you want to create (see Table 7-1). Note that resource script (`.rc`) and binary resource (`.res`) files usually contain more than one resource. If you are starting to create resources for a new application, `.rc` is likely to be your choice of file type. Resource scripts can contain any type of resource.

Table 7-1 Resource file types

File Type	Contains
Resource script (`.rc`)	Any type or combination of resources in text format
Binary resource (`.res`)	Any type or combination of resources in binary format
Icon (`.ico`)	A single icon resource in binary format
Cursor (`.cur`)	A single cursor resource in binary format
Bitmap (`.bmp`)	A single bitmap resource in binary format
Font (`.fnt`)	A single font resource in binary format

Resources such as icons can be contained either in individual icon resource files (`.ico`) or with other resources in a resource script file. Some resources, such as dialog boxes and menus, are contained only in `.rc` or `.res` files. Thus, these resource options do not appear separately in the **New File** dialog box.

The Platform option in the **New File** dialog box lets you create resource files that are compatible with Windows 3.1, Windows NT, or Windows 95. You can also elect to create resources for an MFC application by checking Support MFC.

After selecting a type of resource file, click OK. The window that opens next depends on the type of file you specified. If you selected one of the file types containing a single resource (icon, cursor, bitmap, or font), that resource type's editor opens in a separate window.

If you selected Resource Script or Binary Resource, the Browser window opens (Figure 7-3).

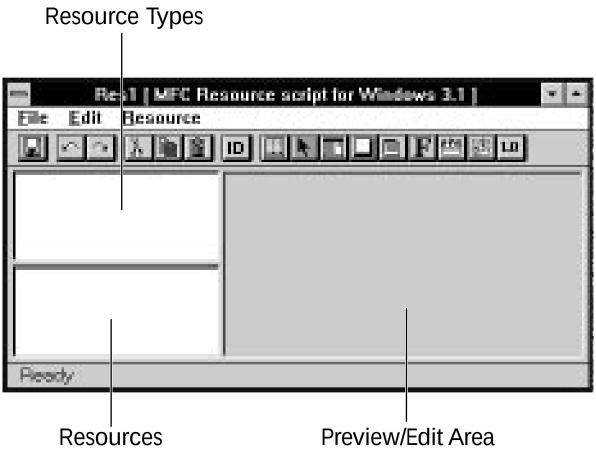


Figure 7-3 Browser window

The Browser window has three main areas:

- Resource Types: A listbox containing all the resource types found in the current file.
- Resources: A listbox containing resources of the currently selected type.
- Preview/Edit: A display of the selected resource. Resources also are opened for editing in this area when created, or by choosing **Edit Resource** from the **File** menu.

When a new resource file is first opened, the Resource Types and Resources listboxes are both empty.

Note

ResourceStudio has several subprograms, each of which is used to edit a particular resource type. In this chapter, for example, you will use the Menu editor, Dialog box editor, and Bitmap editor. Each individual resource editor can either be opened in a separate window or can use the right pane of the Browser window from which it is launched. The latter is the default behavior in most circumstances.

Editing a resource file

Besides the resource files listed in the previous section, you can edit the resources of executable files, even though you don't have the original source code. By choosing **Open** from the Shell window's **File** menu, you can open files of any type listed in Table 7-1, plus files of the following types:

- Windows executable (`.exe`): Bound resources of any type
- Dynamic-link library (`.dll`): Bound resources of any type

Warning

Do not modify or save the resources of a running application. Saving a resource usually changes the locations of the resources within the file.

Creating a new menu resource

The main interface between the user and a Windows application is generally the application's menu. The IDDE's **Resource** menu is a good example, as shown in Figure 7-4.

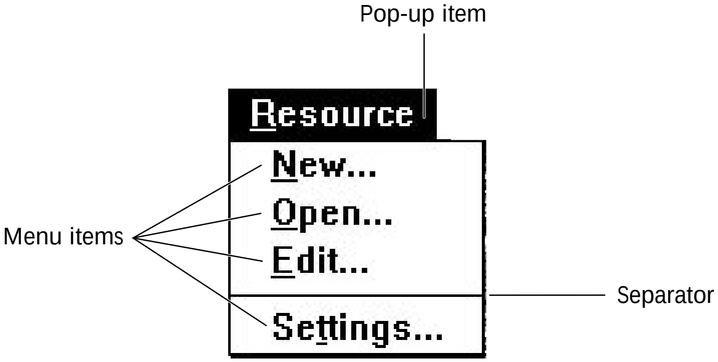


Figure 7-4 A typical menu

A menu is a hierarchical structure containing pop-up items, menu items, and separators.

- Pop-up items open a drop-down menu box containing menu items and additional pop-ups. The top-level items that are displayed on the window's menu bar are usually pop-up items. A pop-up item can be thought of as a container for other items (including other pop-ups).
- Menu items usually constitute the majority of items in drop-down menus. When a menu item is chosen, Windows sends a message containing the item's command identifier to the application.
- Separators are lines that divide the menu into logical groups.

To create a new menu resource, choose **New Menu** from the **Resource** menu of the Browser window. The Menu editor opens in the right pane of the Browser window (Figure 7-5).

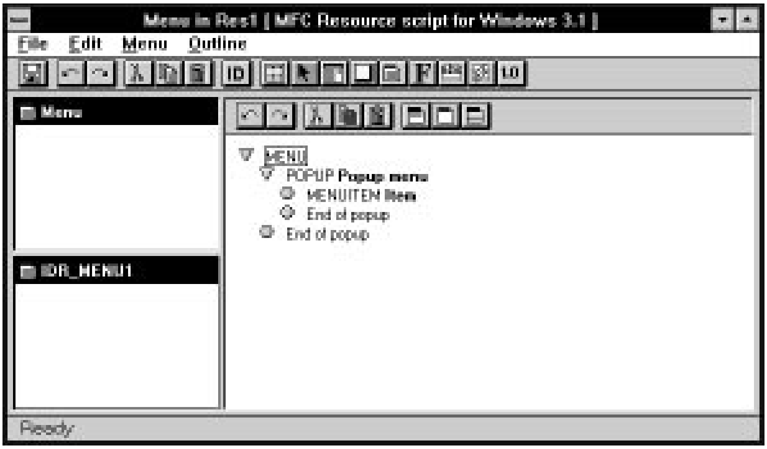


Figure 7-5 Menu editor open in Browser window

Note

While a resource editor is open in the Browser window, the Browser window's menu is replaced by that of the particular resource editor. The editor's toolbar is displayed at the top of the right pane, below the Browser window's toolbar.

The new menu resource initially contains a single pop-up item and a menu item. The list of items is indented to show the hierarchy. Pop-up items are labeled POPUP, menu items are labeled MENUITEM, and separators are labeled SEPARATOR. The currently selected item is enclosed in a box; to select any item, click on it. The Property Sheet shows properties of the selected item.

The Test menu window (Figure 7-6) opens at the same time. You can test the menu in progress at any time in this window.

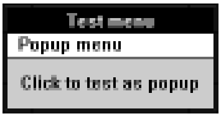
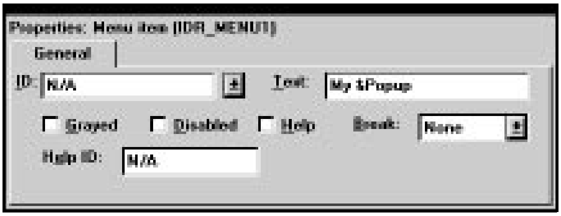


Figure 7-6 Test menu window

Adding a pop-up item

You can start creating a menu by adding a new pop-up item to the menu bar. To do so, click on Menu at the head of the item list. Then choose **New Popup** from the **Menu** menu. The new pop-up item is inserted at the start of the list.

The Property Sheet displays the properties of the new pop-up item (Figure 7-7). At this point, you probably want to change only the menu name, which is contained in the Text field.


Properties: Menu item (IDR_MENU1)

ID: N/A

Text: My &Popup

☐ Grayed ☐ Disabled ☐ Help

Break: None

Help ID: N/A

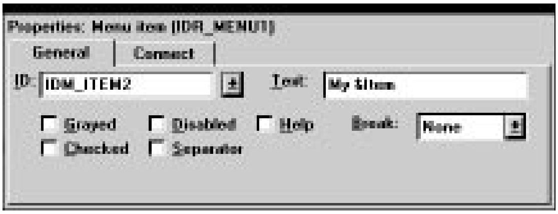
Figure 7-7 Properties of a Pop-up menu

When changing the menu name, you can also assign an activation key to the menu. Put an ampersand (&) in front of the corresponding letter in the text.

Adding a menu item

To add one or more menu items to a drop-down menu, first make sure that the pop-up item is selected. Then choose **New Item** from the **Menu** menu. The new menu item is inserted into the hierarchy.

The Property Sheet displays the properties of the new menu item (Figure 7-8). You can change the item's text and add an activation key. As with pop-up items, you add an activation key by typing an ampersand (&) in front of the corresponding letter in the text.


Properties: Menu item (IDR_MENU1)

ID: IDM_ITEM2

Text: My &Item

☐ Grayed ☐ Disabled ☐ Help

Break: None

☐ Checked ☐ Separator

Figure 7-8 Properties of a menu Item

The ID field contains the command ID, which is sent to the application when the user chooses this menu item. New command and resource IDs are assigned automatically to new objects when they are created; you can also change ID names and numerical values as desired. For more information on resource IDs, see the section "Managing Resource IDs" later in this chapter.

Adding separators

Separators enhance the readability of menus. To insert a separator after the currently selected item, choose **New Separator** from the **Menu** menu.

Editing menus

The normal editing operations (cut, copy, paste, and delete) work with items in a menu. Note that when you perform one of these operations on a pop-up item, all of the items it contains are affected as well.

You can rearrange menu items quickly by clicking and dragging. To move an item, drag the item to the location where it should be inserted. If you hold down the Control key while dragging, a copy of the selected item is inserted. The copy is given a new, unique resource ID.

You can also rearrange menus with the arrow keys. To move an item up or down within its present level in the hierarchy, press Ctrl+Up Arrow or Ctrl+Down Arrow. To move an item horizontally, use Ctrl+Right Arrow or the Ctrl+Left Arrow.

Closing the menu editor

After you have arranged your new menu, close the Menu editor by choosing **Close Editing** from the **File** menu. The Browser window is updated to show the new resource (Figure 7-9).

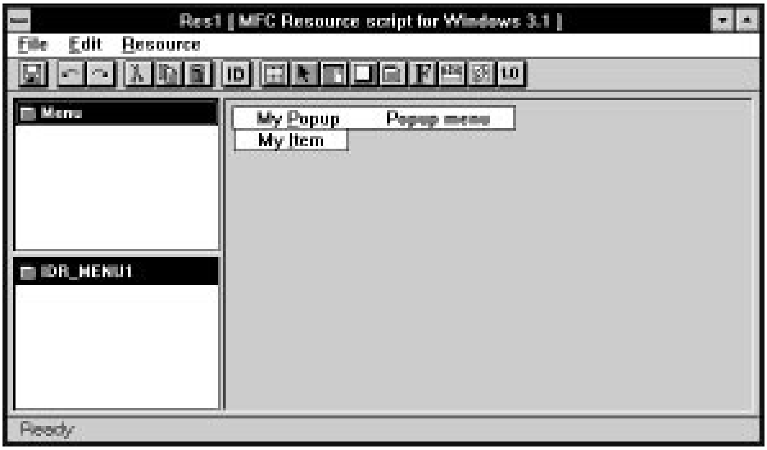
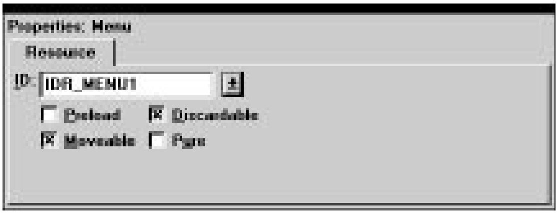


Figure 7-9 Browser window after creating menu resource

The Resource Types list contains an entry called Menu to show that you have at least one menu resource. The Resources list contains the ID of your menu resource. The Preview/Edit area shows a preview of the new menu.

The Property Sheet displays the menu resource's ID and memory options (Figure 7-10).



ID: IDR_MENU1

☐ Preload ☒ Discardable

☒ Moveable ☐ Pure

Figure 7-10 Properties of a menu resource

To edit a menu resource, double-click on its ID in the Resource list, or select the resource and choose **Edit Resource** or **Edit in Separate Window** from the **File** menu.

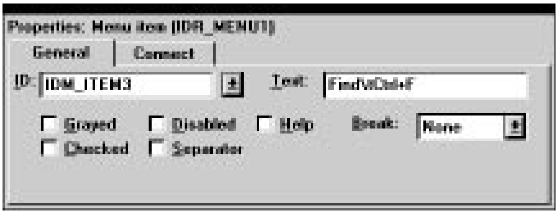
Creating menus with accelerators and item help

As you create menus, you usually want to assign accelerator key combinations and item help strings at the same time. To do so, you must set up the string and accelerator tables before adding menu items, as outlined here:

1. Create a string table resource by choosing **New String Table** from the Browser window's **Resource** menu.
2. Close the String Table editor by choosing **Close Editing** from the **File** menu.
3. Create a new accelerator table resource by choosing **New Accelerator Table** from the Browser window's **Resource** menu.
4. Close the Accelerator Table editor by choosing **Close Editing** from the **File** menu. Note the accelerator table's resource ID.
5. Create a new menu resource by choosing **New Menu** from the Browser window's **Resource** menu.
6. Make the new menu's resource ID the same as that of the accelerator table resource. In the Property Sheet, select the accelerator table resource's ID from the drop-down list in the ID field. You now can add menu items with accelerators and item help.

To set a menu item's accelerator, click on the Connect tab in the Property Sheet, then click on the Next Key. You are prompted to press the key combination that is used as the accelerator. When you do so, the accelerator is stored in the associated accelerator table. To set a menu item's help string, type the string into the Prompt field in the Property Sheet. The string is automatically placed in the string table.

To show which key combination is associated with each menu item, add the accelerator to the item text. For readability, the accelerator is usually separated from the command name with a tab. For example, if you want Ctrl+F to be the accelerator for a command called Find, enter `Find\tCtrl+F` into the Text field of the menu item's Property Sheet (Figure 7-11).



Properties: Menu item (IDR_MENU1)

General | Connect

ID: IDM_ITEM3

Text: Find\tCtrl+F

☐ Grayed ☐ Disabled ☐ Help

☐ Checked ☐ Separator

Break: None

Figure 7-11 Menu item text, showing accelerator

Simple guidelines for creating menus

When creating a menu, keep in mind the following points:

- Place frequently chosen options near the top of the menu to make them more accessible to the user.
- Group related options under specific menu titles.
- Follow standard conventions to make the application easier to use.

Windows users are accustomed to certain menu arrangements. For example, the **File** menu is usually the first menu in the menu bar and contains commands such as **New**, **Open**, and **Save**. You can choose commands from the **Menu** menu to quickly create standard **File**, **Edit**, and **Help** menus.

Creating a new dialog resource

A dialog box is a window that communicates information and receives user input. It contains graphical elements called “controls.” Figure 7-12 shows a typical dialog box.

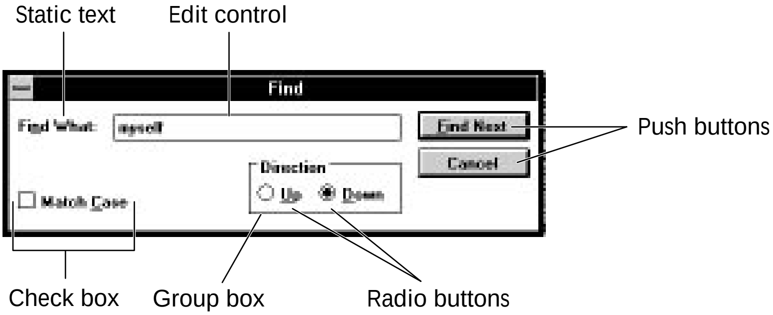


Figure 7-12 A typical dialog box

The following types of controls can be included in your dialog boxes:

- Push buttons send command codes to your application when the user clicks on them. They trigger an immediate action.
- Check boxes are rectangular buttons that turn options on or off. Text describing the option is displayed to the left or right.
- Radio buttons provide a set of options, only one of which may be selected. They are generally arranged in logical groups of two or more.
- Edit controls (or textboxes) are rectangular boxes used for text entry.
- Listboxes contain a list of items, usually in text form. The user can browse through the list and select one or more items.
- Comboboxes combine the features of an edit control and a listbox; they let the user select an item from a list or type a selection into the textbox.

- Scroll bars let the user specify a numerical option with an analog control. (Edit controls and listboxes have their own scroll bars and do not need a stand-alone scroll bar.)

- Group boxes are rectangular frames with an optional caption, used to group other controls together visually.

- Static text is used to convey information or to label controls that do not have captions (such as edit controls).

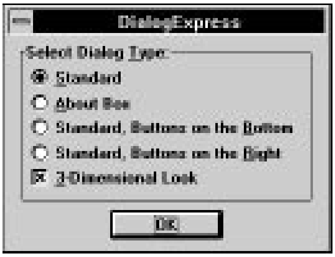
- Pictures are used for decorative purposes. Picture types include empty frames, solid boxes, and icons.

- Custom controls are user-defined controls implemented in a DLL.

- User controls are user-defined controls that are not implemented in DLLs, or whose implementation is non-standard. (ResourceStudio treats VBX controls as user controls.)

- The following Windows 95 controls are supported: Animate controls, Tab controls, Tree View controls, List View controls, Hotkey controls, Track Bars, Progress controls, and Up/Down controls.

To create a new dialog box resource, choose **New Dialog Box** from the **Resource** menu of the Browser window. The **DialogExpress** dialog box opens (Figure 7-13).



● Standard
○ About Box
○ Standard, Buttons on the Bottom
○ Standard, Buttons on the Right
☒ 3-Dimensional Look

Figure 7-13 DialogExpress dialog box

DialogExpress gives you several options for simple dialog boxes that you may use as a starting point for your own dialog box. Select Standard and click OK. The Dialog editor opens in the right pane of the Browser window (Figure 7-14).

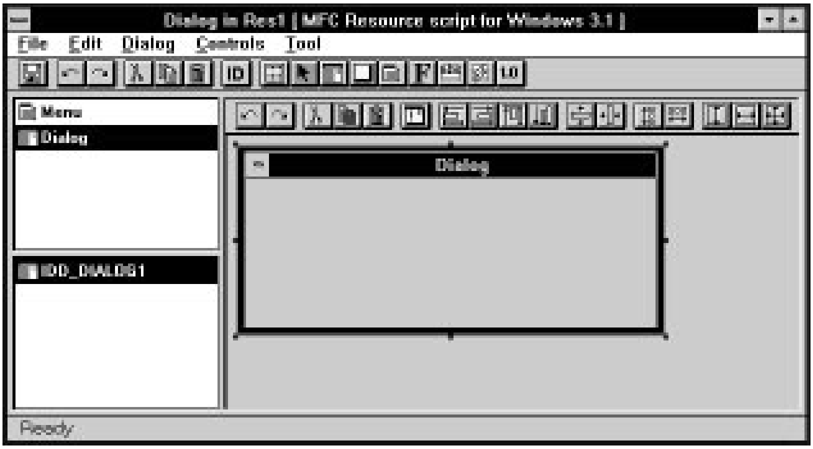


Figure 7-14 Dialog editor open in Browser window

At the same time, the Dialog editor toolbox (Figure 7-15) opens. This toolbox provides shortcuts to commands in the **Tool** menu that let you create the different types of dialog box control.

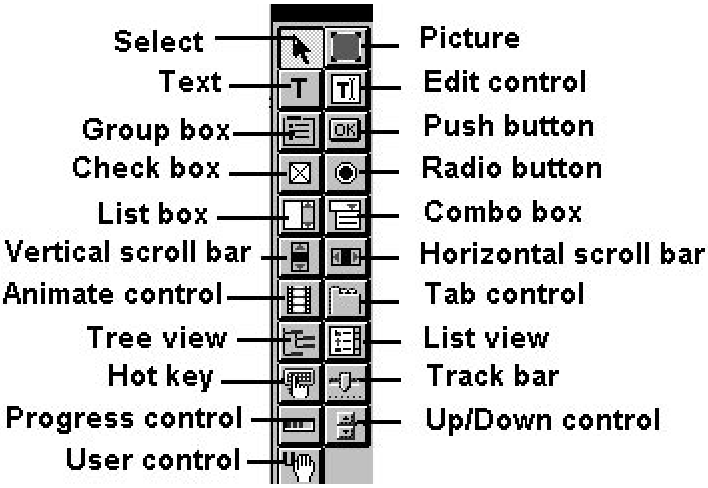
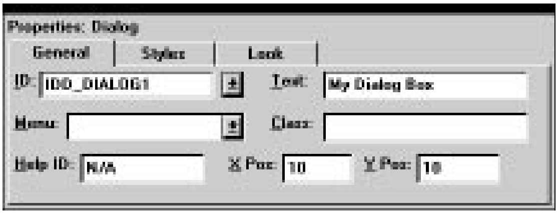


Figure 7-15 Dialog editor toolbox

The Property Sheet shows the General, Styles, and Look properties of the dialog box (Figure 7-16). You may want to start by changing the dialog box title.

Type the new title into the Text field of the General properties page.



ID: IDD_DIALOG1
Text: My Dialog Box
Menu:
Class:
Help ID: N/A
X Pos: 10
Y Pos: 10

Figure 7-16 Properties of a dialog resource

Adding a push button

Initially, a new Standard dialog has no controls. To create a new push button:

1. Choose **Push Button** from the **Tool** menu or click on the corresponding icon in the toolbox.
2. Click in the dialog box on the point at which you want to position the upper-left corner of the new button. Hold down the left mouse button and drag the cursor. A rubber-band rectangle appears.

3. Release the left mouse button when the rectangle is the proper size for the new button. A new push button is placed in your dialog box (Figure 7-17).

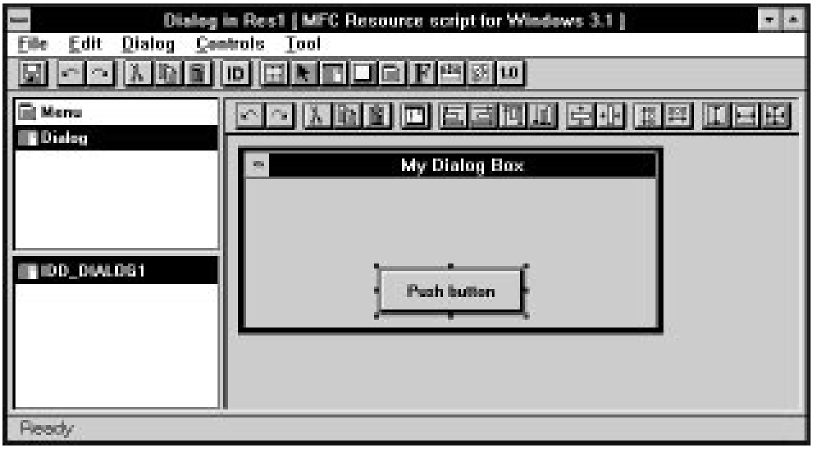

Dialog in Res1 | MFC Resource script for Windows 3.1 |
File Edit Dialog Controls Tool
Menu
Dialog
IDD_DIALOG1
My Dialog Box
Push button
Ready

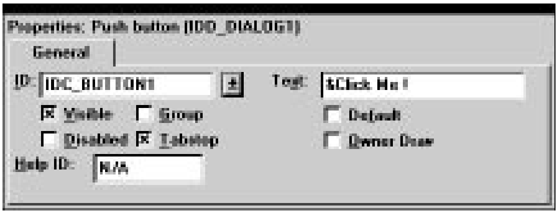
Figure 7-17 Dialog box resource with new push button

The new push button is selected and highlighted. You can then move and resize your new push button control with the mouse.

- To move a control, click and drag it to a new location.
- To resize a control, click and drag one of the eight “handles” that appear along its edges.

You also may center the selected control in the dialog box. Choose **Vertical** or **Horizontal** from the **Center** submenu of the **Controls** menu.

The Property Sheet lets you modify push button properties (Figure 7-18). To change the button text, for example, change the Text field to the desired text. As with items in menus, you can set the activation key of a button or other control by typing an ampersand (&) before the corresponding letter in the text.



ID: IDC_BUTTON1

Text: &Click Me !

☒ Visible ☐ Group ☐ Default

☐ Disabled ☒ Tabstop ☐ Owner Draw

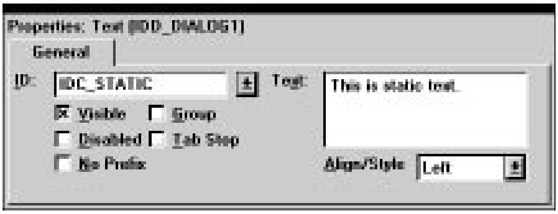
Help ID: N/A

Figure 7-18 Properties of a push button control

Adding static text

To add static text to a dialog box (for example, an informational message), choose **Text** from the **Tool** menu or click on the corresponding icon in the toolbox. Then click in your dialog box and drag the rubber-band box to the desired size.

The Property Sheet shows the properties of the static text (Figure 7-19). To change the text, edit the Text field. After entering the text, you can resize the static control to the size of the text by dragging the handles.



ID: IDC_STATIC

Text: This is static text.

☒ Visible ☐ Group

☐ Disabled ☐ Tab Stop

☐ No Prefix

Align/Style: Left

Figure 7-19 Properties of a static text control

To center the text within the control, select Center from the drop-down list in the Align/Style property. You can center the control within the dialog box by choosing **Horizontal** from the **Center** submenu of the **Controls** menu.

A static text control may contain multiple lines of text. While typing text into the Caption field, press Ctrl+Enter to start a new line. Likewise, insert tab characters by pressing Ctrl+Tab.

Testing the dialog box

You can test your dialog box resource without leaving ResourceStudio. To open the new dialog box for testing, choose **Test Dialog** from the **Dialog** menu (Figure 7-20).

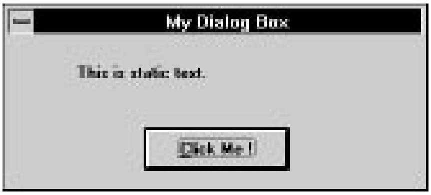


Figure 7-20 Testing the new dialog box

When you are testing the dialog box, ResourceStudio does not function; you must close the test dialog box by pressing Alt+F4 before continuing.

Aligning controls

Commands are available to align controls, to space them evenly, and to make them the same size. The following steps illustrate the three operations:

1. Add a few odd-sized buttons to your dialog box, using the procedure described in the section “Adding a push button” earlier in this chapter. Place them in a horizontal row.

2. Select all the buttons at once by clicking on one, then holding down either the Shift or Control key as you click on the others. You can also choose **Select** from the **Tool** menu and drag a rubber-band box around the controls.

3. Establish one button as the “standard.” Hold down Control and click on this button. The standard button is highlighted in blue on your screen. Figure 7-21 shows what your dialog box might look like. In the example,

the button labeled “#2” has been selected as the standard.

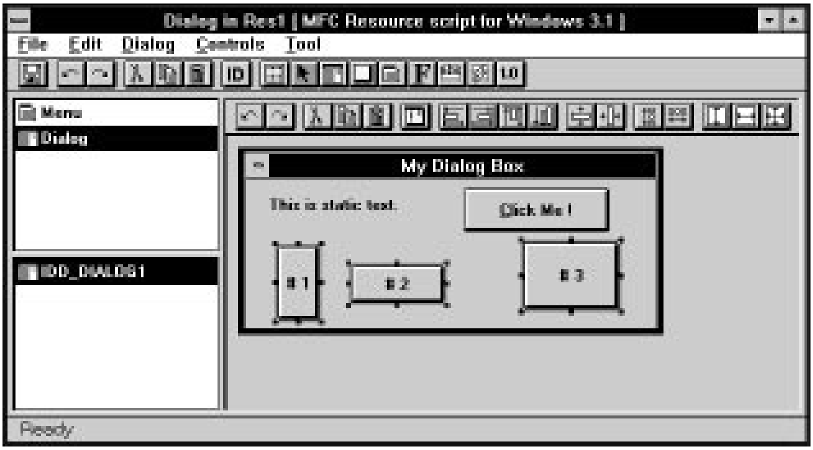


Figure 7-21 Multiple selection of controls

Now you can perform any of the following alignment operations from the **Controls** menu:

- To make all the buttons the same size as the standard, choose **Both** from the **Make Same Size** submenu.
- To align the tops of the buttons with the top of the standard, choose **Top** from the **Align** submenu.
- To even out the horizontal spacing between the buttons, choose **Horizontal** from the **Space Evenly** submenu.

You also can move the selected controls as a unit by holding down the left mouse button and dragging them to a new location.

Closing the dialog box editor

After you finish working with a dialog box, close the Dialog editor by choosing **Close Editing** from the **File** menu. The Browser window is updated to show the new resource (Figure 7-22).

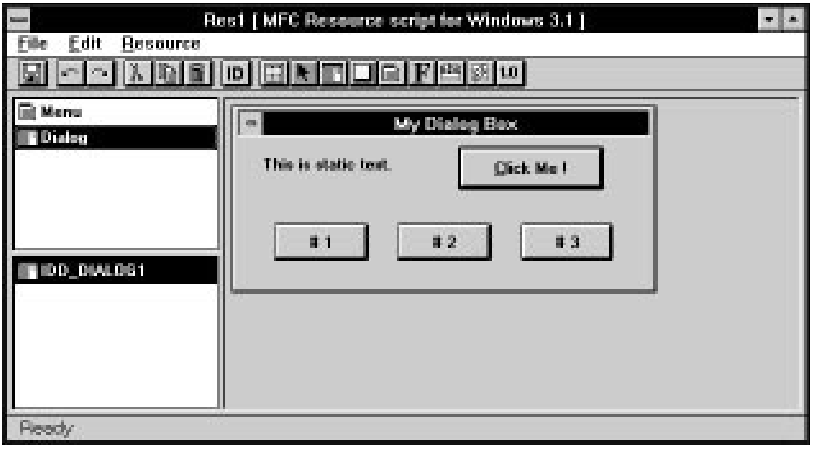


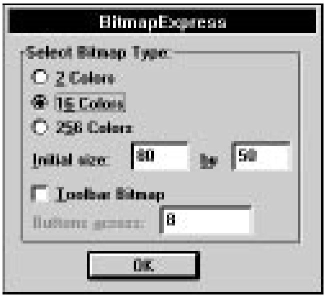
Figure 7-22 Browser window after creating dialog box resource

After creating a dialog box resource, you can use ClassExpress to create a corresponding dialog box class, set up the class's message map, and implement dynamic data exchange and validation. ClassExpress can be run directly from the Browser window's **File** menu. For further information, see Chapter 18, “More about ClassExpress,” as well as Chapter 13, “Lesson 4: Add Messages with ClassExpress,” and Chapter 14, “Lesson 5: Add a Dialog Box with ClassExpress.”

Creating a new bitmap resource

A bitmap is a picture that may be used for informational or decorative purposes. The Bitmap editor contains the functionality of a typical paint program, allowing you to create pictures of any size in 2, 16, or 256 colors.

To create a new bitmap resource, choose **New Bitmap** from the **Resource** menu of the Browser window. The BitmapExpress dialog box opens (Figure 7-23).



BitmapExpress

Select Bitmap Type:

○ 2 Colors
● 16 Colors
○ 256 Colors

Initial size: 80 by 50

☐ Toolbar Bitmap

Buttons across: 8

Figure 7-23 BitmapExpress dialog box

BitmapExpress allows you to set the number of colors and the initial size of the bitmap. If you are creating a bitmap for an MFC toolbar, you also can specify the number of buttons. For now, accept the defaults and click OK. The Bitmap editor opens in the right-most pane of the Browser window (Figure 7-24).

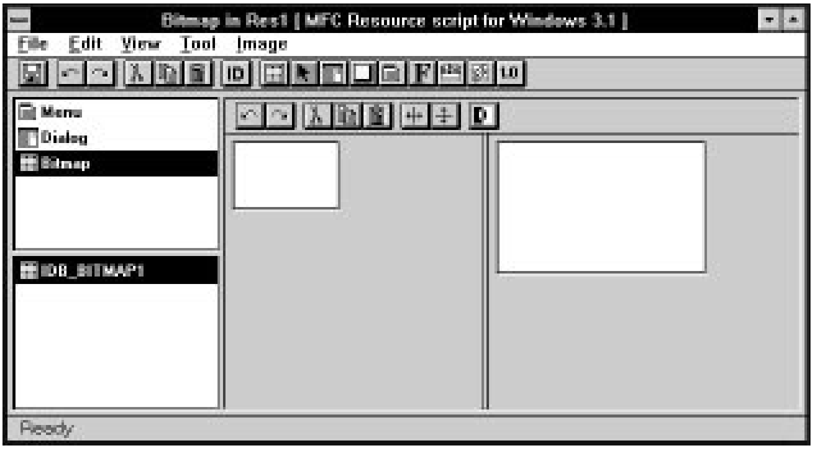


Figure 7-24 Bitmap editor open in Browser window

The Bitmap editor window is divided into two panes. You can draw in either. To change the relative sizes of the panes, click and drag the bar separating the panes.

The Bitmap editor toolbox (Figure 7-25) also opens. This toolbox lets you choose graphics tools (also available in the **Tool** menu), set foreground and background colors, and select the line width and background pattern.

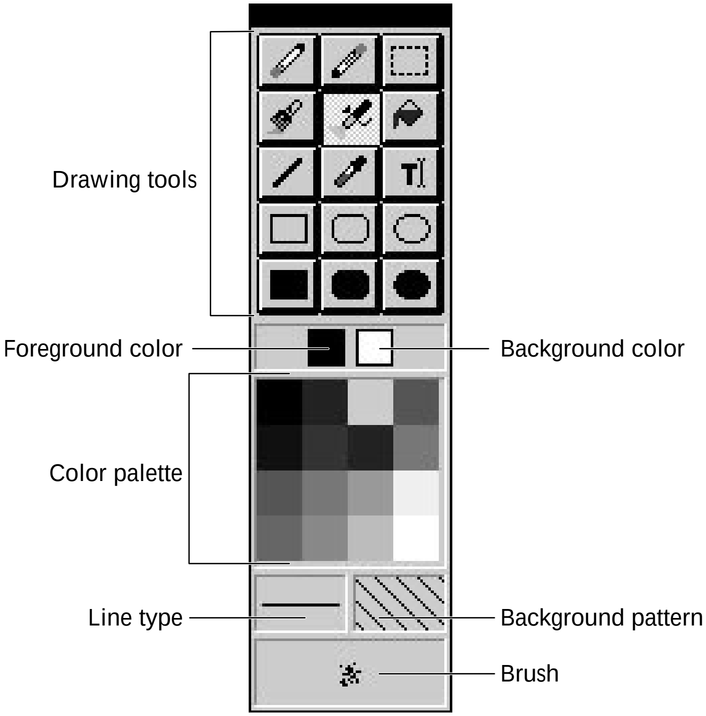
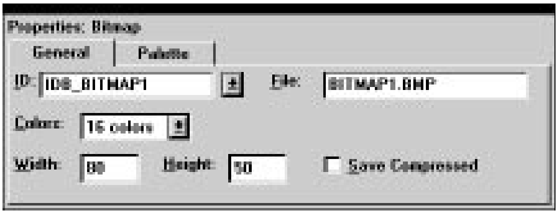


Figure 7-25 Bitmap editor toolbox

The Property Sheet shows the General and Palette properties of the bitmap (Figure 7-26). You can page between the two groups of properties by clicking on the tabs. The File field specifies the bitmap file; bitmaps are included in resource scripts by reference. If you want to change the bitmap file's name, type a new name into the File textbox.



ID: IDB_BITMAP1
File: BITMAP1.BMP
Colors: 16 colors
Width: 80
Height: 50
☐ Save Compressed

Figure 7-26 Properties of a bitmap resource

Changing bitmap size

To change the bitmap size, change the width and height in the Property Sheet or resize the bitmap directly by dragging one of the handles along the right and bottom edges of the bitmap.

Selecting colors and patterns

You can select drawing color and patterns from the toolbox in the following way:

- Foreground color: Click on the color in the color palette with the left mouse button.
- Background color: Click on the color in the color palette with the right mouse button.
- Line type: Click in the line type display and select a line type from the pop-up menu.
- Background pattern: Click in the background pattern display and select a pattern from the pop-up menu.
- Brush: Click in the brush display and select a paintbrush or spray brush from the pop-up menu. The menu is only available when the Paintbrush or Spray brush tool is selected; it is also available by right-clicking on these tools.

Drawing tools

The following drawing tools are available in the toolbox or in the **Tool** menu:

- Eraser: Removes all or part of the image
- Pen: Draws individual pixels or freehand lines
- Selection tool: Selects a rectangle that may be cut, copied, flipped, or inverted
- Brush: Paints freehand lines with the selected brush
- Spray brush: Paints patterns of pixels
- Paint can: Floods an area with color
- Line: Draws straight lines

- Eye-dropper: Picks up a color from the image
- Rectangles and ovals: Draw outlines or solid shapes

To select a drawing tool, click on the tool in the toolbox. To use a tool, click or click and drag (as appropriate) with either the left or the right mouse button. Tools make use of the current line type and background pattern whenever possible. Using the right mouse button to click or click and drag reverses the roles of foreground and background colors.

Zoom and grid

You can work with an image at normal size or zoom by factors of 2, 4, or 8. To zoom, click in the pane containing the image, then choose which **Zoom** command you want from the **View** menu. You can set one image at normal size for reference and a second image at a larger size for easier drawing.

To help finish the details of your image, you can display a pixel grid in the image. A pixel grid can be displayed only when the zoom factor is 4 or 8. To show a pixel grid, choose **Grid** from the **View** menu.

Closing the bitmap editor

After you are finished with the bitmap, close the Bitmap editor by choosing **Close Editing** from the **File** menu. The Browser window is updated to show the new resource (Figure 7-27).

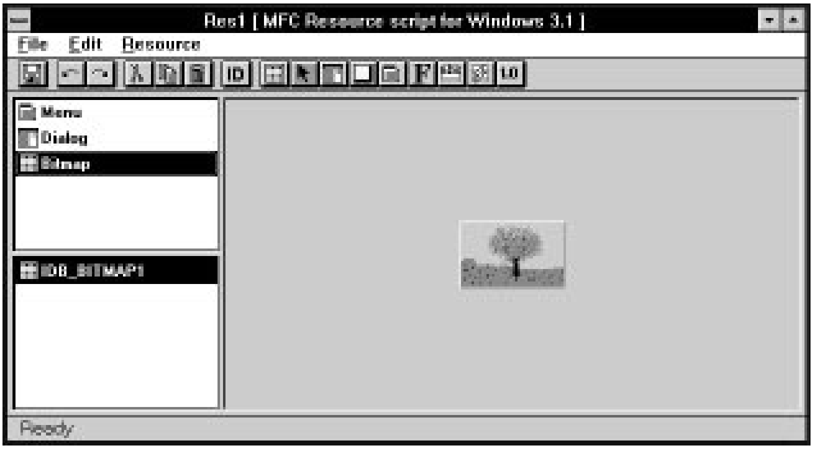


Figure 7-27 Browser window, after creating bitmap resource

Useful ResourceStudio features

You will find a number of ResourceStudio features useful as you create and edit different resources. These include toolbars, the undo and redo functions, and the dragging and dropping of items.

Toolbars and pop-up menus

ResourceStudio makes extensive use of toolbars. Toolbar icons offer quick access to frequently used menu items. For a list of toolbar commands, see the “ResourceStudio Reference.”

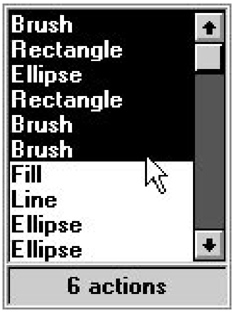
In many parts of ResourceStudio, clicking with the right mouse button opens a pop-up menu. The contents of the menu depend on the item clicked and on the current mode or tool. Pop-up menus are a convenient way to access commands available from the main menu.

Undo and redo

In ResourceStudio, you can undo any operation you have performed or redo any operation you have undone. To undo the previous operation, choose **Undo** from the **Edit** menu or click on the Undo icon in the toolbar. Notice that the operation that will be undone is noted within the **Undo** menu item. If instead you want to redo an action that you have just undone, choose **Redo** from the **Edit** menu.

Operations that you have performed are saved in lists. To set the number of operations that are saved (and thus that can be undone or redone), choose **Preferences** from the Shell window’s **Edit** menu. Each Browser window and each resource type’s editor keeps its own undo and redo list.

To view a list of stored operations, click the right mouse button on the Undo or Redo toolbar icon. You can select one or more operations from the list (see Figure 7-28).



Brush
Rectangle
Ellipse
Rectangle
Brush
Brush
Fill
Line
Ellipse
Ellipse

Figure 7-28 List of stored operations in Bitmap editor

Clipboard and drag and drop

ResourceStudio supports cut, copy, and paste of resources and of objects within resources, such as dialog box controls and menu items.

ResourceStudio also supports drag and drop of all items that can be cut, copied, and pasted. To move a resource or resource element, click and drag it to the new location. For example, to move resources from one file to another, open a Browser window for each file, then drag the resources from the first file to the second. To copy rather than move the element, hold down the Control key until you have released the item.

Managing Resource IDs

Windows programs identify resources by a resource name, which can be a string or a number. Windows programs also use numbers to identify commands resulting from menu selections and accelerators and to identify dialog box controls. The number of identifiers used in a single application easily can run into the hundreds. One of the major features of ResourceStudio is its ability to automatically create and manage resource IDs.

Resource ID field

Many property pages contain an ID field for the current resource or resource element. There are three types of resource IDs in ResourceStudio:

- Textual
- Symbolic
- Numeric

Textual IDs

Textual IDs are allowed only for certain resource types. To specify a textual ID, enclose the text in the ID field in double quotes. ResourceStudio warns you if you try to assign a textual ID to a resource type for which it is not allowed.

Symbolic IDs

Symbolic IDs are names that correspond to numbers. By using this type of ID, you work with names that are meaningful, leaving ResourceStudio to keep track of the numbers that match the names. Symbols and their corresponding numbers are saved in the resource header file as `#define` statements. Thus the same symbols can be used to refer to the resources in your application code.

You can assign a symbolic ID either by typing the name into the ID field or by selecting a pre-existing ID from the drop-down list. If you enter a new symbol, ResourceStudio automatically assigns a unique numeric value. If you want to specify the numeric value, follow the symbolic name with an equal sign and the value (for example, `IDD_TEST=451`). You can reassign the value of an existing symbol in the same way.

Numeric IDs

Numeric IDs should not be used in new resource script (`.rc`) files; use symbolic IDs instead. If you create a binary resource file (`.res`), or edit the resources in an existing `.exe` or `.dll` file, you see the resource IDs as numbers, because the symbolic information is not saved for these file types. You can assign a numeric ID by typing the number into the ID field.

Note

> When editing resources in `.exe` or `.dll` files, remember that the compiled source code refers to resources and commands by the numeric (or textual) ID; reassigning or changing IDs may result in incorrect behavior.

Automatic creation of resource IDs

ResourceStudio automatically creates a new resource ID every time one is needed. The new ID is unique within its context: for resource elements, the ID is unique within the resource and, for resources, unique within the resource file.

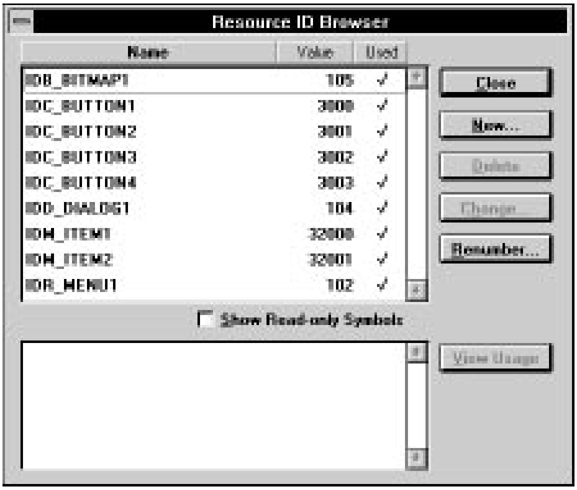
When assigning values to new symbols typed in by the user or created automatically, ResourceStudio uses different ranges for various purposes:

- Resource item range (100-1999): Used for resources within a resource file
- User range (2000-2999): Used for resource IDs created by the user from within the **Resource ID Browser** dialog box (see below)
- Control range (3000-31999): Used by the Dialog editor for dialog controls
- Command range (32000-65535): Used by menu and accelerator editors for menu commands

Resource ID browsing

To browse and modify resource IDs, choose **Edit Resource IDs** from the **File** menu.

The **Resource ID Browser** dialog box opens (Figure 7-29).



Name	Value	Used
IDB_BITMAP1	105	✓
IDC_BUTTON1	3000	✓
IDC_BUTTON2	3001	✓
IDC_BUTTON3	3002	✓
IDC_BUTTON4	3003	✓
IDD_DIALOG1	104	✓
IDM_ITEM1	32000	✓
IDM_ITEM2	32001	✓
IDR_MENU1	102	✓

Figure 7-29 Resource ID Browser dialog box

The upper listbox shows all the symbolic resource IDs in the resource file. Symbols can be sorted by name, value, or used/unused state. Click on an ID to show a list of resources using the ID in the lower listbox. You can open a resource shown in the lower listbox by double-clicking on it, or by selecting it and clicking View Usage.

The remaining buttons perform the following functions:

- New: Opens the **New Resource ID** dialog box. Here you can create a new symbol and assign a value. By default, the value is in the User range (2000-2999).
- Delete: Deletes the currently selected symbol. A symbol may only be deleted if it is not in use.
- Change: Opens the **Change Resource ID** dialog box. Here you can change the symbol name or value.
- Renumber: Performs a renumbering operation on all symbolic IDs. Each symbol is examined and a new value is assigned based on how it is used. If a symbol is used by items whose values are usually in different ranges, a dialog box asks you to decide the range in which to place the value.

Testing an Application

8

This chapter provides an overview of building, running, and debugging an application in Symantec C++. Whenever you finish making any significant addition or change to the source files of an application, it is wise to test the result and verify that you have achieved your goals. To do so, you must first build (or rebuild) the executable so that it incorporates your modifications. Then you must observe its behavior—either by tracing through modified sections, or at least by running the application under the control of the debugger.

This entire phase of the development cycle can be carried out in the IDDE. Because all debugging in Symantec C++ takes place within the IDDE, you do not have to leave the IDDE and run a separate debugger. The IDDE itself provides a wealth of tools for examining all facets of your application's structure and behavior.

Two other chapters present detailed information about the debugging capabilities of Symantec C++. Chapter 23, "Controlling and Configuring the Debugger," describes the commands you use when debugging, and the options available in the IDDE for configuring the debugging environment. Chapter 24, "Commands Available in Debugging Mode," describes the commands available in each of the debugging windows.

Debugger Highlights

The integrated debugger:

- Provides a Windows graphical user interface
- Debugs Windows applications under Windows on the same screen

- Debugs DOS applications running in a DOS box under Windows 3.1
- Takes advantage of virtual memory, letting you debug large DOS applications under Windows 3.1
- Debugs Win32s applications (in the Win32s IDDE, `scw32s.exe`) under Windows 3.1
- Debugs 32-bit Windows and character-mode console applications (in the Win32 or 32-bit IDDE, `scw32.exe`)
- Provides a graphical representation of data structures
- Supports high-speed, hardware watchpoints, and breakpoints
- Lets you drag and drop to execute commands

Choosing an Environment for Debugging

There are three versions of the IDDE included with Symantec C++. An icon for each version you install is available in your Symantec C++ program group. These different versions of the IDDE are essentially identical, except that they allow you to debug different kinds of executables:

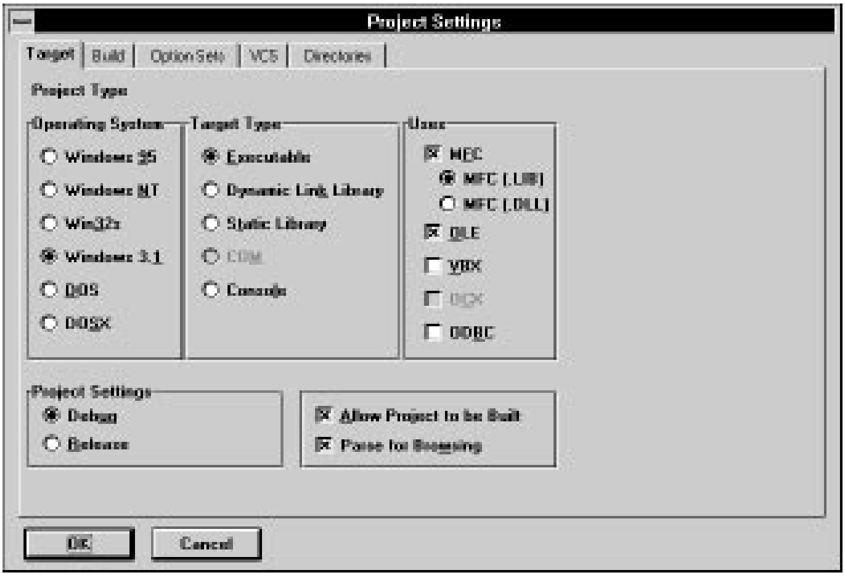
- To debug a Win32 executable under Windows 95 or Windows NT, use the Win32 IDDE (`SCW32.EXE`).
- To debug a 16-bit executable, use the 16-bit IDDE (`SCW.EXE`).
- To debug Win32s executables under Win32s, use the Win32s IDDE (`SCW32S.EXE`).

Building a Project

To test an application, you must first build the executable file, which is the file you test. Your project specifies the source and library files needed to build the executable, as well as options that control the build process. Windows executables also incorporate such resources as menus, dialog boxes, icons, and bitmaps; these are defined in resource files. The building process begins by compiling the source files into object files. Object files are then linked with library files to create the executable file.

Selecting the project type

You must specify the type of executable you plan to build. You do so by choosing **Settings** from the **Project** menu and clicking on the Target tab to select the Target page, as shown in Figure 8-1.



Project Type

Operating System
○ Windows 95
○ Windows NT
○ Win32s
● Windows 3.1
○ DOS
○ DOSX

Target Type
● Executable
○ Dynamic Link Library
○ Static Library
○ COM
○ Console

Uses
☒ MFC
● MFC (.LIB)
○ MFC (.DLL)
☒ OLE
☐ VBX
☐ OCX
☐ ODBC

Project Settings
● Debug
○ Release

☒ Allow Project to be Built
☒ Parse for Browsing

Figure 8-1 Target page of the Project Settings dialog box

On this page, choose the operating system your target executable will run on, and the target type. Not all target types are available with all operating system types.

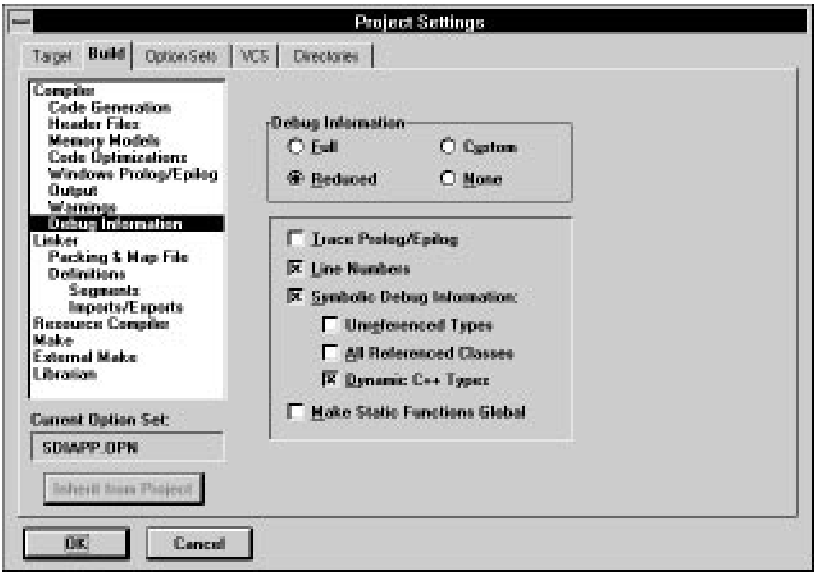
The Target page also presents the option of building a Debug or Release version of the executable. If you want to debug your executable, select the Debug option.

For more information on the Target page of the Project Settings dialog box refer to Chapter 15, “More about Projects and Workspaces.”

Setting compiler and linker options for debugging

When you indicate that a debugging version should be built, the IDDE sets the appropriate compiler and linker options, and also prevents debugging information already in object files from being discarded.

To verify that the these settings are appropriate, choose **Settings** from the **Project** menu, and then click on the Build tab to select the Build page of the **Project Settings** dialog box. Select the Debug Information subpage by clicking on that label in the left listbox. The Debug Information subpage is shown in Figure 8-2.



Debug Information
○ Full
○ Custom
● Reduced
○ None

☐ Trace Prolog/Epilog
☒ Line Numbers
☒ Symbolic Debug Information:
  ☐ Unreferenced Types
  ☐ All Referenced Classes
  ☒ Dynamic C++ Types
☐ Make Static Functions Global

Current Option Set: SDIAPP.OPN

Figure 8-2 Debug Information options

For more information about project, compiler, and linker options, see Chapter 15, "More about Projects and Workspaces."

Building executable files

The IDDE provides three ways to build executable files from your project: by performing a standard build, by rebuilding the entire project, or by linking the existing object files.

Performing a standard build

This is the most common of the three build options. To begin, choose **Build** from the **Project** menu. The IDDE recompiles only those files that changed since you last compiled them, then links the project. The IDDE displays any errors in the Output window.

The IDDE Make facility is used by default to determine the steps needed to build your project. It is functionally identical to the DOS command-line Make utility SMAKE, included with Symantec C++. If you need to use a different, DOS command-line Make utility, such as NMAKE or PolyMake, you can do so; see Chapter 16, “More about Project Build Settings,” for details on how to use an external Make program.

Rebuilding the project

If you want to recompile every file in your project—even those files that are up-to-date—choose **Rebuild All** from the **Project** menu. The IDDE recompiles all files in your project, whether they’ve been edited recently or not, then links the project.

Typical situations in which you use the **Rebuild All** command include:

- When you have changed some of the project options
- When you suspect corrupted object `.obj` files
- When you want to create a final build

Linking the project

If you want to build a program with the existing object files but without recompiling your source files, choose **Link** from the **Project** menu. The IDDE links the object files.

Typical situations in which you use the **Link** command include:

- When you have added a new library `.lib` file
- When you wish to build from object `.obj` files only
- When you have changed linker options

Other project options

The Build page of the **Project Settings** dialog box offers additional settings that you may find useful for precompiling headers and generating an assembly listing.

Precompiling headers

The Header Files subpage lets you precompile one or all of the header files that are included by source files in your project. Precompiling a header is useful when the header file changes infrequently or not at all between builds, or when a header file is included by most of the source files in a project. Precompiled headers speed up the build process, especially with large header files. For example, `windows.h` can be beneficially precompiled because it is large and is not likely to change.

Generating an assembly listing

The Assembly Listing (.COD) check box on the Output subpage lets you create a `.cod` file that contains the assembly language code into which the compiler converts your source code. In this file, C++ source statements are preserved as assembly language comments. Each commented statement precedes the assembly language code to which it corresponds.

Running a Project

You can run the application that your project produces without leaving the IDDE. Commands that run your application ask you to build it if required.

If your program requires no command-line arguments, choose **Execute Program** from the **Project** menu. The IDDE launches your application.

If your program requires arguments, first choose **Arguments** from the **Project** menu. The IDDE displays the dialog box in Figure 8-3.

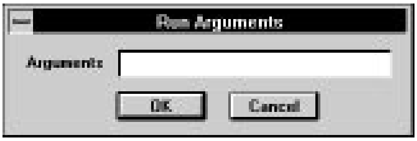


Figure 8-3 Run Arguments dialog box

Enter your arguments—not the program name—in the dialog box. For example, entering:

```
*.* /s
```

launches your program with the arguments `*.*` and `/s`.

After specifying the arguments, click OK. Then run your application as described above.

Quick Start: Debugging an Application

This section provides a brief description of how to perform common debugging tasks. You can find a complete presentation of the debugging capabilities of Symantec C++ in Chapter 23, “Controlling and Configuring the Debugger,” and in Chapter 24, “Commands Available in Debugging Mode.” In Chapter 23, see especially the sections “Commands on the Debug Menu” and “Debug Toolbox Icons.”

After building your project, you can enter debugging mode by choosing **Start/Restart Debugging** (F4) from the **Debug** menu. (That is, you may either choose this command from the menu, or type its keyboard shortcut given in parentheses.) Alternatively, you can click on the Restart Debugging icon in the Debug toolbox. Any open Source windows change to debugging mode. Your application executes to the breakpoint set automatically on `WinMain` (for Windows applications) or `main` (for DOS applications and 32-bit console applications).

Other debugging windows, such as the Function window and Data/Object window, can be opened as needed from the Views toolbox or from the **Goto View** submenu of the IDDE’s **Window** menu.

In debugging mode, the **Start/Restart Debugging** command on the **Debug** menu remains available. This command restarts the program. The **Stop Debugging** command on the **Debug** menu exits debugging mode, and returns the IDDE to editing mode.

Stepping through code

To step to the next source code statement, choose **Step Into** (F8) from the **Debug** menu. Alternatively, you may click on the Step Into icon in the Debug toolbox. If the current line is a function call, and debugging information is available for that function, you will trace into that function. To step over a function call, choose **Step Over** (F10) from the **Debug** menu, or click on the Step Over icon in the Debug toolbox.

Note

If you accidentally step into a function call when you meant to step over it, you can move to where you intended to be—the statement following the call—by choosing **Return from Call** from the **Debug** menu. This command executes to the current function’s return address (unless a breakpoint or watchpoint is encountered before control reaches that point).

Setting and clearing breakpoints

Setting a breakpoint on a source code statement causes your program to stop when execution reaches that statement. The debugger regains control, and you may again perform any debugging mode actions.

To set a breakpoint on a statement, first click on that statement in the Source window to make it the selected line. Then choose **Set/Clear Breakpoint** (F9) from the Source window pop-up menu (which appears when you click the right mouse button in the Source window), or click on the Toggle Breakpoint icon in the Debug toolbox. These commands act as toggles; repeating them clears the breakpoint.

The Source window pop-up menu, as it appears in debugging mode, is shown in Figure 8-4.

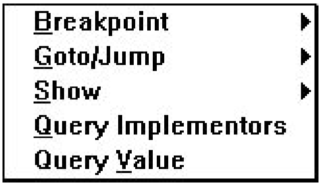


Figure 8-4 Source window pop-up menu in debugging mode

A breakpoint symbol in the left margin of the Source window indicates a breakpoint on the adjacent line. You can also clear a breakpoint by dragging the symbol out of the Source window.

Executing up to a statement

To execute up to the currently selected line, choose **Go Until Line** from the Source window pop-up menu. Double-clicking on any line in the Source window causes your program to run until execution reaches that line.

Viewing a list of functions

The Function window lists the functions in the current module or in all modules. Toggle the display by choosing **Current Module** or **All Modules** from the **View** menu. Double-click on any function to view that function in a Source window.

You can easily set a breakpoint at the beginning of a function from within the Function window by choosing the **Set/Clear Breakpoint** (F9) command from the **Bpt** menu.

Examining the values of variables

The Data/Object window lets you view either global data or the variables local to a function. The **Show** menu of the Function window has commands **Global Data** and **Local Data** that select the type of data displayed in the Data/Object window. If you choose **Local Data**, the Data/Object window shows the local variables for the function currently selected in the Function window, provided that function is in the call chain. (If a function is not in the call chain, it has no local data.) The display can also be toggled by choosing **Local/Global Data** from the **View** menu of the Data/Object window.

Examining the call chain

The Call window displays the stack of function calls in your code that have not yet returned. The list is presented in reverse chronological order from most to least recent; thus, `WinMain` (or `main`) is at the bottom of the list.

The Call window's **Show** menu contains commands that let you zoom in on a particular function in the chain. For example:

- The **Source** command updates the Source window and displays the statement that is executing within the selected function.
- The **Data** command updates the Data/Object window and displays local data for the selected function.

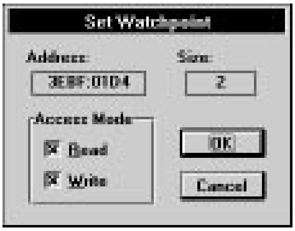
Setting watchpoints

A watchpoint specifies that execution of your program should stop when a particular variable or memory location is written to or read from. This capability is essential for detecting problems arising from wild pointers, for example, which can manifest themselves in extremely elusive and seemingly random behavior.

Symantec C++ debuggers are designed to take advantage of hardware watchpoints provided by 386 and higher microprocessors. The Symantec C++ installation program may install the file `SCWDEBUG.386` in the `[386Enh]` section of `system.ini` if your system needs it to allow the use of hardware watchpoints. Because watchpoints are implemented with hardware assistance, using this powerful tool imposes no speed penalty on program execution.

Setting a watchpoint on a variable

Using the methods described in the sections above, make sure that the desired variable is displayed in the Data/Object window. Click on the line referencing the variable to select it; the line should be highlighted. Choose **Set Watchpoint** (Ctrl+W) from the **Watch** menu of the Data/Object window. This opens the **Set Watchpoint** dialog box, shown in Figure 8-5.



Address: 3E8F:01D4
Size: 2
☒ Read
☒ Write

Figure 8-5 Set Watchpoint dialog box

The debugger maintains information about the type of variables and their location in memory. The selected variable's type determines the size of the watchpoint. The **Set Watchpoint** dialog box displays the address and size of the watchpoint as noneditable fields, and provides options for breaking on a Read access, Write access, or both. To clear the watchpoint, choose **Clear Watchpoint** from the **Watch** menu of the Data/Object window.

Warning

Exercise caution when setting a watchpoint on an automatic variable (that is, a local, nonstatic variable). If you attempt to set such a watchpoint, the debugger warns you by displaying a message on the status line. You should clear the watchpoint by the time the function whose local variable you are watching returns. If you don't, Windows itself can use the stack location subsequently, thus triggering the watchpoint and causing Windows to crash.

In addition to setting watchpoints on variables, watchpoints can also be set on locations in memory using the Memory window, whose **Watch** menu is identical to that of the Data/Object window.

Letting your program run until the next breakpoint

You can make your program run until execution reaches a breakpoint by choosing **Go until Breakpoint** (F5) from the IDDE's **Debug** menu, or by clicking on the Go until Breakpoint icon in the Debug toolbox. You can use the Breakpoint window to see at a glance where breakpoints have been set.

Letting your program run until it terminates

You can make your program run until it terminates—ignoring any breakpoints—by choosing **Go until End** from the **Debug** menu.

Interrupting execution of the debugged application

Use the Ctrl+Alt+SysRq key combination to break execution of the application being debugged and return control to the debugger. It may be necessary to press this key combination a few times before the debugger regains control. (If you use this key combination to break execution when control is within Windows itself, it may be difficult for the debugger to step out of Windows code.)

The technique is most useful when you suspect that your own code is hung. Returning to the debugger lets you examine your program's state, which may be one that you thought impossible. Inspecting the values of variables and the call chain can suggest how to identify and eliminate the source of the error.

Note

Use Ctrl+Alt+F11 to break execution of Win32s applications.