# CA-Clipper®

For DOS

Version 5.3

# Workbench User Guide

June 1995

COMPUTER ASSOCIATES®
Software superior by design.

PRINTED ON
RECYCLED PAPER

Printed in the United States of America
Computer Associates International, Inc.
Publisher

# Contents

## Chapter 1 : Introduction



## Chapter 2 : Working in the Workbench

# Chapter 3 : Browsing Applications, Modules, and Entities

# Chapter 4 : Using the Form Editor

# Chapter 5 : Using the Menu Editor

# Chapter 6 : Using the Source Code Editor



# Chapter 7 : Defining Data Servers and Field Specs

# Chapter 8 : Debugging Your Applications

# Appendix A : Workbench File Types



# Appendix B : Using Linker Template Files



# Index

# Chapter 1 Introduction

## In This Guide

This guide explains how to use the various features of the CA-Clipper Workbench component—a user-friendly, Windows GUI interface to CA-Clipper Release 5.3.

This guide is organized into the following chapters:

1. Introduction

   Details the conventions and symbols used in presenting the information in this guide. Because they are vital to your understanding of this guide, it is highly recommended that you take the time to familiarize yourself with them.

2. Working in the Workbench

   Presents the CA-Clipper Workbench and its various components, explains how to customize and save the current desktop, and describes how to set default system options.

3. Browsing Applications, Modules, and Entities

   Explains how to use the Application, Module, and Entity Browsers to create, view, and modify your applications, modules, and entities.

4. Using the Form Editor

   Explains how to create forms for your applications.

5. Using the Menu Editor

   Describes how to create custom menus, as well as standard, predefined menus, and how to add them to your applications.

6. Using the Source Code Editor

   Demonstrates how to enter and edit source code in the Workbench.

7. Defining Data Servers and Field Specs

   Describes how to create data servers and field specs, and to maintain database ancillary information using the DB Server and FieldSpec Editors.

8. Debugging Your Applications

   Demonstrates how to set various debugging options, and test and debug your Clipper DOS applications from within the CA-Clipper Workbench.

Appendix A: Workbench File Types

Lists the different types of files generated or used by the Workbench.

Appendix B: Using Linker Template Files

Explains how to use a linker template file to define the exact commands that are executed during the link phase of an application build.

Index

# What You Need to Know

In addition to an understanding of basic programming concepts, this guide assumes that you are familiar with Microsoft Windows terminology and navigational techniques, including how to work with standard Windows items like menus, dialog boxes, the Clipboard, and the Control Panel. If you are unfamiliar with Windows, please refer to your Windows documentation before using the CA-Clipper Workbench.

**Note:** In general, when this guide indicates a procedure using toolbar buttons or mouse actions, it takes for granted that you know the alternative procedure, using only the keyboard. For example, you will be directed in most cases to "click the Find toolbar button," rather than "select the Edit Find command, press Alt+F3, or press Alt+E, F."

This guide also assumes that you have read the *Getting Started* guide and are, therefore, familiar with its various features, and that you have worked through its "hands-on" tutorial.

# General Typographic Conventions

This guide also employs several typographic conventions (such as capitalization or italic formatting) to distinguish between language elements.

### Key Names

The names of keys, such as Enter, Ctrl, and Del, appear in the document as they do on your keyboard, where possible. Also, the direction arrow keys are referred to individually by name (for example, Up arrow or Left arrow).

### Key Combinations

Whenever two keys are joined together with a plus (+) sign (for example, Ctrl+R), you should hold down the first key while pressing the second key to complete the command. Release the second key first.

### Key Sequences

When keys are separated by a comma (,), press them in the sequence indicated. The keystroke sequence Alt+E, C, for example, indicates that you should hold the Alt key down while pressing the E key, release them both, and then press and release the C key.

### User Input Examples

The following conventions are used for user input:

- Literal information (text that the user must enter exactly as shown) is shown in bold:

  Insert the diskette into drive A and type **a:\install**.

- Placeholder text (variable information a user must enter) is denoted by a bold and italic typeface:

  Enter **login *username***.

UPPERCASE

The following appear in uppercase:

- Commands (like CLEAR MEMORY)
- Keywords (for example, AS, WORD, and INT)
- Reserved words (for example, NIL, TRUE, and FALSE)
- Constants (for example, NULL_STRING and MAX_ALLOC)

Mixed Case / Initial Capitalization

The following are displayed using mixed case:

- Function and procedure names (like AOpen())
- Variable names (like *aInfo*)

*Italic*

Variable names are displayed in italic in syntax (for example, Abs(<*nValue*>)) and when referring to them in the discussion text.

Cross References

The following conventions are used:

- Guide name in italic:

  See the *Programming and Utilities Guide.*

- Chapter name in double quotes:

  See "Using the Source Code Editor" in the *Workbench User Guide.*

- Section name as it appears in the document:

  Also see the Adding Predefined Menus section.

# Getting Help


The CA-Clipper Workbench provides online Help, which can be used to display information on your PC as you work. You can use any of the following Help menu commands:

| Menu Command | Description |
|---|---|
| Index | Displays an index of available help topics about the Workbench. |
| Context Help | Allows you to get context-sensitive help for an item or area currently displayed on your screen. |
| Help Using Help | Describes how to use the Windows online Help system. |

In the Workbench you can also receive context-sensitive help for a menu or menu command by pressing either the F1 key or the Shift+F1 key combination.

Press Shift+F1 to receive context-sensitive help for most dialog boxes and windows. Note that some dialog boxes also have a Help push button.

Additionally, when the Source Code Editor is open, you can receive context-sensitive help for the keywords, commands, classes, and functions in a selected module or entity. Simply highlight the keyword, command, class, or function and press the Shift+F1 key combination.

# Chapter 2
# Working in the Workbench



## In This Chapter

This chapter introduces you to the CA-Clipper Workbench and its various features, explains how to customize and save the current desktop, and describes how to set default system options.

## Workbench Basics

The Workbench is a flexible, intuitive, and powerful environment for creating CA-Clipper DOS applications. Almost all features of the Workbench—the browsers, the visual editors, the source code editor, the compiler, the debugger—are available at the touch of a button from almost any window. For example, you can:

- Create applications and modules.
- Open and work with multiple applications.
- Double-click on any entity—function, menu, form, data server—to launch the editor associated with that entity.
- Modify a single entity and click the Build button to rebuild only the parts of the application affected by that change.
- Generate an executable file, and execute it or debug it from within the Workbench.

## Arranging and Manipulating Windows

The Workbench permits you to open and simultaneously work with multiple windows and editors. The only exception is the Application Browser window, since there is only one.

You can switch between the currently open windows by simply clicking them with the mouse or by choosing them from the list displayed on the Window menu. You can also use the commands on the Window menu to reformat the current window display (for example, to tile or cascade all open windows).

In addition, almost all windows can be resized, repositioned, and minimized/maximized using the standard Windows techniques.

## The Toolbars

Almost every window, browser, and editor in the CA-Clipper Workbench contains a customized toolbar that provides buttons as shortcuts to commonly used menu commands. Most toolbars have the same set of common buttons on the left, and buttons specific to the particular editor or browser on the right.

For example, the toolbar in the Application Browser contains buttons for creating a new application, opening an existing application, and setting default application options (among others). The buttons in the Menu Editor toolbar, on the other hand, allow you to create a new menu; add predefined menus automatically; cut, copy, paste, and insert menu items; promote and demote items in a menu's hierarchy, and so on.

**Tip:** If you want to know what a toolbar button does, simply point to it—a description appears in the status bar, located in the lower-left portion of the window.

## The Status Bars

The status bar of almost every window, browser, and editor in the Workbench displays helpful, informative text about various system features, giving you a quick summary or reminder about their contents, or the actions they perform.

For example, when moving the mouse over the buttons in a toolbar, the status bar contains a description of what each button does. Likewise, highlight a menu command to display a description of what it does in the status bar, or move the mouse over the entities in an Entity Browser to view the first line of each entity in the status bar (if you select the Show Entity on Status Bar option when you set your system options).



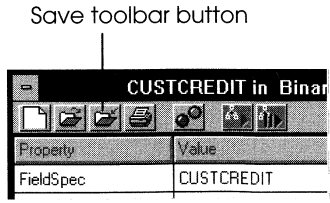
## Saving, Building, and Executing

At almost any time and location, you can save, build, and/or test the current application, because in almost every browser and editor, toolbar buttons and menu commands are provided for saving, building, and executing.

### Saving Your Work

When you create a new *entity*, you should store it in the *repository* (the repository is discussed in greater detail later in this chapter) and then save any additional edits on a frequent basis.

To initially save a new entity in the repository, click the Save toolbar button:



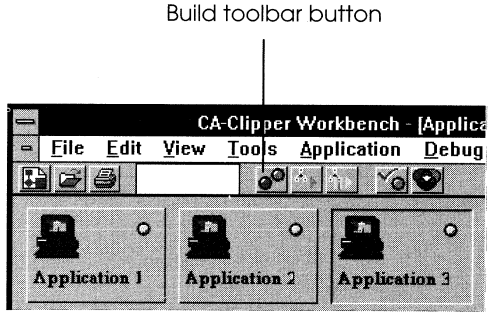

To save any subsequent changes, just click the Save toolbar button again.

Note that when you create an *application* or *module,* it is automatically added to the repository—there is no need to explicitly save these items.

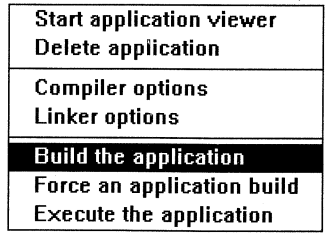
**Note:** Users who have a *write-cache system* enabled may want to periodically select the Save Repository command on the File menu to forcibly flush the buffer to disk. See your online Help reference for details about Save Repository.

## Building an Application

You can build an application at any time by either clicking the Build button on the toolbar or selecting the Build command from the Application menu:

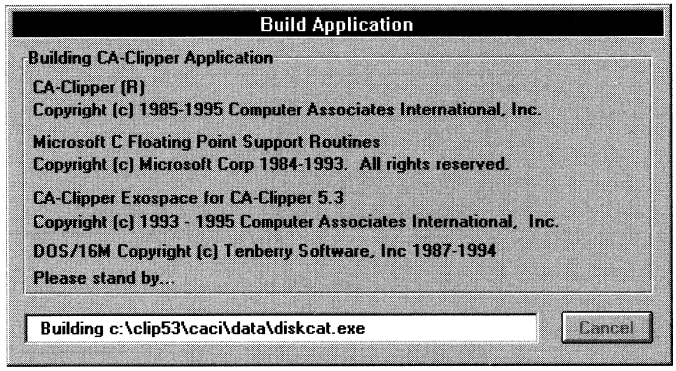



Alternatively, right-click on an application button and select the Build the Application command from the local pop-up menu that appears:



Based on date and time stamps and the system-maintained dependency list, the Workbench automatically rebuilds only those parts of the current application that have been changed, or are affected by changes, since the application's last build and then does any necessary recompilation and relinking.

If the application has *not* been changed, you will receive the following message: "Application is up-to-date." Otherwise, the Build Application dialog box appears, displaying initialization and other build information:

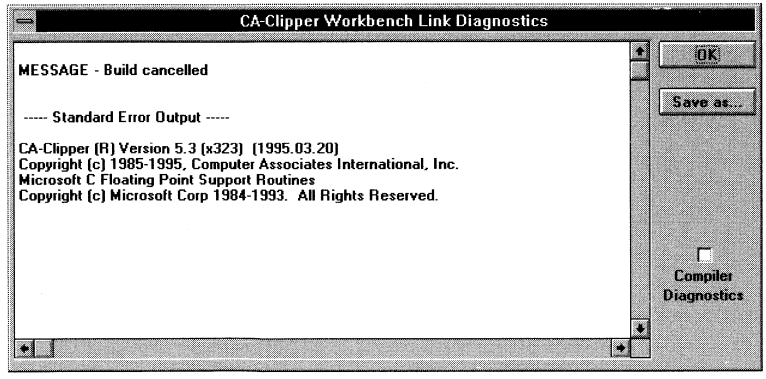



Force Build Command

You may also force the system to build your application *ignoring* date and time stamps. To do this, choose the Force Build command from the Application menu.

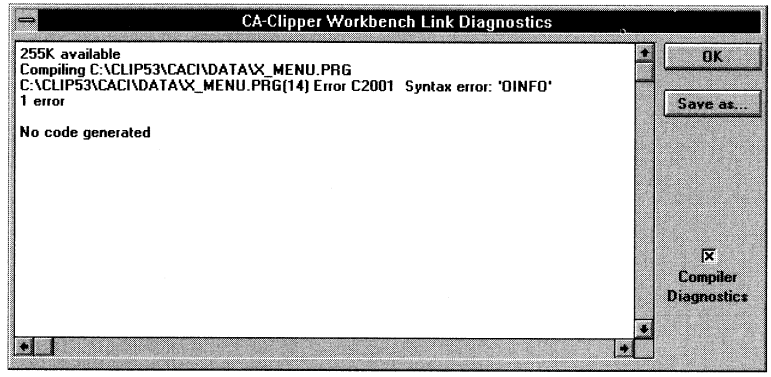
Link Diagnostics

Once the build is done, the following dialog box appears only if there are any link errors or warnings:



**Note:** Refer to the *Error Messages and Appendices Guide* for help with linker error messages.

Note that you can also display compilation information in this dialog box by selecting the Compiler Diagnostics check box. For example:



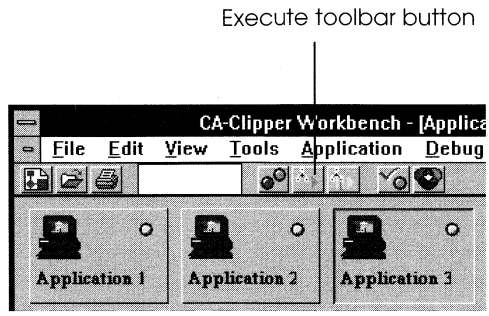

Any compilation errors can then be browsed using the Error Browser. For more information about Error Browsers, see The Workbench Tools section later in this chapter; and for detailed information about compiler error messages, refer to the *Error Messages and Appendices Guide.*

**Tip:** You can save the link and compilation data in a text file. Just choose the Save As push button, and then enter a file name and path for the text file in the Save Link Diagnostics As dialog box that appears.

The manner in which an individual application is built can be controlled using application-specific compiler options instead of system-wide default compiler options. See Setting Compiler Options later in this chapter for details.

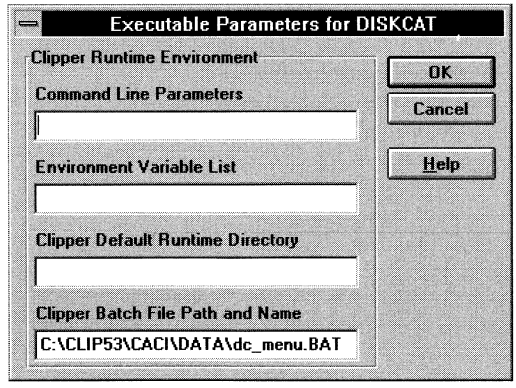
## Executing an Application

After building an application, the easiest way to execute it is to select the Execute toolbar button when the desired application is selected:




Alternatively, choose the Execute command from the Application menu.

The Executable Parameters box appears allowing you to specify your CA-Clipper runtime environment. For example:



The Workbench runtime environment options are described below.

**Command Line Parameters**

Specify any application command line arguments for your CA-Clipper application. The default is none.

**Environment Variable List**

Specify an environment variable for the application.

The default is none.

**Clipper Default Run Directory**

Specify a valid runtime directory for your application. This default runtime directory does *not* necessarily have to be the same directory that contains the executable (.EXE) file.

The default is the executable file path.

**Clipper Batch File Path and Name**

Specify a batch file name and fully qualified path for the generated batch file used to invoke the application with its specified configuration settings.

A default batch file path and name are supplied by the system, as this information is *mandatory*.

**Note:** For more information about defining the runtime environment, refer to the *Programming and Utilities Guide*.

# The Workbench Tools

The CA-Clipper Workbench provides a host of tools for its integrated development environment. There are *browsers*, which let you organize and view the layout of your applications, and *editors*, which allow you to create forms, menus, source code, and data servers. Other tools include the CA-Clipper compiler and debugger.

In the Workbench, all development tools are closely integrated with the *repository*. The repository is where the Workbench stores all application components (i.e., applications, modules, and entities), and it automatically manages the relationships between the various components of an application. For example, double-clicking on an entity automatically invokes the appropriate editor for that entity: the Form Editor or Menu Editor for forms or menu entities, respectively, and the Source Code Editor for code (functions, etc.).



## Browsers

Browsers allow you to view and manipulate the code that is currently stored in your repository in a convenient and organized way. In the Workbench, you can browse:

- Applications
- Modules
- Entities
- Errors

**Tip:** Most browsers in the Workbench can be customized to display a particular subset of code. For example, the Error Browser displays errors in a collapsible/expandable tree structure.

## Application, Module, and Entity Browsers

Applications represent the highest level in the Workbench application hierarchy: *Applications* consist of *modules*, which consist of *entities*. Modules can be of several types: binary, program source (.PRG) files, and header (.CH) files. Entities are components that have distinct names and can be edited (for example: text blocks, functions, forms, data servers, and menus). (Module types and entities are discussed in greater detail in the next chapter, "Browsing Applications, Modules, and Entities.")

The primary browsers in the CA-Clipper Workbench, therefore, follow this same top-down hierarchy. When you start the Workbench, the *Application Browser* is automatically loaded, displaying all of the various applications that currently exist. Double-clicking on an application displays the binary and non-binary modules defined for it in a *Module Browser*. Similarly, double-clicking on a module displays the entities defined for that module in an *Entity Browser*.

You can also choose the Tools Entity Browser command to display all of the entities in the current *application* in the Entity Browser (not just those in the current module). This version of the Entity Browser displays entities by grouping also in a collapsible/expandable tree and allows you to set a name filter.

**Note:** The Application, Module, and Entity Browsers are described in more detail in the "Browsing Applications, Modules, and Entities" chapter.

### Error Browser

You can access the Error Browser after any unsuccessful attempt to build an application, in which case you can view a comprehensive list of all the *compilation* errors and warnings within that application.

The Error Browser displays those entities with errors in a tree structure that is collapsible/expandable. If you then double-click on an entity, you are brought directly to the line in the source code where the error or warning lies.

See the "Debugging Your Applications" chapter for details about the Error Browser.



## Editors

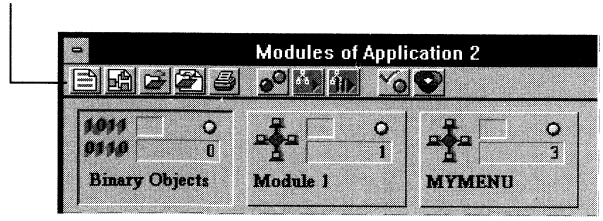
The various Workbench editors allow you to create forms, menus, source code, and data servers easily, conveniently, and efficiently.

### Accessing the Editors

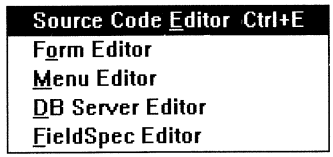
The editors can be accessed either by creating a new entity or opening an existing one.

The most logical place to create a new entity is from within the Module Browser—simply click on the New Entity toolbar button:

New Entity toolbar button



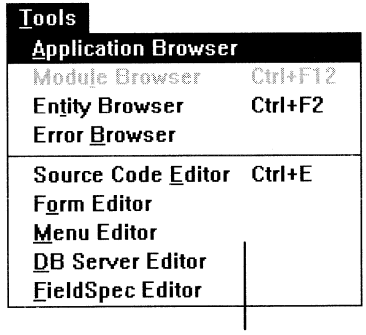
Then select the appropriate editor for the type of entity you want to create from the local pop-up menu that appears:




The selected editor is launched, allowing you to define and save the new entity.

**Tip:** The local pop-up menu shown above appears whenever the New Entity button is selected from any browser's toolbar *except* the Application Browser.

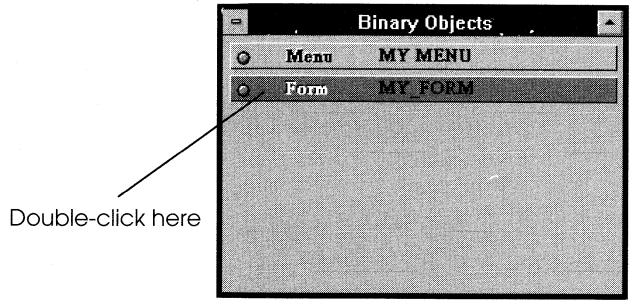
You also can create a new entity by accessing an editor from the Tools menu:




Select an editor appropriate for the new entity

## Opening an Entity

In the Workbench, *opening* an entity simply means to display the entity in its associated editor. To open an existing entity, simply double-click on it, highlight it and then press Enter, or right-click on it and select the Edit Entity command from a local pop-up menu:




For example, double-clicking a form entity in the special "Binary Objects" module in our example invokes the Form Editor, while double-clicking a function entity in a non-binary objects module activates the Source Code Editor. (See Creating Modules in the next chapter for more information about binary and non-binary modules.)

## Creating and Editing Source Code

You can create source code entities at any time in the Workbench in several different ways. First of all, you can access the Source Code Editor directly and manually type in code.

Secondly, and more importantly, when you define entities using the Form and Menu Editors, the Workbench *automatically* generates new modules containing the associated generated .PRG file(s), as well as .CH file(s) for forms. This generated code can then be edited from within the Source Code Editor.

***Important!*** *Modifying the associated .PRG and .CH files created by either the Menu Editor or the Form Editor invalidates the associated binary object, i.e., the image that is displayed subsequently in the appropriate editor is not a reflection of the modified .PRG or .CH file. For example, if you change the caption for a push button on an existing form using the Source Code Editor, you will receive a warning message that the program files have been manually modified the next time you attempt to view this form entity in the Form Editor. Furthermore, when you click OK in response to the warning, you will see the form as it was originally created, not as it was modified.*

The Source Code Editor in all cases displays various information in the source code, such as keywords, literals, and comments, in different colors of your choice for your convenience while editing.

Refer to "Using the Source Code Editor" for details about editing your applications' source code.

## The Visual Editors

Some of the editors in the Workbench are *visual*; that is, you can lay out various graphical user interface (GUI) controls—like push buttons, check boxes, and list boxes—on a form, and create a custom menu using point-and-click, drag-and-drop techniques. Visual feedback is immediate when designing objects in the Menu and Form Editors.

See the following chapters for more information about these editors: "Using the Menu Editor" and "Using the Form Editor."



## The Database Editors

The CA-Clipper Workbench provides a set of database specific information editors: the DB Server and FieldSpec Editors.

### The DB Server Editor

The DB Server Editor allows you to create *data servers* based on the traditional Xbase model of a .DBF file. Data servers are high-level objects used to provide a consistent GUI interface for a database and its ancillary information.

You import an existing database structure into a data server and generate a default set of field specifications that you can optionally modify. Information about the database is stored in the data server along with detailed field information stored in the form of FieldSpec objects. You can then create automatic layouts for data servers in the Form Editor that you can easily modify.

### The FieldSpec Editor

Although the DB Server Editor has built-in mechanisms for defining field specs, the FieldSpec Editor is independent of them and is used to set properties for common field types that can be accessed by multiple data servers. For example, if you specify properties for a Salary field in the FieldSpec Editor, you can simply reuse those properties when describing a Salary field in another data server.

See "Defining Data Servers and Field Specs" for more information about these editors.

## The Debugger

Not only can you build and execute applications at the touch of a button, but you can also debug them just as easily. To start the Workbench's debugger, simply select the Run command from the Debug menu.

The Workbench's debugger allows you to:

- Use one of several execution modes to control the execution of your application while viewing the source code in the Debug source code window
- Evaluate and trace expressions
- Set, reset, and clear breakpoints
- View and modify variables
- Create watch expressions
- View the call stack
- View database and other work area information in a separate window and modify database field values
- View and modify system settings

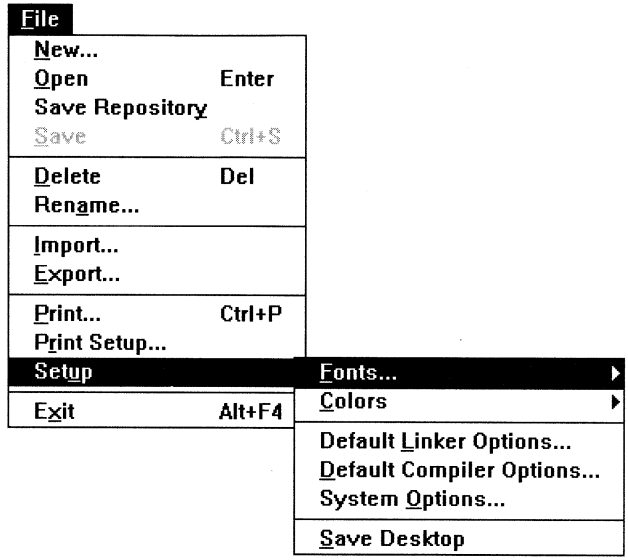
To facilitate debugging, the debugger library, CLD.LIB, is specified by default in the Additional Object Files edit control of the Linker Options dialog box. Therefore, to utilize the debugger for your application, verify that this default setting is unchanged. Additionally, the Debug Information option must be checked in the Compiler Options dialog box for the application.

See the "Debugging Your Applications" chapter for complete information about the debugger.

# Setting System-Wide Options

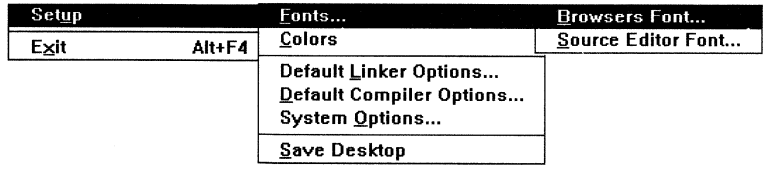


You can set a number of system-wide options for the CA-Clipper Workbench, including compiler options and settings for the Workbench desktop. System setup options can be set and changed by selecting commands from the File Setup menu:

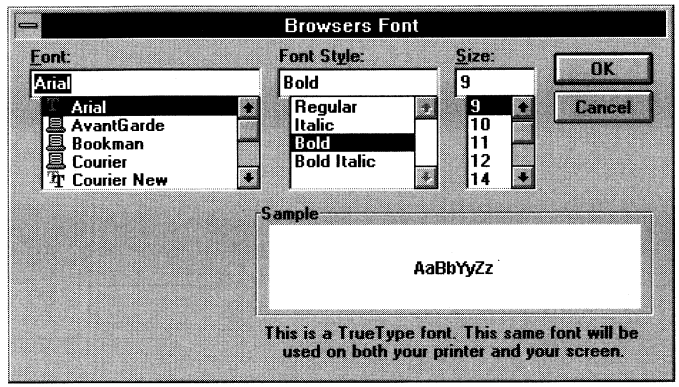



## Selecting Fonts

The Workbench allows you to customize fonts on two levels: for the various browsers and for the Source Code Editor, as indicated by the Setup Fonts menu:



Choosing either command displays a standard Font dialog box. For example:




Browsers

The font you select for the Browsers command is used for the text on the module and application buttons, the standard text in the Entity Browser, and the status bar text in all windows.

Source Code Editor

The Source Editor Font command under the Setup Fonts menu allows you to specify the font to be used for the text displayed in Source Code Editor windows.

## Selecting Source Code Editor Colors

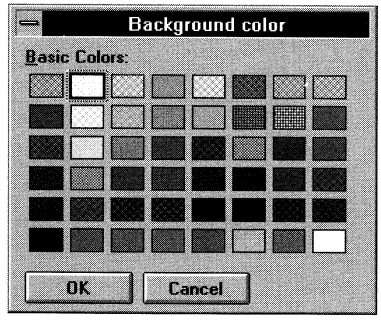
The Workbench allows you to specify different text colors for syntactic elements in the Source Code Editor, as well as select a background color for its window.

By default, as you type text in the Source Code Editor, the system monitors each keystroke as it is entered, recognizing syntactic elements and color-coding them according to their category. (It also appropriately color-codes pasted or imported text.)

To customize these colors, choose the File Setup Colors menu command. Then select one of the following commands from the pop-up menu:

| Command | Description |
|---|---|
| Background | The background color for the Source Code Editor window. |
| Keywords | The color for keyword text. |
| Comments | The color for commentary text. |
| Text | The color for functions, variables, fields, etc. |
| Constants | The color for all constants. |

In each case, a standard Color dialog box with the appropriate title appears. For example:



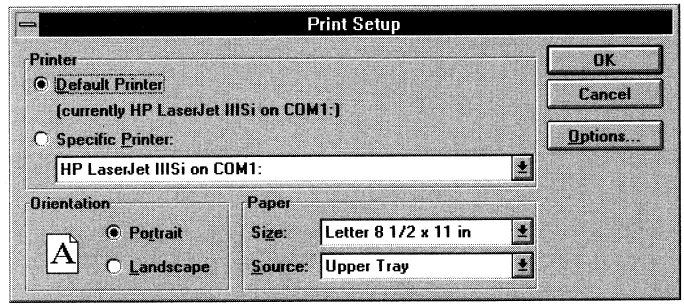
## Setting Up Your Printer

By default, the Workbench uses the printer currently selected as the default in your Windows setup.

To select a different printer (other than the current default) and/or to modify printer settings:

1. Choose the Print Setup command from the File menu.

   You are presented with a standard Print Setup dialog box, indicating the current default printer:




2. Choose a different printer and/or select additional printer options, such as orientation, paper format, and print quality.
3. Choose OK.

**Note:** For information about the various settings, refer to your Microsoft Windows documentation.

## Setting Linker Options

The Workbench's linker allows you to create and run CA-Clipper DOS-applications in protected mode. It eliminates the DOS 640 K barrier, and allows your CA-Clipper applications to access extended memory directly without requiring swapping of any kind.

Additionally, overlaying of code occurs automatically and transparently when your protected-mode CA-Clipper application is running—all you have to do is specify all your .OBJs and .LIBs, and then build!

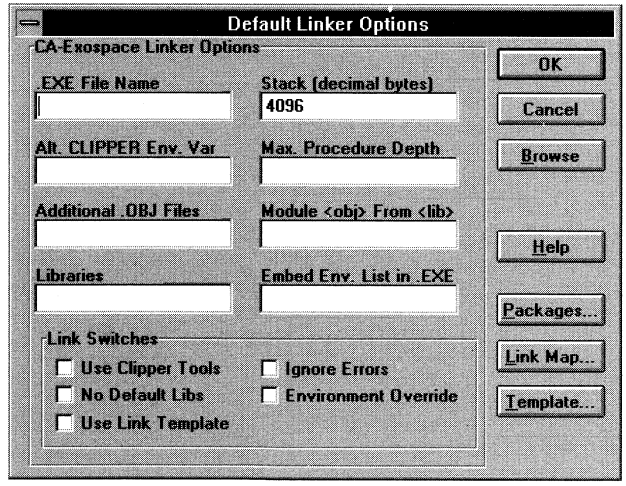
Lastly, as the Workbench is compatible with most third-party libraries for CA-Clipper, just select any of the built-in linker packages that are provided when you define your other default linker options.



### Default Linker Options

To set system-wide, default linker options, choose the Default Linker Options command from the File Setup menu.

The Default Linker Options dialog box appears:



Typically, you will want to specify the most commonly used settings as permanent system defaults and then, if necessary, override some of the system-wide settings on an application-by-application basis. (The various options are described in the Linker Field Options section later in this chapter.)

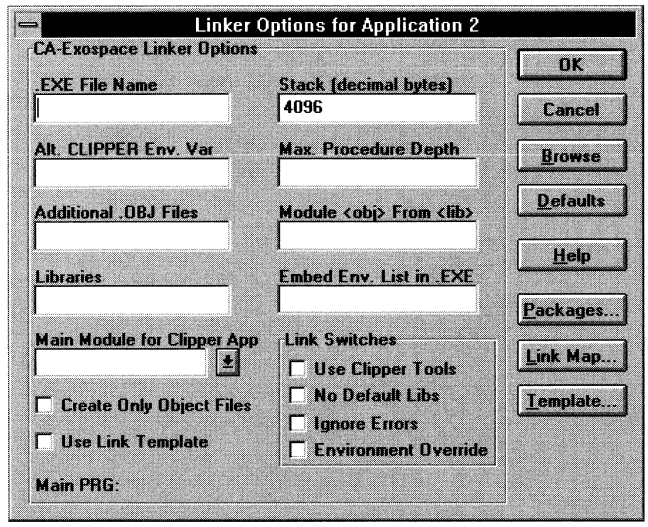
**Note:** Default linker settings are in effect for *all* applications, unless you override them by using the application-level linker options.

## Application-Specific Linker Options

As mentioned above, you can override some of the system-wide linker settings at the application level.

To set application-specific linker options, choose the Linker Options toolbar button (or the Linker Options command from the Application menu).

The Linker Options dialog box appears:



Note that the Linker Options dialog box differs slightly from the Default Linker Options dialog box. Besides different text for the title bar, the Linker Options dialog box has a Main Module for Clipper App. drop-down list box and a Create Only Object Files check box. Additionally, the application-specific version has a *Defaults* button. Use this button at any time to *reset* the linker options for the selected application to the system-wide, default linker options.



## Linker Field Options

.EXE File Name

Specify the name of the executable file to be generated by the linker. The file name can optionally include a directory path. The file extension is optional; if omitted, it defaults to .EXE.

The default is the name of the application's main .PRG module with the .EXE extension.

> **Tip:** Use the Browse push button to access a standard Browse dialog box as an aid in selecting existing files and libraries. This button is active only for the following edit controls: Executable File Name, Additional Object files, and Libraries. See Browsing Directories for more information.

Altered Clipper Env. Var.

Optionally, specify a different name for the environment variable from which the Workbench obtains option settings.

The default is EXOSPACE ENVIRONMENT CLIPPER.

Additional .OBJ Files

Specify the names of one or more object files (separated by commas) to be linked to the .EXE file. The default object file extension .OBJ is optional. File names can optionally include a directory path.

**Note:** If you intend to use the Workbench's debugger to debug your application, the debugger library, CLD.LIB, *must* be included here in the Additional Object Files edit control.

**Tip:** Use the Browse push button to access a standard Browse dialog box as an aid in selecting existing files and libraries. This button is active only for the following edit controls: Executable File Name, Additional Object files, and Libraries. See Browsing Directories for more information.

Libraries

Specify the names of one or more library files to be searched for object files required during linking. The default library file extension .LIB is optional. File names can optionally include a directory path.

**Tip:** Use the Browse push button to access a standard Browse dialog box as an aid in selecting existing files and libraries. This button is active only for the following edit controls: Executable File Name, Additional Object files, and Libraries. See Browsing Directories for more information.

Main Module for Clipper App.

Specify the name of the main module for your Workbench application. The default is MAIN.PRG.

**Note:** This field is available only at the application-specific level.

Create Only Object Files

If checked, instructs the linker to sort linked files by extension and to construct only .OBJ files.

**Note:** This field is available only at the application-specific level.

Use Link Template

If checked, allows you to specify user-defined linker options or provide necessary information to invoke a third-party linker instead of the default linker, CA-Clipper/Exospace.

Choose the Template push button to access the Application Linker Template dialog box and then define a linker template. (See Linker Template Options for more detailed information about linker templates.)

**Note:** This field must be checked in order for user-specific linker options to take effect. After a template is specified, it becomes the default option whenever an application-wide link is executed.

Stack

Specify the size of the application stack in decimal bytes. The default is 4096.

Maximum Procedure Depth

Specify the maximum number of procedures allowed to be nested in a procedure call stack. The default is 30.

Module <obj> From <lib>

Optionally, enter the name of a module to be linked from an *explicitly* specified library. This option may override the default link sequence.

Embed Environ. List in .EXE

Optionally, specify that the default option settings be "burned" into the .EXE file being linked.

Use Clipper Tools

If checked, the CA-Clipper Tools protected mode library, CTP.LIB, will automatically be included in the link script. However, the CA-Clipper object files, CTUSP.OBJ and CTINTP.OBJ will not be included. If you wish to use these object files, you must be aware that conflicts will arise between the CA-Clipper Tools object files and the Light Lib Graphics library. You may not use both the CA-Clipper Tools object files and the Light Lib Graphics library at the same time. If you need to use the CA-Clipper Tools object files, select the "No Default Libs" check box to remove the Light Lib Graphics Library from the link script and then add the appropriate libraries (such as CLIPPER.LIB, TERMINAL.LIB, EXTEND.LIB, and DBFNTX.LIB) to the "Libraries" edit box. Note that most of the functionality included in CTUSP.OBJ and CTINTP.OBJ can be duplicated by using functions from the Light Lib Graphics library.

No Default Libs

By default, the Workbench searches the libraries specified as defaults in the object modules being linked. Choose this option if you do not want the linker to search the default libraries.

Ignore Errors

By default, the Workbench will not create an .EXE if errors, such as duplicate symbols, occur. Choose this option to ignore linking errors and to create an .EXE file anyway.

Environment Override

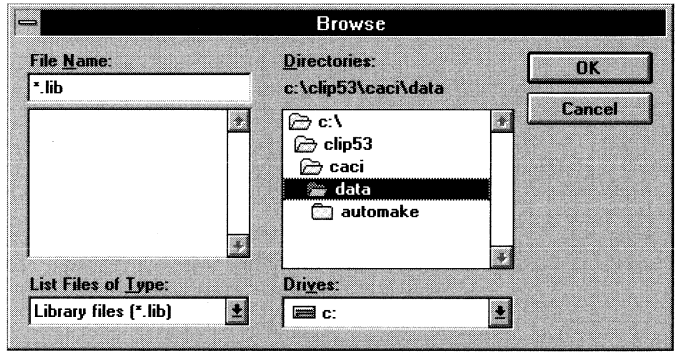
If checked, allows environment settings to override any defaults burned into the .EXE file.

## Browsing Directories

To browse your system's directories in order to select the appropriate files and libraries for the various Linker Field Options:

1. Choose the Browse push button in the Default Linker Options or Linker Options dialog box.

   A standard Browse dialog box appears:




   **Note:** The Browse push button is active only for the following options: Executable File Name, Additional Object Files, and Libraries.

2. Enter or select the name of the desired file.

   The list box displays files with the extension listed in the List Files of Type box.

3. Optionally select another directory or drive, and then select the name of the desired file.

4. Choose OK.

   You are returned to the Linker Options dialog box, and the file name is displayed in the appropriate edit control.

**Tip:** To display a list of files that matches a certain criterion, type standard DOS wildcard characters in this combo box. For example, enter **new*.lib** to display a list of files whose names begin with the letters "new" and have the file extension LIB.



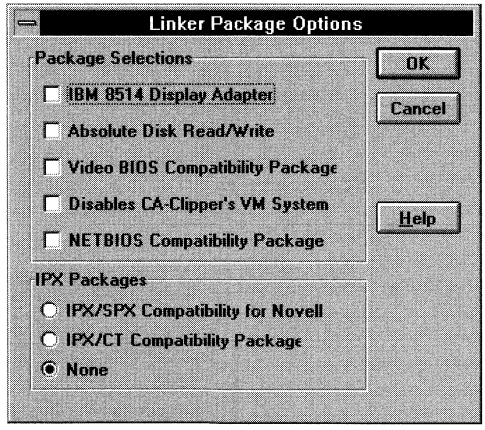
## Selecting Linker Packages

The Workbench's linker provides mouse driver protected-mode compatibility for any program or third-party library that utilizes a mouse. It also provides a selection of optional built-in packages that provide protected-mode support for other low-level operations, such as direct screen I/O and network calls.

To select one or more of these built-in linker packages:

1. Choose the Packages push button in the Default Linker Options or Linker Options dialog box.

   The Linker Package Options dialog box appears:




2. Select one or more linker packages, which are described below.
3. Choose OK.

***Important!*** *You never need to use these packages in an application consisting purely of CA-Clipper code that does not use any third-party libraries. However, some third-party libraries that perform low-level operations require one or more of these packages to be included. If you are using a third-party library that does not appear to operate correctly with the CA-Clipper Workbench component, consult the appropriate vendor to determine whether it requires any packages to be included when linking with the Workbench.*

IBM 8514 Display Adapter

If checked, provides IBM 8514 display adapter compatibility, and allows direct calls from protected mode with no mode switching.

Absolute Disk Read/Write

If checked, provides absolute disk read/write compatibility and services INT 25h and 26h.

Video BIOS Compatibility Package

If checked, provides video BIOS compatibility and services INT 10h.

Disables CA-Clipper's VM System

If checked, disables the CA-Clipper 5.3 VM system. The resulting application will only run on systems that have sufficient RAM to contain the application's entire .EXE file and all its data.

NETBIOS Compatibility Package

If checked, provides NETBIOS compatibility and services INT 5Ch.

IPX/SPX Compatibility for Novell

If checked, provides IPX/SPX compatibility for Novell Networks and services INT 7Ah.

IPXCT Compatibility Package

If checked, provides IPX compatibility specifically for CA-Clipper Tools.

None

If checked, indicates that neither IPX package is installed. This is the default setting.

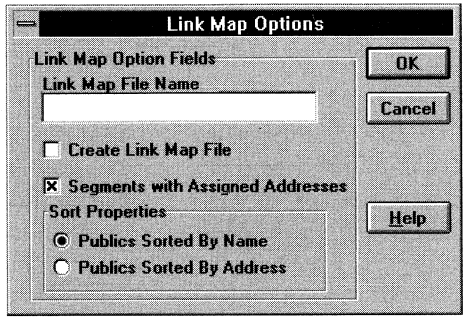
***Important!*** *For IPX compatibility, you must choose either IPX/SPX Compatibility for Novell or the IPXCT Compatibility Package for CA-Clipper Tools, but not both.*

## Link Map Options

To generate a link map:

1. Choose the Link Map push button in the Default Linker Options or Linker Options dialog box.

   The Link Map Options dialog box appears:




2. Specify the following link map option fields described below.
3. Choose OK.

**Note:** The application-specific version, in addition to different title text, has a *Defaults* button. Use this button at any time to *reset* the link map options for the selected application to the system-wide, default link map options.

The choices listed below allow you to declare context-specific link map field options:

Link Map File Name — Specify the file name of the map file being used or created. The default is the name of the .EXE file with the extension .MAP.

Create Link Map File — If checked, generates a link map file.

Segments with Assigned Addresses — If checked, reports segments with assigned addresses. This is the default setting.

Sort Properties

The Publics Sorted by Name and Publics Sorted by Address radio buttons allow you to choose how public symbols are to be sorted. The default setting is Publics Sorted by Name.

**Note:** For more detailed information about any of the link map options or all other Workbench linker options, refer to the *Programming and Utilities Guide.*

## Linker Template Options

A linker template provides the CA-Clipper Workbench with the commands necessary to execute user-defined file linking options and, additionally, provides information necessary to invoke a utility other than CA-Clipper/Exospace, the default linker.

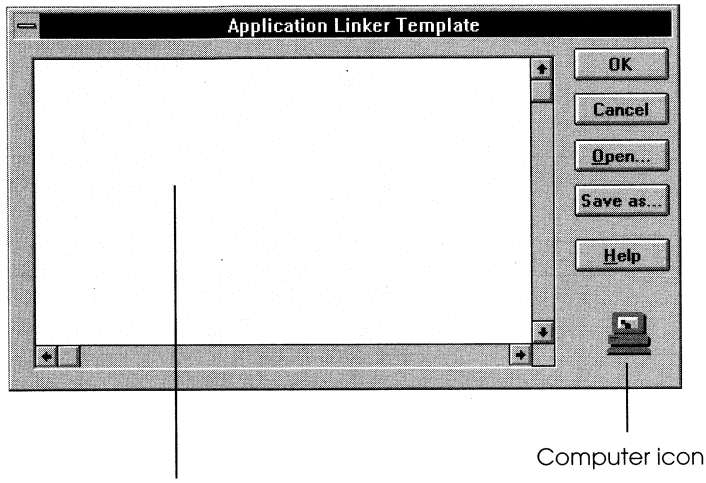
The link template is used along with application and module information to construct a link batch file and a link script file. Application and module data are available via symbols which may be used in the link template. Symbols are replaced by actual application or module data during the generation of the link batch and link script files.

The linker template provides editing options for you to alter sample template ( .TPL) files. You may also generate your own link batch files using Linker Template specific command options (see the "Using Linker Template Files" appendix for more information).

To define a linker template:

1. Choose the Template push button in the Default Linker Options or Linker Options dialog box.

   The Linker Template dialog box appears:




   **Note:** The only differences between the default and application-specific versions of this dialog box are the title bars and the computer icon, which appear in the bottom right-hand corner of the application-specific version only.

2. Define your linker template by entering link batch file command instructions for executing specific file linking options. For example, to invoke the Microsoft Librarian using a script file generated by the same template file, you would enter the following code:

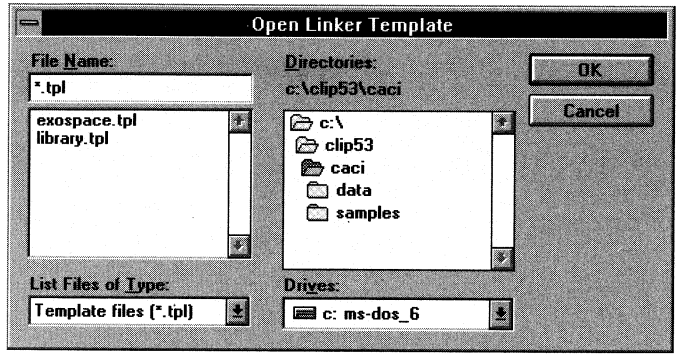
```
#batch
LIB $(__OutFile) @$(__ScriptFile); >> $(__LogFile)
#endbatch
```

   See the "Using Linker Template Files" appendix for detailed information about generating context-specific coding for your linker template.

3. Choose the Save As push button to access a standard Save As dialog box for naming and saving your linker template as a template (.TPL) file.

4. Alternatively, if you wish to edit an existing link batch file, choose the Open push button.

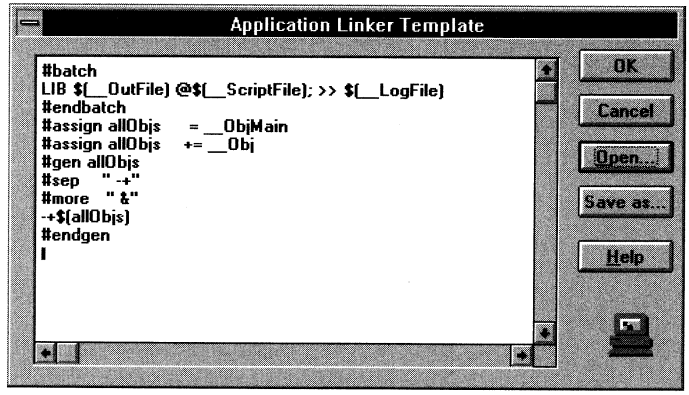
   The Open Linker Template dialog box appears:




   **Note:** Sample template files, EXOSPACE.TPL and LIBRARY.TPL, already exist in the CLIP53\CACI directory and may be edited. Additionally, files from any other directory may be referenced and edited.

5. Highlight a template file in the File Name list box and choose OK.

   For example, if you choose LIBRARY.TPL, the following code appears in the Linker Template list box:




6. Make any necessary edits and choose OK.

   You will be returned to the appropriate Linker Options dialog box.

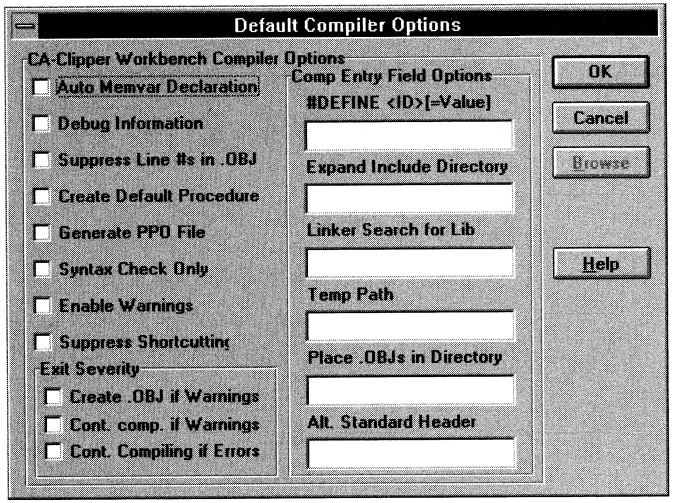
## Setting Compiler Options

The Workbench provides a set of default compiler options, which can be changed on a system-wide basis, thereby affecting all *new* applications from that point forward, or changed at the application-level, affecting only the *current* application.

### Default Compiler Settings

To set system-wide, default compiler options, choose the Default Compiler Options command from the File Setup menu.

The Default Compiler Options dialog box appears:



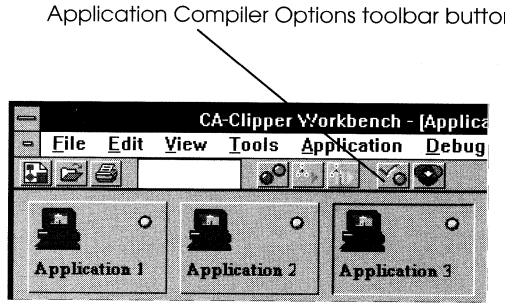

Typically, you will want to specify the most commonly used settings as permanent system defaults and then, if necessary, override some of the system-wide settings on an application-by-application basis. (The various options are described later in this chapter.)

**Note:** Default compiler settings are in effect for *all* applications, unless you override them by using the application-level compiler options.

## Application-Specific Compiler Settings

As mentioned earlier, you can override any one or all of the system-wide compiler settings at the application level.

To set application-specific compiler options, choose the Application Compiler Options toolbar button (or the Compiler Options command from the Application menu):




The Compiler Options dialog box is identical to the Default Compiler Options dialog box except for the text in the title bar and the fact that the application-specific version has a *Defaults* button. Use this button at any time to *reset* the compiler options for the selected application to the system-wide, default compiler options.

**Note:** You can also set various compiler options at the module level. For more information, see Setting Compiler Options at the Module Level in the next chapter.

## Workbench Compiler Options

| | |
|---|---|
| Auto Memvar Declaration | The MEMVAR declaration statement causes the compiler to resolve references to variables specified without an explicit alias by implicitly assuming the memory variable alias (MEMVAR->). Therefore, if this option is checked, any variable included in a PRIVATE, PUBLIC, or PARAMETERS statement is declared as part of a MEMVAR statement. |
| Debug Information | If checked, debugging information is included in the object file. This is the default setting. |
| Suppress Lines Numbers in .OBJ | Excludes the program source code line numbers from the object file, if checked. |
| Create Default Procedure | Automatically defines a procedure with the same name as the program (.PRG) file, if checked. |
| Generate PPO File | If checked, the Workbench preprocesses the .PRG file and copies the result to an output file with a .PPO extension. |
| Syntax Check Only | If checked, verifies the syntax of the current .PRG file but does not create an object (.OBJ) file. |
| Enable Warnings | Generates warning messages for undeclared or unaliased variable references. |
| Suppress Shortcutting | If checked, suppresses shortcutting optimizations on the .AND. and .OR. logical operators. (This option is provided as an aid to isolating code that depends on the behavior of older versions of CA-Clipper.) |

### Exit Severity Options

| | |
|---|---|
| Create .OBJ If Warnings | If checked, forces the creation of an object file (.OBJ) despite warnings encountered during compilation. |
| Cont. Comp. If Warnings | If checked, continues compiling the current .PRG file despite warnings encountered. |
| Cont. Compiling If Errors | If checked, continues compiling the current .PRG file despite errors encountered. |

### Comp. Entry Field Options

| | |
|---|---|
| #DEFINE <ID>(=Value) | Optionally define a set of identifiers to the preprocessor with or without corresponding values and separated by commas or semicolons. If *<text>* is not specified, *<identifier>* is given an empty value. |
| Expand Include Directory | Enter one or more header file search directories (separated by commas or semicolons) to be added to the front of the INCLUDE path list. |

**Tip:** Like the Linker Options dialog box, both versions of the Compiler Options dialog box have a Browse push button that allows you to access a standard Browse dialog box as an aid in selecting existing files and libraries. This button is active only for the following edit controls: Expand Include Directory, Linker search for Lib, and Temp Path.

Linker Search for Lib

Optionally specify one or more library search requests to be embedded in the .OBJ file. The defaults are CLIPPER.LIB, EXTEND.LIB, DBFNTX.LIB, and TERMINAL.LIB.

***Warning!*** *If you specify an additional library, it must be* ***added*** *to the list of default library search requests. In other words, you must enter the name of the additional library as well as the names of the default libraries in the Linker Search for Lib edit control. For example:* ***NEW.LIB, CLIPPER.LIB, EXTEND.LIB, DBFNTX.LIB, TERMINAL.LIB****. Otherwise, the default libraries are completely* ***superseded*** *by the contents of this edit control.*

Temp Path

Specify a different directory for temporary files generated during compilation.

Place .OBJs in Directory

Specify the path of the output object file.

Alt. Standard Header

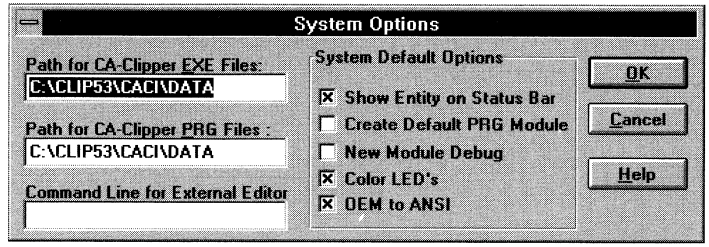
Specify an alternate standard header file to be preprocessed instead of the default STD.CH file.

**Note:** For more detailed information about any of the Workbench compiler options, refer to the *Programming and Utilities Guide*.

## Setting System Options

The Workbench provides a set of default system options, such as defining default paths and creating default modules. You can also select or deselect other system options, such as debugging new modules and showing prototypes on the status bar.

To set or override the default system options, select the System Options command from the File Setup menu and customize the System Options dialog box as desired:




### Default Path Options

Path for CA-Clipper EXE Files
: Specify the default directory in which to place generated executable (.EXE) files.

Path for CA-Clipper PRG Files
: Specify the default directory for program (.PRG) files.

Command Line for External Editor
: Specify the path for an external program editor. The default is the Workbench's Source Code Editor.

## Miscellaneous System Options

Show Entity on Status Bar

By default, the CA-Clipper Workbench automatically displays prototypes for entities (like functions) in the status bar. You can suppress this display by deselecting this option.

Create Default PRG Module

By default, the Workbench does *not* create and load a new empty module in addition to the Binary Objects module every time you create a new application. Select this option if you want the system to create and load a default module automatically when creating a new application.

New Module Debug

By default, the system does *not* automatically turn debugging on for all new modules. Select this option if you do want the system to automatically debug any new modules.

Color LEDs

By default, the Workbench automatically displays LED-style indicators for compilation status. You can override this option by deselecting it. Compilation status will then be indicated appropriately by a check mark, an "X", a question mark, or an exclamation mark. The following table describes the status for each indicator:

| Indicator | Status |
|---|---|
| Green LED or "√" | Application has compiled successfully with no errors, contains an executable file, and is ready to run providing that the .EXE has not been deleted from DOS. |
| Blue LED or "!" | Application has compiled successfully, but with warnings. |
| Red LED or "X" | Application has compiled unsuccessfully with errors. |
| Yellow LED or "?" | Application needs to be compiled as its state is undetermined by Workbench. |

OEM to ANSI

By default, when you import external modules from within the Workbench, an automatic translation between the OEM and ANSI character sets is performed. Deselect this option to prevent this translation (for example, to preserve words with special, non-English language characters, such as ä and å).

## Saving the Current Desktop

The Save Desktop command on the Setup menu allows you to save the current configuration of all open windows and/or editors. The next time you start the CA-Clipper Workbench, all windows/editors in the current configuration will be opened and arranged as they were when you selected the Save Desktop command.

# Chapter 3
# Browsing Applications, Modules, and Entities

## In This Chapter

This chapter explains how to use the Application, Module, and Entity Browsers.

For information about the Error Browser, see Using the Error Browser to Resolve Compiler Errors in the "Debugging Your Applications" chapter.

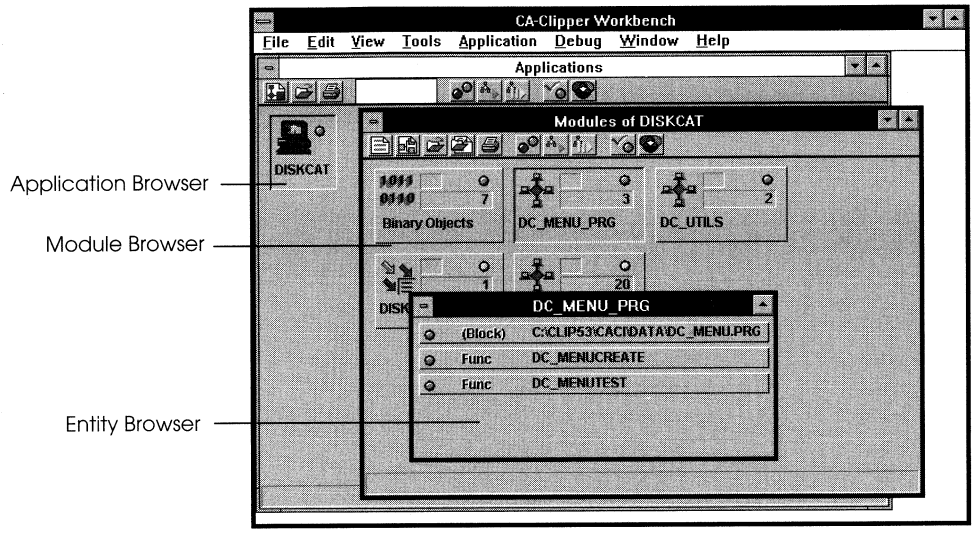


## The Browser Hierarchy

In the CA-Clipper Workbench, *applications* consist of *modules*, which consist of *entities*. The primary browsers, therefore, follow this same top-down hierarchy. When you start the Workbench, the *Application Browser* is automatically loaded. Double-clicking on an application displays the modules defined for that application in a *Module Browser*.

Similarly, double-clicking on a module displays the entities defined for that module in an *Entity Browser*. Alternatively, if you select the Entity Browser command from the Tools Menu, the Entity Browser displays all of the entities defined for the *application*, not just the specified module.

The following example from the tutorial in the *Getting Started* guide reflects this hierarchy:

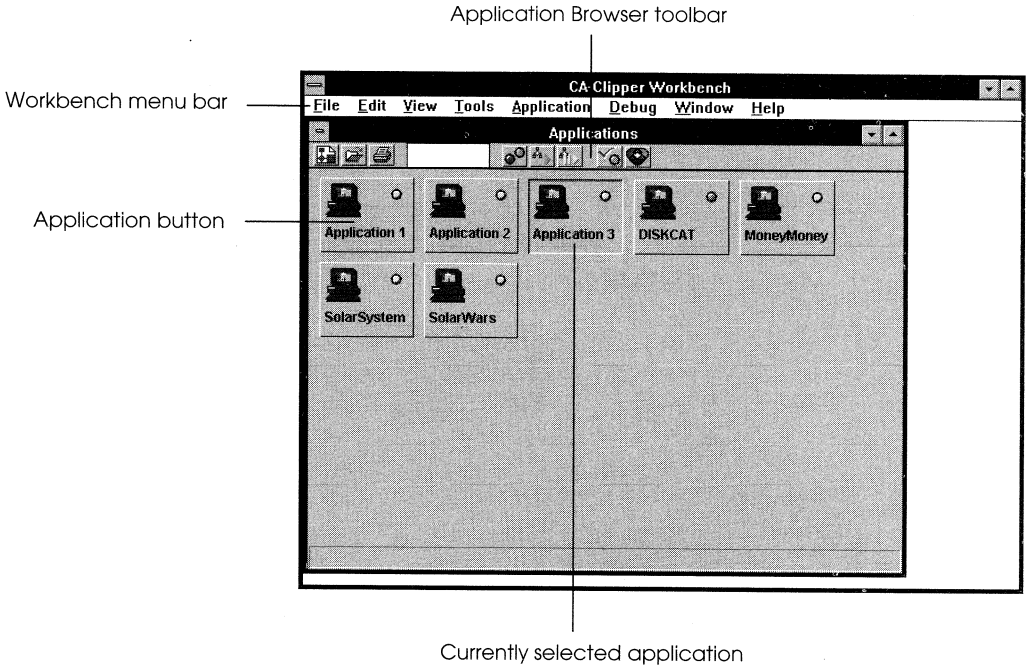



Note that you can customize the display while in either the Application or Entity Browser by using the Name filter on their respective toolbars. (See Customizing the Application Browser later in this chapter for details.)

**Tip:** While there is only one Application Browser window, you can open multiple Module and Entity Browsers in the desktop to simultaneously view the different modules and entities of one or more applications.

# The Application Browser

When you start the Workbench, the Application Browser is automatically loaded. It displays, as a button, each of the applications currently stored in the Workbench's repository:




## Working in the Application Browser

When you are in the Application Browser, you can:

- Create, open, rename, delete, import, export, print, build, and debug applications
- Set linker and compiler options at the application level
- Access other browsers and editors by double-clicking on an application

## Application Buttons

The Application Browser displays applications as buttons arranged in alphabetical order by application name.

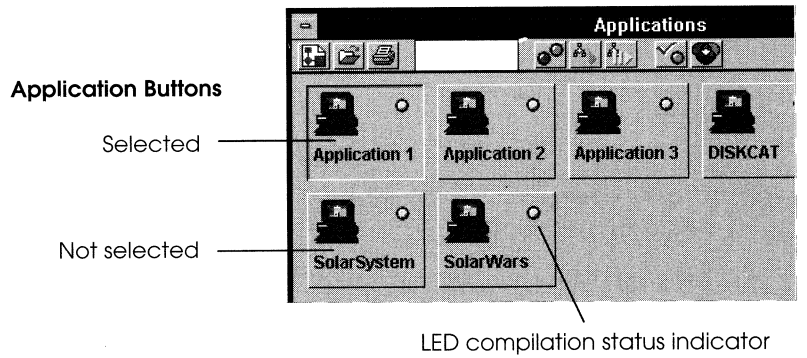
LED Icons

By default, compilation status is indicated by LED-style icons on the application buttons. *Green* indicates that the application has been compiled successfully, contains an executable file, and is ready to run provided that the .EXE has not been deleted from DOS. *Red* indicates compilation errors or warnings. *Blue* indicates that the compile of your application (or module) has been successful, but there are compiler warnings. Lastly, *yellow* indicates that the application is in an indeterminate state: either it is new and needs to be compiled, or has been modified and needs to be recompiled.

If, however, you have chosen to override the default LED icons when setting your system options, then compilation status will be indicated by a check mark, an "X," an exclamation mark, or a question mark, respectively. (See Setting System Options in the previous chapter, "Working in the Workbench.")

Selected/Deselected

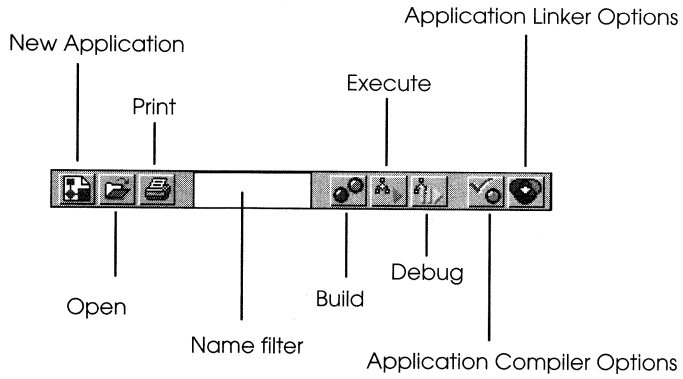
One of the application buttons is always selected in the Application Browser:




**Note:** All commands that you perform apply to the selected application. To select a different application, simply click on its button.

## The Toolbar

The Application Browser toolbar contains the following buttons:




The New Application, Open, Print, and Linker Options buttons and the Name filter are described later in this chapter. See "Working in the Workbench" for information about Build, Execute, and Application Compiler Options. See "Debugging Your Applications" for information about the Debug option.

**Tip:** For a quick description of these toolbar buttons, look at the status bar as the mouse pointer passes over the buttons. Also see your online Help reference.

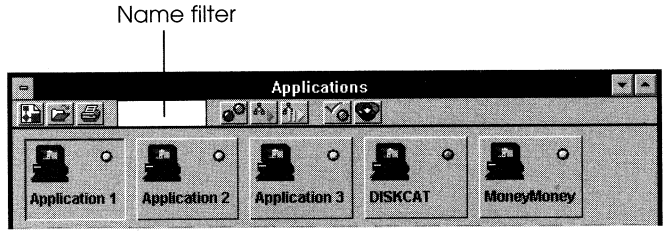
## Customizing the Application Browser

By default, the Application Browser initially displays all applications. You can, however, customize the Application Browser's initial display by restricting the display to applications with a certain name.

### Using the Name Filter

For example, if you want to view only applications whose names begin with the word "solar":

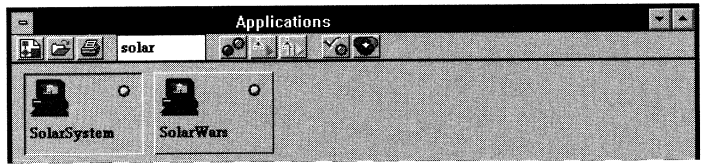
1. Enter **solar** in the Name filter on the toolbar:




**Note:** No wildcards are allowed. However, you may enter just a portion of the application name.

2. Press Enter.

The Application Browser changes, displaying the following applications:




### Restoring the Display

To restore the Application Browser window to its original display:

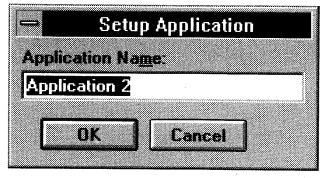
1. Click in the Name filter at the end of the text it contains (for example, after the "r" in "solar").
2. Press the Backspace key until all characters have been erased.
3. Press Enter.

## Creating a New Application

The process of creating a new application in the CA-Clipper Workbench consists of the following basic steps:

1. From within the Application Browser, click on the New Application button.

   The Setup Application dialog box appears:

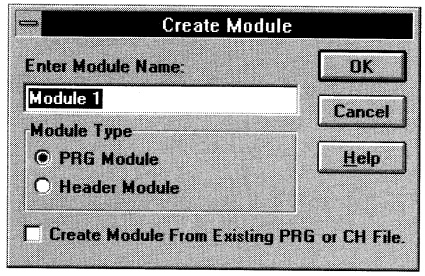



2. Enter your application's name in the Application Name edit control.

   Application names can be up to 30 characters long, including spaces and special characters. The default is Application [*n*], where *n* is an integer.

3. Choose OK.

   If the Create Default PRG Module option was checked previously in the System Options dialog box when setting up your system options, the Create Module dialog box appears:



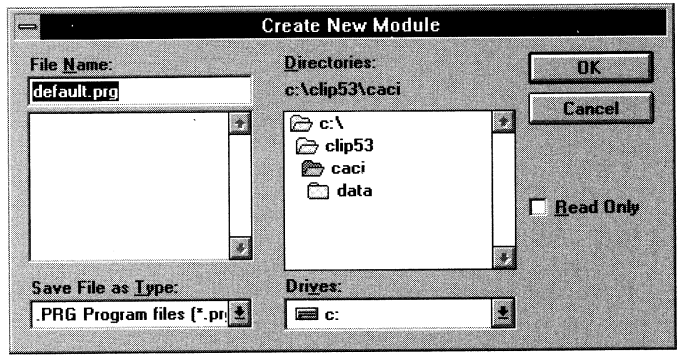

4. Enter a name in the Enter Module Name edit control.

   This is the module name for the first, non-binary objects module in your application. Module names can be up to 30 characters long, including spaces and special characters. The default is Module [*n*], where *n* is an integer.

5. Specify the module type: either a program source file module or a header file module. The default is PRG Module.

6. Optionally, specify whether the new module is to be created from an existing .PRG or .CH file by checking the corresponding check box.

   If you are using an existing file, a standard Open dialog box appears. If you are creating a new module "from scratch," the Create New Module dialog box appears:



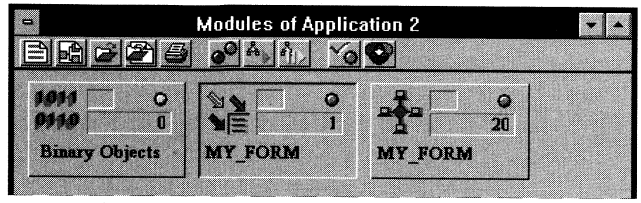

7. Specify the file name and path of the file being created.

   **Note:** If the new module is being created from an existing file, the external file name will *overwrite* the module name. Furthermore, if you have both an external .PRG file and a header file with the same name (for example, MY_FORM.PRG and MY_FORM.CH), the two new modules that are created will have the same name (i.e., MY_FORM) but different graphics on their corresponding module buttons in the Module Browser.

8. Optionally select the Read Only check box.

9. Choose OK.

   The Module Browser for the new application is activated:




   **Note:** The new application automatically contains a special module, called "Binary Objects," which *cannot* be deleted. It contains the source binary information for menus, forms, data servers, and field specs. At this time, it contains no entities. For more information about this special module and the other module types, see Creating Modules later in this chapter.

10. Define additional components for your application. These can include program source file modules, header file modules, and objects created by the Menu, Form, DB Server, and FieldSpec Editors.

11. Optionally edit the generated .PRG and .CH files using the Source Code Editor while within the appropriate modules. (See the "Using the Source Code Editor" chapter later in this guide.)

12. Optionally, override the system-wide, default linker and compiler options for the new application by choosing the Application Compiler Options and Linker toolbar buttons, respectively. (See Setting Compiler Options and Setting Linker Options in the "Working in the Workbench" chapter.)

13. Choose the Build toolbar button to build the application. (See Building an Application in the "Working in the Workbench" chapter.)

14. Execute the application using the Execute toolbar button. (See Executing an Application in the "Working in the Workbench" chapter.)

15. Debug the application using the Debug menu options. (See "Debugging Your Applications" later in this guide.)

## Opening Applications

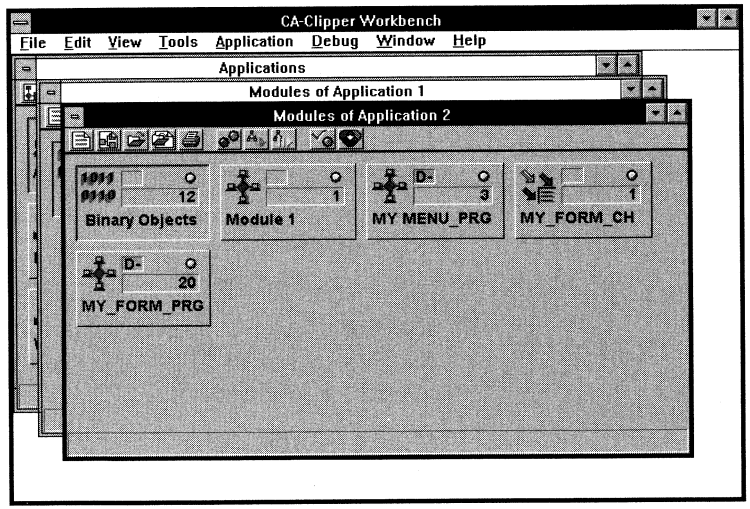
In the Workbench, *opening* an application simply means to display its modules in a Module Browser. To open an existing application, simply double-click on its button in the Application Browser. Alternatively, when the desired button is selected, click on the Open Application toolbar button.

Another method is to right-click on an application button to open a local pop-up menu:

| |
|---|
| **Start application viewer**<br>**Delete application** |
| **Compiler options**<br>**Linker options** |
| **Build the application**<br>**Force an application build**<br>**Execute the application** |

Then choose the Start Application Viewer command to open the application. In all cases, a Module Browser appears, displaying all the modules in that application.

Note that you can have multiple applications open at the same time. For example, two applications are open below, with Module Browsers for each displaying their respective modules:



See The Module Browser section later in this chapter for detailed information about Module Browsers.

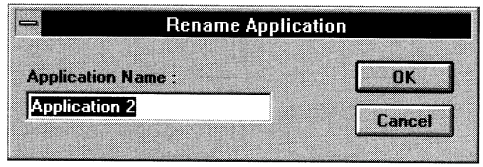
## Renaming and Deleting Applications

The Rename and Delete commands on the File menu allow you to manipulate your applications.

For example, if you want to rename an application:

1. Select the button of the application you want to rename.
2. Choose the Rename Application menu command.

   The Rename Application dialog box appears:




3. Enter the new name in the Application Name edit control (for example, **My Application**).
4. Choose OK.

See your online Help reference for more information about these menu commands.

## Importing and Exporting Applications

Instead of creating a new application, you can import and modify an existing CA-Clipper Workbench application that has been stored as an external application (.CEF) file.

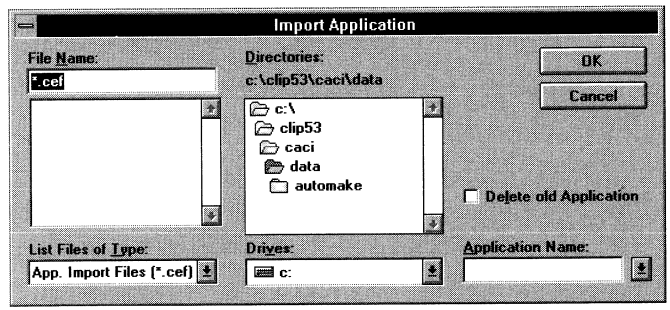
Similarly, while within the Application Browser you can export the current application in its entirety in a single .CEF file. This would be useful if you need, for example, to copy an application to disk so that it may be copied onto another PC.

### Importing a CA-Clipper Application

To import an external application file into the CA-Clipper repository:

1. Switch to or open the Application Browser.
2. Choose the File Import menu command.

   The Import Application dialog box appears.




3. Choose a .CEF file to be imported by highlighting it in the File Name combo box.

   **Note:** To display a list of files that matches a certain criterion, type standard DOS wildcard characters in this combo box. For example, enter **new*.cef** to display a list of files whose names begin with the letters "new" and have the file extension .CEF.

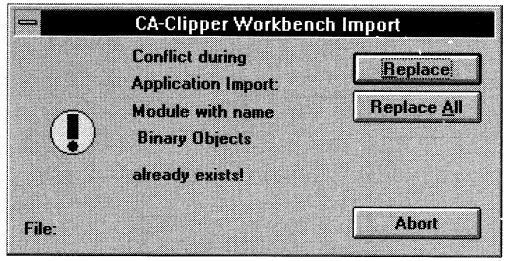
4. Optionally enter a new name of up to 30 characters in the Application Name combo box, or select the name of an existing application.

   By default, an imported application is created under the same name as its .CEF file. This option provides you with the opportunity to assign a longer, more meaningful name.

Additionally, a new name would also allow you to import an application without deleting, or *overwriting* an existing application of the same name.

5. Optionally, select the Delete Old Application check box to delete or overwrite an existing, same-named application.

   Note that if you do not select the Delete Old Application option, you will be prompted regarding conflicts during the import process by the following dialog box:




   Choose Replace to overwrite the repository version of the named entity with the version stored in the export file.

   Choose Replace All to overwrite the repository counterparts of *all* entities stored in the export file without further warning.

   ***Important!*** *The Replace All option should be used with* ***caution****—you should choose this option* ***only if*** *you are absolutely sure that the entities in the export file are more current than the versions in the repository.*

6. Choose OK.

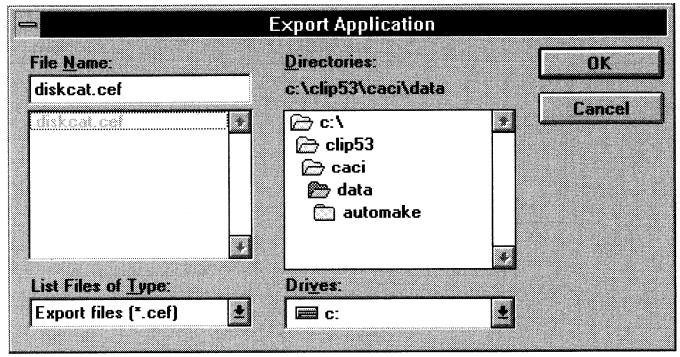
You should then save and compile the new application in one step using the Build toolbar button.

## Exporting an Application

To export a CA-Clipper application as a .CEF file:

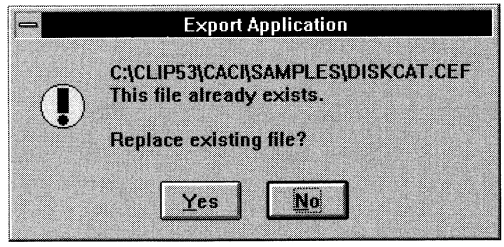
1. Switch to or open the Application Browser.
2. Click on the button representing the application you want to export.
3. Choose the File Export menu command.

   The Export Application dialog box appears.




4. Enter or select the .CEF file to be created.
5. Choose OK.

**Note:** If an application already exists with the same name, you will be prompted to overwrite the existing file during the export process by the following dialog box:

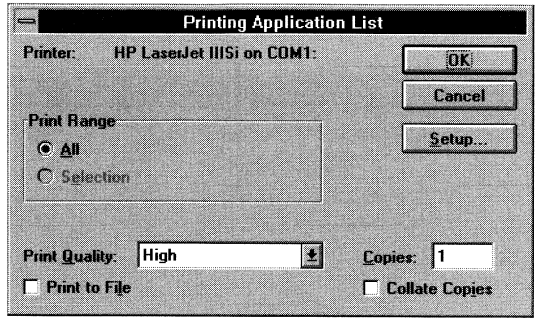


## Printing an Application List

To print a list of all applications stored in the repository:

1. Choose the Print toolbar button.

   The Printing Application List dialog box appears:




2. Specify your printing options.

   **Note:** Click the Setup push button to view/modify the current printer's settings. You are presented with a standard Print Setup dialog box, from which you can choose a different printer and/or select additional printer options, such as orientation and paper format.

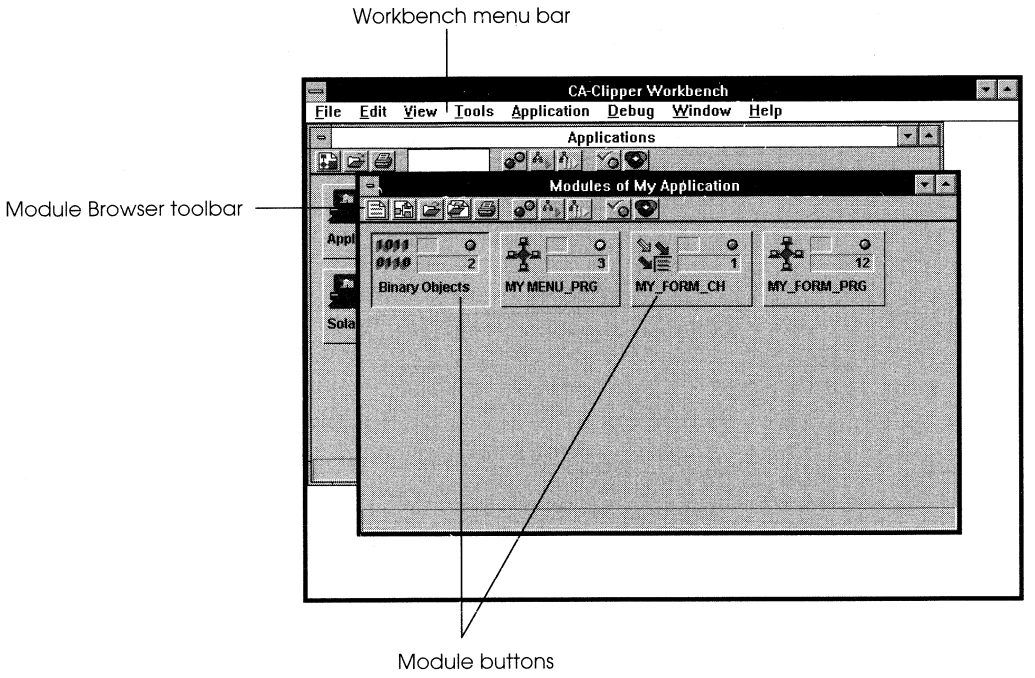
3. Choose OK.

This standard Print dialog box contains the following options:

| | |
|---|---|
| Printer | Displays the target printer. |
| Print Range | This option is permanently set to All. |
| Print Quality | Displays the current DPI setting (see your printer documentation for details on what is supported for this feature). |
| Print To File | If checked, will send output to a file. You will be prompted for a file name. If not checked, output will be sent to the printer. |
| Copies | Enter the number of copies to be printed. The default is 1. |
| Collate Copies | If checked, prints copies organized in order of page numbers. |

# The Module Browser

Applications contain modules; therefore, double-clicking on any application in the Application Browser displays *all* its modules in the Module Browser:




## Working in the Module Browser

When you are in the Module Browser, you can:

- Create, open, rename, delete, print, build, and debug modules
- Specify compiler settings at the module level
- Access other browsers and editors

## Module Buttons

Just as the Application Browser displays applications as buttons, a Module Browser displays modules as buttons that are arranged in alphabetical order. These buttons display a graphic depicting module type, and also indicate the compilation status of each module. In addition, module buttons contain other information, including a debugging indicator and an entity number.

### Graphics

The graphic used in a module button indicates the module *type*:

| Graphic | Represents |
|---|---|
|  | The Binary Objects module. |
|  | A program source file module. |
|  | A header file module. |

See Creating Modules for more information about module types.

### LED Indicators

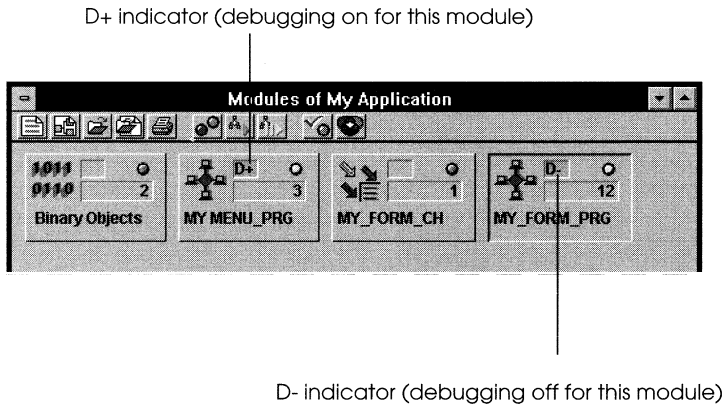
The module's status is indicated by LED indicators. *Green* means the module has been compiled successfully, while *Red* indicates compilation errors or warnings. *Blue* indicates a successful build but with warnings. Lastly, *Yellow* means that the module is in an indeterminate state: either it is new and needs to be compiled, or has been modified and needs to be recompiled.

However, for the Binary Objects module only two colors are available: *Red* means either that the .PRG file attached to a form or menu entity has been changed (via the Source Code Editor), or that the field specs being used to generate the .PRG file have changed since the last time the .PRG was generated. *Yellow* is used in all other cases.

## Debugging Indicators

In every application, each module has a debugging indicator. When you create an application initially, you can turn the debug option on or off. Each module within the application then automatically inherits the application's debug setting. For example, if an application's debug option is on, all modules that are created subsequently will have debugging turned on. Conversely, if an application's debug option is *not* turned on, all modules will have debugging turned off.

However, you may choose to enable or disable debugging on a module-by-module basis. Therefore, the debugging indicators "D+" and "D–" are used to denote that individual modules have been explicitly enabled or disabled, respectively:




See Setting Debugging Options in the "Debugging Your Applications" chapter for more information about the debugging indicators.

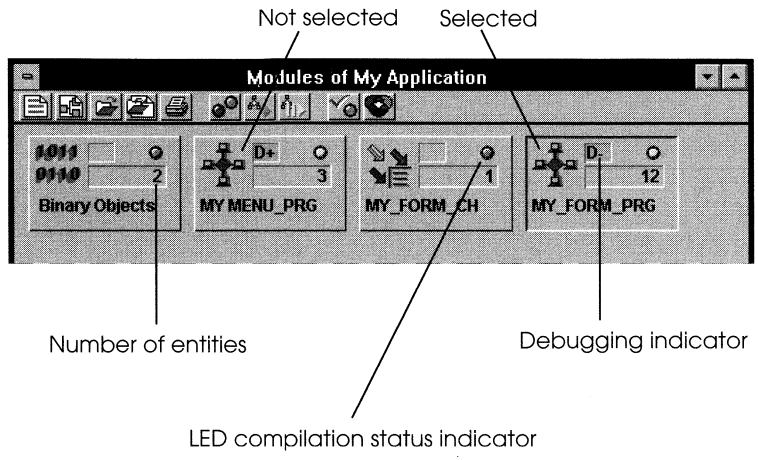
## Entity Numbers

Lastly, a module button also displays the total number of entities contained within the individual module.

Selected/Deselected

One of the module buttons is always selected in the Module Browser:

**Module Buttons**

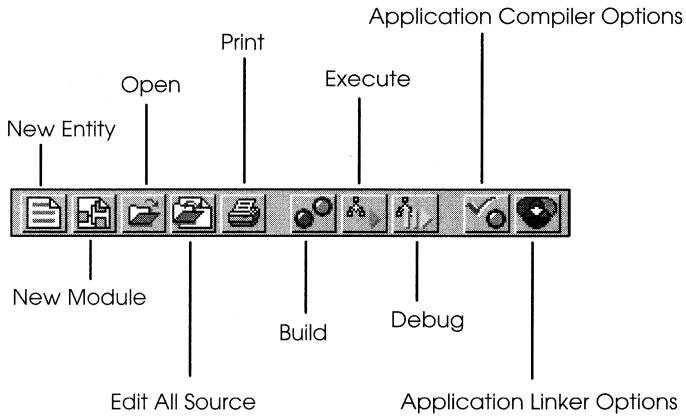



**Note:** All commands that you perform from within the Module Browser apply to the selected module. To select a different module, simply click on its button.

**Tip:** Like any other Workbench window, a Module Browser can be maximized or minimized to an icon. Also, windows may be cascaded or tiled.

## The Toolbar

The Module Browser toolbar contains the following buttons:




The New Entity, New Module, Open, Edit All Source, and Print buttons are described later in this chapter. See "Working in the Workbench" for information about Build, Execute, and Application Compiler Options. See the Linker Options section earlier in this chapter for information about Linker Options. For information about Debug, see "Debugging Your Applications."

**Tip:** For a quick description of these toolbar buttons, look at the status bar as the mouse pointer passes over the buttons. Also see your online Help reference.

## Creating Modules

The Binary Module

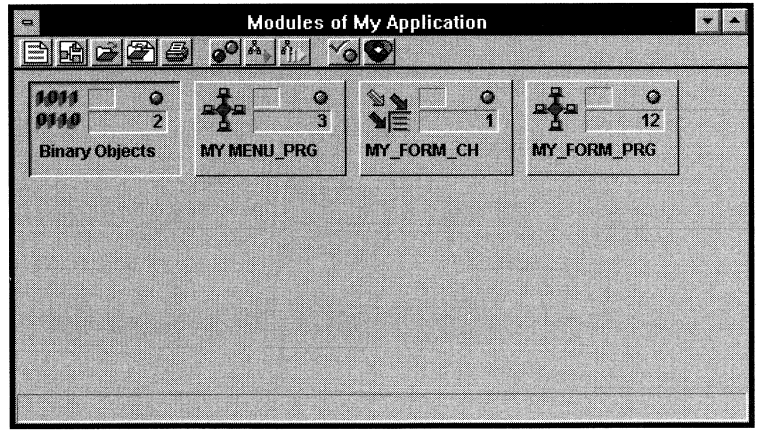
When an application is created, the special Binary Objects module is automatically created. This module will contain the binary definitions of all form, menu, data server, and field spec entities that you create using the Workbench's four editors: the Menu Editor, Form Editor, DB Server Editor, and FieldSpec Editor.

Non-Binary Modules

All other modules are non-binary and represent source files either as individual program modules (usually .PRG files) or as header modules (usually .CH files).

The Menu and Form Editors, in addition to creating entities that are stored in the Binary Objects module, generate .PRG files and their corresponding modules. Also, the Form Editor creates a .CH file and a corresponding module.

For example, if you created a form called My Form and a menu called My Menu, the Module Browser for our example would look like this:

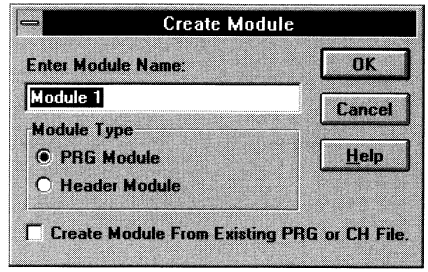


Basic Steps

To create a new module:

1. Click on the New Module toolbar button.

   The Create Module dialog box appears:




2. Type a name in the Enter Module Name edit control.

   Module names can be up to 30 characters long, including spaces and special characters. The default is Module [*n*], where *n* is an integer.

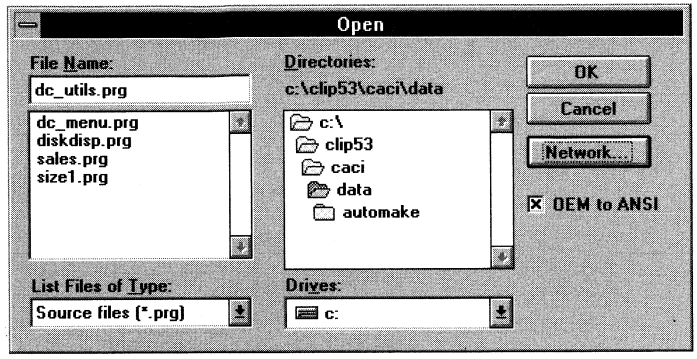
3. Select the module type.

   Choices are: PRG Module and Header Module.

4. Specify whether the module is to be created from an existing program or header file, or if a new module has to be created.

5. Choose OK.

Either the Create New Module or a standard Open dialog box appears:




6. Select or enter the file name attached to the module.
7. Choose OK.

**Tip:** To select multiple modules in a sequence, click on the first file name and then hold down the Shift key while you click on the last file name in the group. To select two or more modules out of sequence, hold down the Ctrl key while you select each file name. This is a standard Windows technique for any Open or Browse-type dialog box.

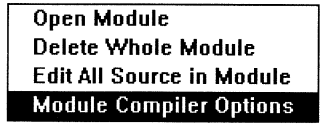
## Setting Compiler Options at the Module Level

In the Workbench, you can set compiler options at the module level, consequently overriding the system settings for an individual module.

To set compiler options for a specified module:

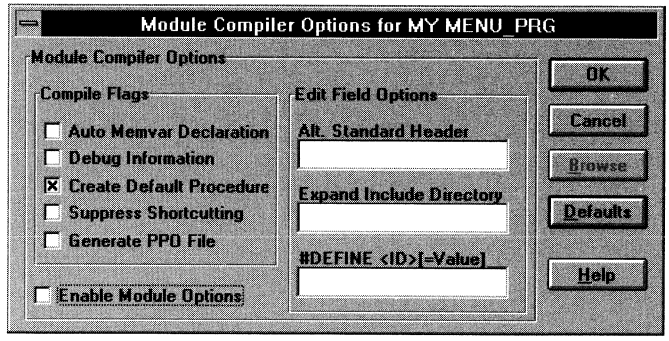
1. Right-click on the desired module button.

   A local pop-up menu appears:




2. Choose the Module Compiler Options command.

   The Module Compiler Options dialog box appears:




3. Optionally set any of the module-level compiler options described below.

4. Press OK.

**Note:** Choose the Defaults push button at any time to *reset* the compiler options for the selected module to the system-wide, default compiler options.

## Module Compiler Options

| | |
|---|---|
| Auto Memvar Declaration | If checked, any variable included in a PRIVATE, PUBLIC, or PARAMETERS statement is declared as part of a MEMVAR statement. |
| Debug Information | If checked, debugging information is included in the object file. This is the default setting. |
| Create Default Procedure | Automatically defines a procedure with the same name as the program (.PRG) file, if checked. |
| Suppress Shortcutting | If checked, suppresses shortcutting optimizations on the .AND. and .OR. logical operators. (This option is provided as an aid to isolating code that depends on the behavior of older versions of CA-Clipper.) |
| Generate PPO File | If checked, CA-Clipper preprocesses the .PRG file and copies the result to an output file with a .PPO extension. |
| Enable Module Options | If checked, the specified module options override the corresponding application compiler options for the current module. |

Alternate Standard Header

Specify an alternate standard header file to be preprocessed instead of the default STD.CH file.

Expand Include Directory

Enter one or more header file search directories (separated by commas or semicolons) to be added to the front of the INCLUDE path list.

**Tip:** Use the Browse push button to access a standard Browse dialog box as an aid in defining search directories. See Browsing Directories in the "Working in the Workbench" chapter for more information.

#DEFINE <ID>(=Value)

Optionally define a set of identifiers to the preprocessor with or without corresponding values and separated by commas or semicolons. If *<text>* is not specified, *<identifier>* is given an empty value.

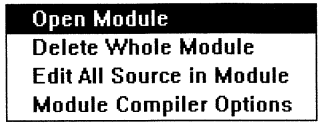
**Note:** For more detailed information about any of the Workbench compiler options, refer to the *Programming and Utilities Guide*.

## Opening a Module

In the Workbench, *opening* a module simply means to display its entities in an Entity Browser.

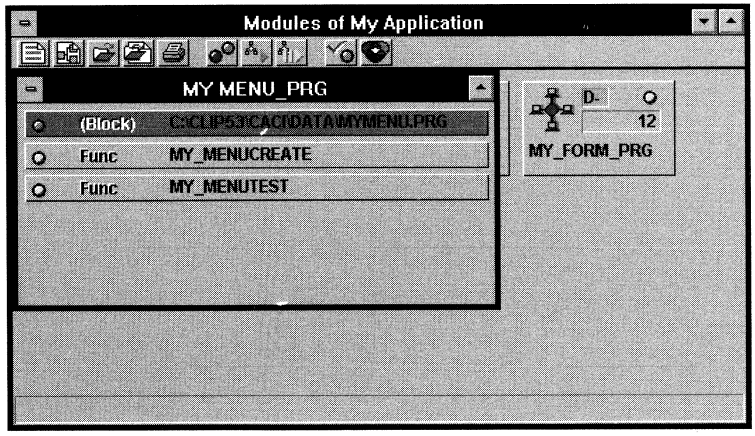
To open an existing module, simply double-click on its module button in the Module Browser or, when the desired module button is selected, click the Open toolbar button.

Another method is to right-click on a module button to open a local pop-up menu:




Then choose the Open Module command.

In all cases, an Entity Browser appears, displaying all the entities in that module. For example:




See The Entity Browser section later in this chapter for more detailed information about Entity Browsers.

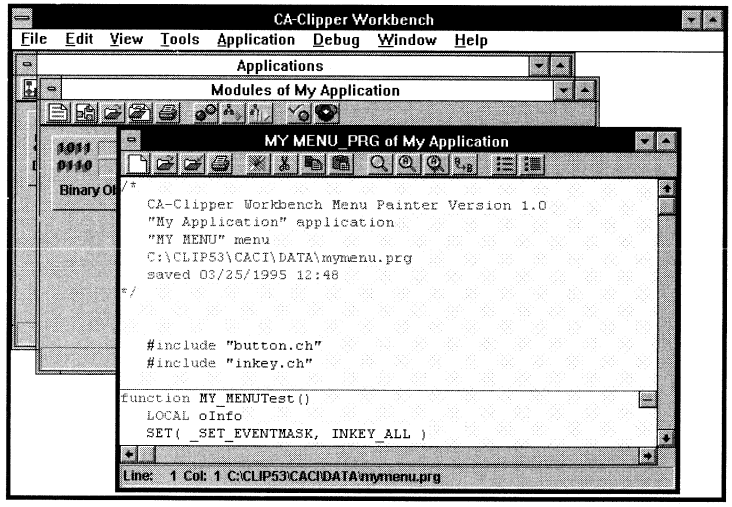
## Editing the Entire Module

Accessing a module's source code can be done either at the module level or the entity level. To view *all* of the module's source code (not just the code for an individual entity within that module), use the Edit All Source toolbar button.

Accessing a module's source code can be done either at the module level or the entity level. To view *all* of the module's source code (not just the code for an individual entity within that module), use the Edit All Source toolbar button.

Alternatively, right-click on the module to open the local pop-up menu shown above. Then choose the Edit All Source in Module command.

In either case, the Source Code Editor appears:




You can now edit the module's source code.

For information on how to access the source code at the entity level, see The Entity Browser section later in this chapter. For more information about editing source code, see the "Using the Source Code Editor" chapter.

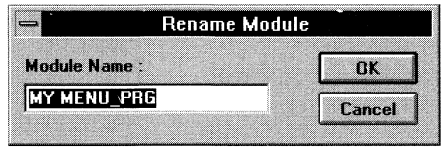
## Renaming and Deleting Modules

The Delete Module and Rename Module commands on the File menu allow you to manipulate modules.

For example, if you want to rename a module:

1. Select the button of the module to be renamed.
2. Select the Rename Module menu command.

   The Rename Module dialog box appears:



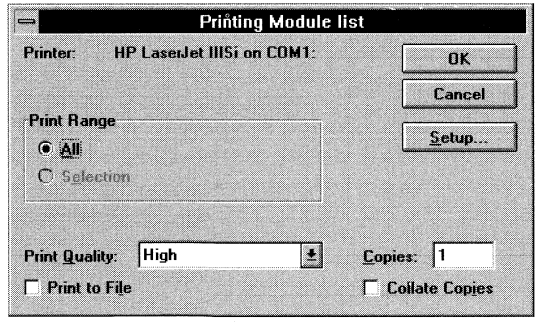

3. Enter the new name in the Module Name edit control.
4. Choose OK.

## Printing a Module List

To print a list of all modules that comprise the current application:

1. Choose the Print toolbar button.

   The Printing Module List dialog box appears:



2. Specify your printing options.
3. Choose OK.

For descriptions of the available standard printing options, see Printing an Application List earlier in this chapter.

# The Entity Browser

Entities form the lowest level in the application hierarchy. An entity is a component that has a distinct name and can be edited (for example, functions, constants, or forms). Applications can share entities directly, or in libraries.

Some of the available entity types are as follows:

- forms
- menus
- field specs
- data servers
- defines
- functions

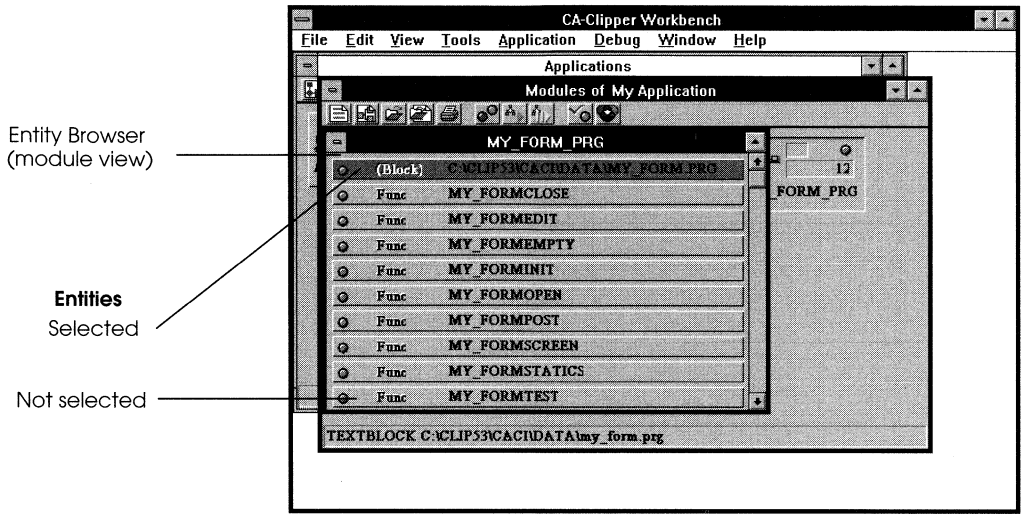
The Workbench allows you to browse entities on two levels using the Entity Browser. You can either view all entities within an *application* or all entities within a *module.* In either case, entities are represented as sculpted 3-D bars with LED-style debugging indicators. Also, entities are grouped by type and are listed in alphabetical order within each group. However, note that the display varies depending on which level you choose for viewing.

The *module-specific* Entity Browser displays its categorical groups in logical order. Note, however, that form, menu, data server, and field spec entities can be viewed only when the Binary Objects module is the *current* module. Other entities are parts of program or header files and can be viewed when a non-binary module is active.

The *application-wide* Entity Browser, on the other hand, displays its categorical groups alphabetically. Additionally, the entities are displayed in a tree structure that can be expanded or collapsed.

## Viewing Entities at the Module Level

To access the Entity Browser and view all the entities defined for a specified module, simply double-click on a module in the Module Browser. For example:



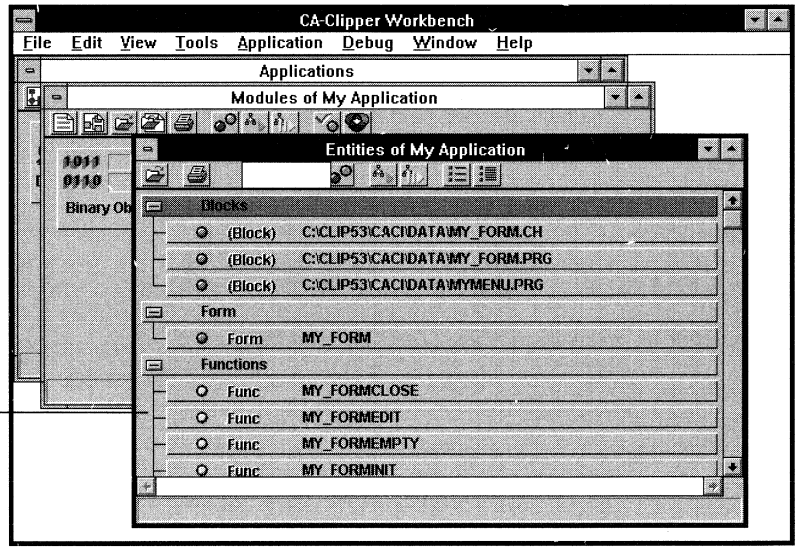

As noted earlier, the module-specific version of the Entity Browser displays its categorical groups in logical order (that is, BLOCK, FUNC, and so on). Also, parentheses around an entity type indicate that it is STATIC, meaning that is visible to the current module only. These sort before regular entities—for example, (BLOCK), BLOCK, (FUNC), FUNC, etc.

**Note:** Entity Browsers are children of their owner Module Browser windows—they do not have their own status bar and are affected by actions to their owner Module Browser window. For example, Entity Browsers are minimized when the owner is minimized.

## Viewing Entities at the Application Level

To view all of the entities defined for an *application*, choose the Entity Browser command from the Tools menu when one of its modules is selected.

The Entity Browser appears:

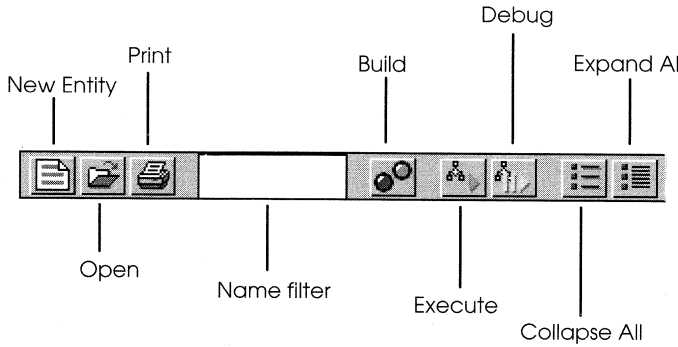



As noted earlier, the categorical groups of entities are arranged alphabetically in a tree structure that can be expanded or collapsed.

Unlike the module-specific Entity Browser, note that this version has a toolbar and a status bar that are described below.

## The Toolbar

The application-wide Entity Browser toolbar has the following buttons:




The New Entity, Open, and Print buttons are described earlier in this chapter. The Collapse All and Expand All buttons and the Name filter are described below. See "Working in the Workbench" for information about Build and Execute. See "Debugging Your Applications" for information about Debug.

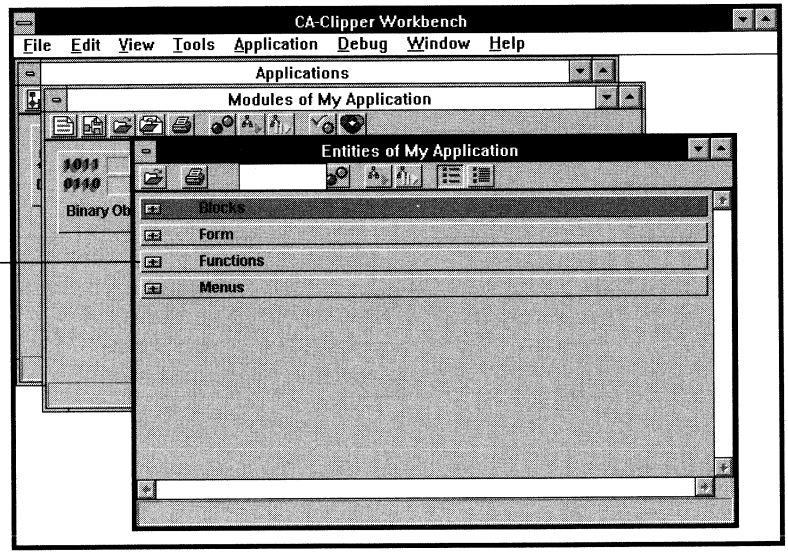
## The Status Bar

The application-wide Entity Browser status bar displays the syntactical prototype for each item as the mouse pointer passes over it, if the Show Entity on Status Bar option has been selected in the System Options dialog box. This functionality is in addition to the usual toolbar and menu command descriptions.

## Collapsing and Expanding the Display

The items in the application-wide Entity Browser are displayed in a collapsible/expandable tree structure that allows selective viewing of entities.

For example, collapse the entire tree by choosing the Collapse All toolbar command. The result is:

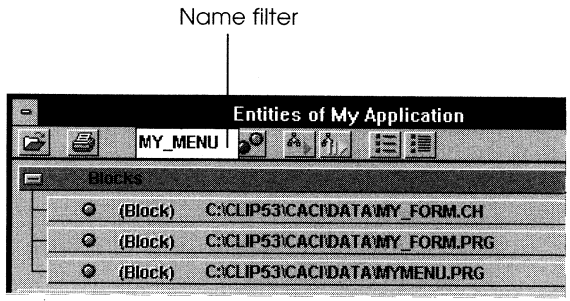



You could then expand only the category you want to view in more detail by clicking on the + *button* to its left. All individual entities within the specified category are subsequently displayed, and the + button changes to a – *button* to reflect this change in status. (Click on the – button to return the specified category to a collapsed condition with its individual entities hidden.)

## Setting a Name Filter

You can also restrict the display at the application level by specifying a particular entity name. For example, to view only entities whose names begin with the string "MY_MENU":

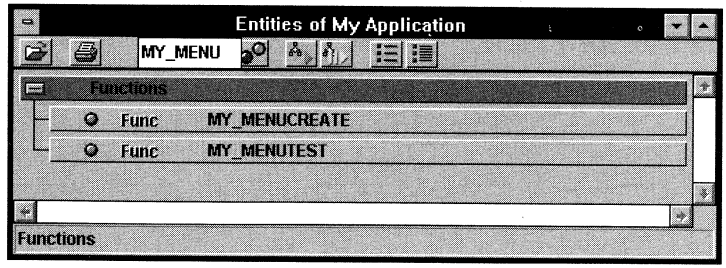
1. Enter **MY_MENU** in the Name filter on the toolbar:




**Note:** No wild cards are allowed. However, you may enter just a portion of the entity name.

2. Press Enter.

The Entity Browser changes, displaying only those entities beginning with "MY_MENU":



## Creating Entities

To create a new entity, simply click on the New Entity toolbar button in the Module Browser, and then select the appropriate editor for the type of entity you want to create from the local pop-up menu that appears. For example, choose Menu Editor to create a new menu entity and open the Menu Editor

The Workbench places the new entity in the Binary Objects module if it is a menu, form, data server, or field spec entity. Otherwise, the system places the new entity in the *current* module if it is a source code edited entity (for example, one created by typing in a keyword such as "FUNCTION").

You may first want to switch to another module or create a new one before clicking the New Entity button, if it is a source code edited entity. See Accessing the Editors in "Working in the Workbench" for details.

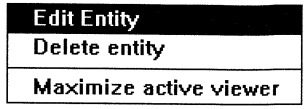
## Editing Entities

As mentioned earlier in this guide, *opening* an entity in the Workbench simply means to display the entity in its associated editor. The techniques for opening an entity vary slightly depending upon the type of entity, as certain entities are stored in the Binary Objects module and others are stored in non-binary, program source file modules.

### From Within the Binary Objects Module

To open an existing menu, form, data server, or field spec entity, simply double-click on that entity button in the Entity Browser for the Binary Objects module; or, when the desired entity button is selected, press Enter.

Alternatively, right-click on the entity while in a module-specific Entity Browser and then choose the Edit Entity command from the local pop-up menu that appears:



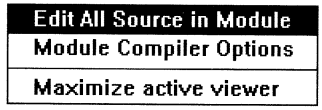

In all cases, the appropriate editor is invoked.

**Tip:** The Maximize Active Viewer command allows you to maximize the current Entity Browser.

From Within a Non-Binary Module

For all other entity types, double-click on the desired entity button in an Entity Browser for a non-binary module; or, when the desired entity button is selected, press Enter.

Alternatively, you can access the Source Code Editor from a *module-specific* Entity Browser and manually edit any entity. Simply right-click on the entity and then choose the Edit All Source in Module command from a local pop-up menu:



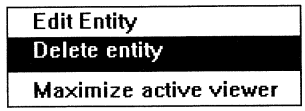

**Tip:** The Module Compiler Options command on this local menu allows you to set compiler options at the module level (see Setting Debugging Options in the "Debugging Your Applications" chapter for more information).

**Note:** From within an application-wide Entity Browser, you can also choose the Open Entity command from the File menu.

## Deleting Entities

To delete an entity, choose the Delete Entity command from the File menu while within the *application-wide* Entity Browser; the system automatically deletes the entity from the module.

Alternatively, right-click on an entity while in a *module-specific* Entity Browser and then choose the Delete Entity command from the local pop-up menu that appears:

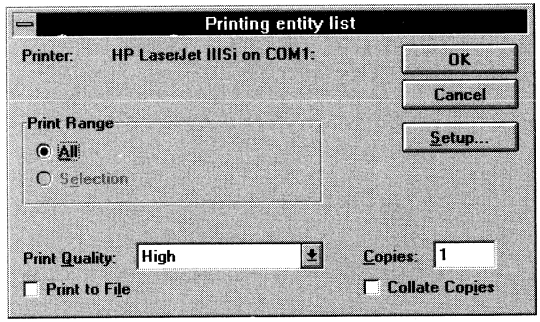



## Printing an Entity List

To print a list of all entities defined for a specified module:

1. Choose the Print toolbar button.

   The Printing Entities dialog box appears:




2. Specify your printing options.
3. Choose OK.

For descriptions of the available standard printing options, see Printing an Application List earlier in this chapter.

# Chapter 4
# Using the Form Editor

## In This Chapter

The Form Editor allows you to create forms and data forms. It provides drag-and-drop placement of buttons, edit controls, list boxes, combo boxes, and other GUI controls. You can then specify properties for your form and its controls, including client area and border colors, border type, and data server links.

When you design and then save a form, the Form Editor automatically generates code that you can use in your application to activate the form. The Form Editor will create a binary entity in the Binary Objects module of the current application, and generate the associated source file (.PRG) and header file (.CH) in separate source file modules.

This chapter describes how to use the Form Editor and shows you how to:

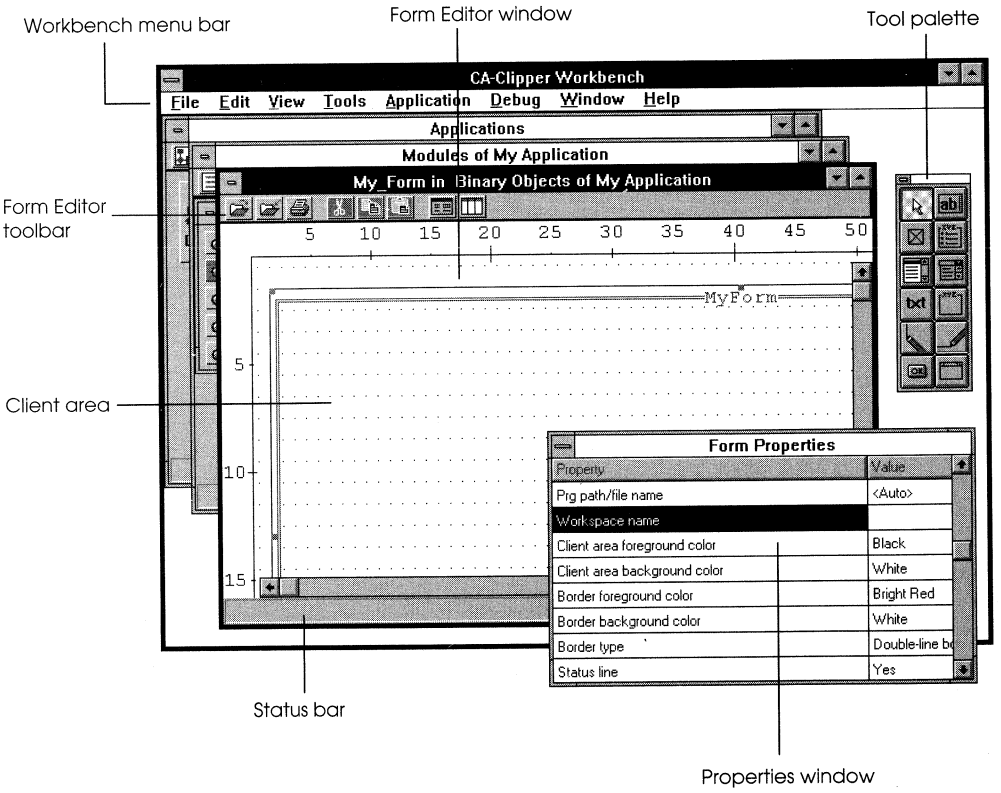
- Create a form and specify its properties (such as caption, border type, and associated data server).
- Place controls (such as push buttons and list boxes) on the form and specify properties for them.
- Create predefined data forms linked to data servers using the Auto Layout feature.
- Modify form and control properties.
- Print using the Form Editor.
- Use the form in your application.

# The Form Editor Workspace

The Form Editor is the primary workspace in the Workbench for creating, viewing, and modifying forms. When you are in the Form Editor, you can:

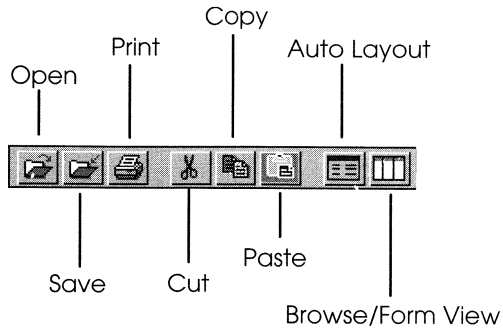
- Create, edit, cut, copy, paste, and print forms and their controls.
- Define properties for forms and their controls.
- Access other browsers and editors, including the Source Code Editor, for defining actions associated with push button controls.

The Form Editor has its own toolbar, status bar, client area, tool palette, and Properties window. For example, when first loaded for a new data form, it looks like this:



## The Toolbar

The Form Editor's toolbar contains the following buttons:

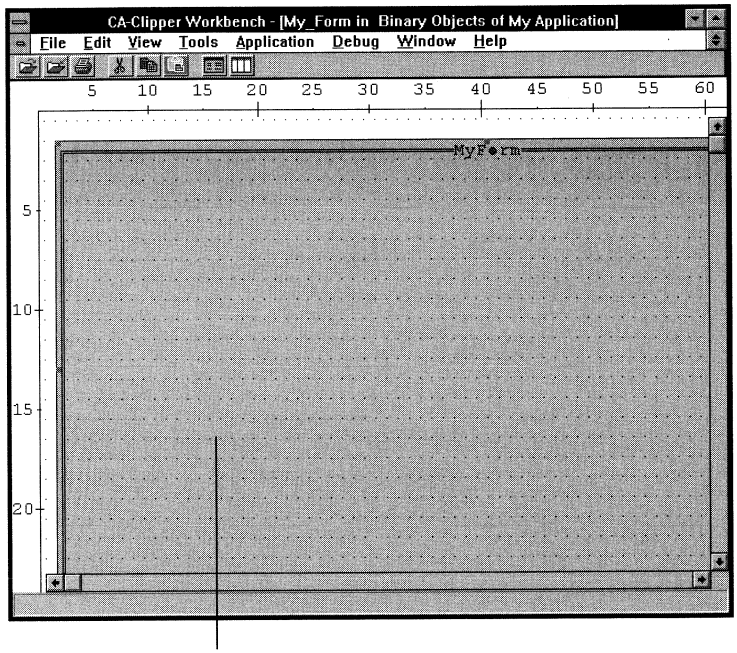



All buttons displayed above are discussed in this chapter.

**Tip:** For a quick description of these toolbar buttons, look at the status bar as the mouse pointer passes over the buttons. Also see your online Help reference.

## The Client Area

When the Form Editor is launched, a blank form appears in the client area:

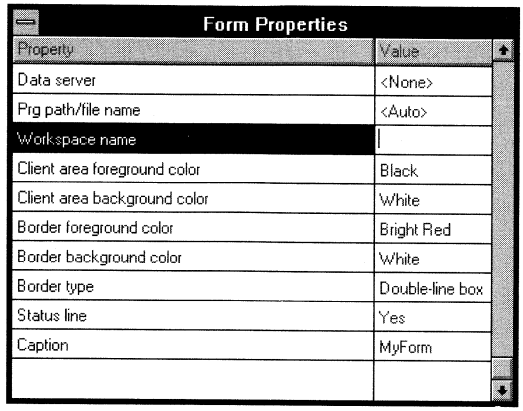



Client area (background shaded for visibilty)

Note that the background in our example has been shaded for greater visibility. Client Area Background Color is just one of many properties that you can specify for any form.

## The Properties Window

When the Form Editor is launched, a *modeless*, floating Properties window, shown below, is automatically opened. Initially, this window allows you to specify properties for the current form:



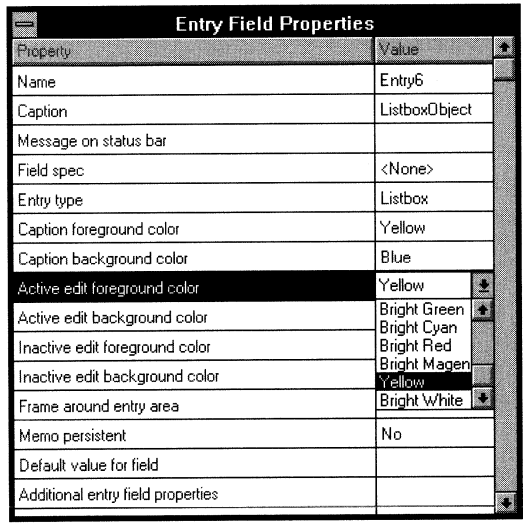

The *Property* column lists all properties that can be specified for the currently selected form, and the *Value* column contains the corresponding cells where you specify a value. For example, you can specify a workspace name, the type of border, or whether text should appear in the status bar.

**Tip:** Use the Properties window's scroll bar to reveal all form properties.

As you develop the form, this window takes on different roles depending on the currently selected item in the Form Editor. For example, if you place a check box control on the form, the Properties window is used to specify properties for the check box when it is selected. (See Types of Controls later in this chapter for detailed information about the types of graphical user interface (GUI) controls you can place on forms.)

Regardless of what is currently displayed in the Properties window, it behaves in the same manner. To use it, simply highlight a property by clicking on it, then use one of the following techniques for specifying its value:

- Enter a new value by typing directly into a single-line edit control or combo box.
- Click on the arrow button and choose a new value from a drop-down list (shown below):

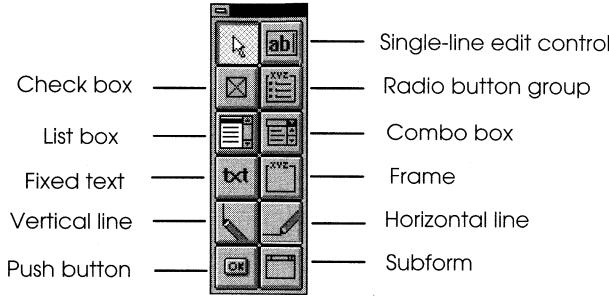



The Properties window is discussed in greater detail in the Specifying Form Properties and Specifying Control Properties sections later in this chapter.

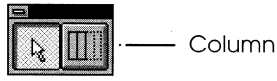
**Note:** The Properties window always remains open until explicitly closed (using the system menu) or until its owner, the Form Editor window, is closed. If explicitly closed, reopen it at any time using the Show Property Window command from the Window menu. Also, because it is a child window, the Properties window is affected by actions to its owner. For example, if the owner window is minimized to an icon, the Properties window will also be minimized.

## The Tool Palette

The Form Editor's tool palette contains a set of icons that allow you to quickly and easily place different types of GUI controls on your forms using the drag-and-drop technique:




Note that when you switch to Browse View, the tool palette changes and displays only a column icon, allowing you to add columns to your form:




(If you prefer, the Form Editor also features an Edit Select from Palette menu command, which allows you to place controls by choosing commands from a menu.)

## Types of Controls

A brief review of the types of controls that you can place on your forms follows:

### Check Boxes

Check boxes indicate a set of options that are either on or off. If more than one check box is present on a form, the user can select as many as are applicable. The state of a check box is indicated in the box to its left. If there is an X in the box, it is selected; otherwise, it is not.

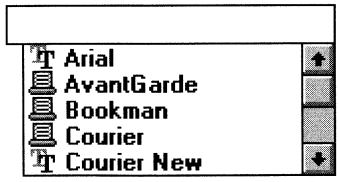
For example, the following check box from the Workbench's Compiler Options dialog box utilizes a logical field for the inclusion of debugging information in your applications. Checking the box would indicate a value of true (.T.), while unchecking it would indicate a value of false (.F.).

☒ Debug Information

### Combo Boxes

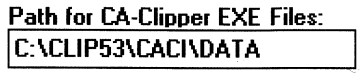
Combo boxes are list boxes with a single-line edit control attached at the top. The user can either type a value directly into the edit control, or click on the Down arrow button to the right to open a list box from which to make a selection. The selection is used to fill in the edit control, which can then be edited.

In a data entry form, you can use a combo box instead of a list box when the field value has more possibilities than you care to list. By placing the most commonly used values in the associated list box, you give the user a quick way to make a selection without removing the flexibility of entering values that are not listed. Shown here is an example from the Workbench's Browsers Font dialog box:



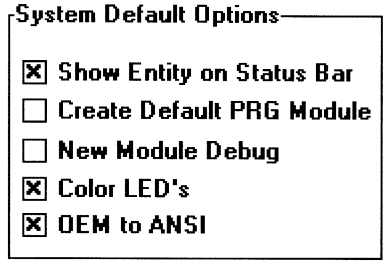
Fixed Text

Fixed text displays a caption or label anywhere within a form. A common use of this type of control is to create a caption for a single-line edit control, a feature that is utilized by the Form Editor's Auto Layout feature. Below is an example from the Workbench's System Options dialog box:



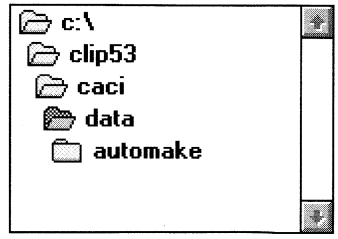

Frames

Frames visually indicate a set of related controls. They provide a caption to describe the controls, but serve no other purpose. They are most often used to display a group of related check boxes. Below is an example from the Workbench's System Options dialog box which allows you to choose various system default options:




List Boxes

List boxes display a list of choices to the user and allow the user to scroll through them and select one. In a dialog box, you might use list boxes to allow the user to select a file name and a path. On a data entry form, you might use a list box to display all possible values for a particular field. This example is from the Workbench's Browse dialog box:



### Push Buttons

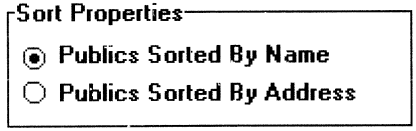
Push buttons react when the user chooses them by generating an event. Some examples of push buttons are the standard OK and Cancel push buttons, shown below, used to close a dialog box:


### Radio Button Groups

Radio button groups visually indicate a group of *radio buttons*. Like a frame, they provide a descriptive caption for the controls they contain, but they have another special purpose—only one of the radio buttons within a radio button group can be selected at a time. When the user chooses a new radio button in the group box, the previously selected one is deselected.

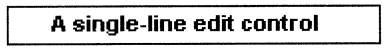
Each radio button group behaves independently. In other words, you can place several groups of radio buttons on the same form, and the user can select one radio button in each radio button group.

Radio button groups let you use radio buttons to present a set of choices to the user. The following example from the Workbench's Link Map Options dialog box allows you to sort public symbols either by address or by name:




### Single-Line Edit Controls

Single-line edit controls are probably the most commonly used controls on data entry forms and are often used to represent fields into which the user may type almost any value:




### Subforms

Subforms are simply data forms that you place on other data forms as controls. Typically you would use a subform to show a master-detail relationship between two related data servers. For more information, refer to Data Forms in the "Using the Form Editor" chapter of this guide.

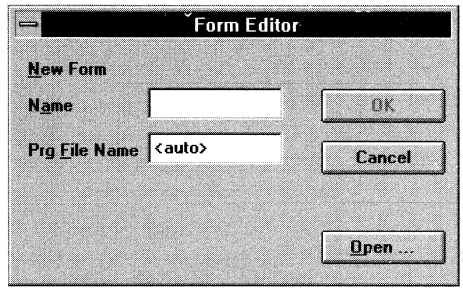
# Creating a Form

The basic steps below illustrate the ease with which you can design fixed-character forms for your CA-Clipper applications. Just follow our example, a typical deletion confirmation dialog box for a simple accounting program called "ABC123."

1. Create a new application and name it **ABC123**.
2. Start the Form Editor.

   The Form Editor is accessed using the Tools menu or the New Entity toolbar button from within the Binary Objects module.

   The Form Editor dialog box appears:




3. Enter a name for the new form in the Name edit control. For example, **Confirmation Box**.

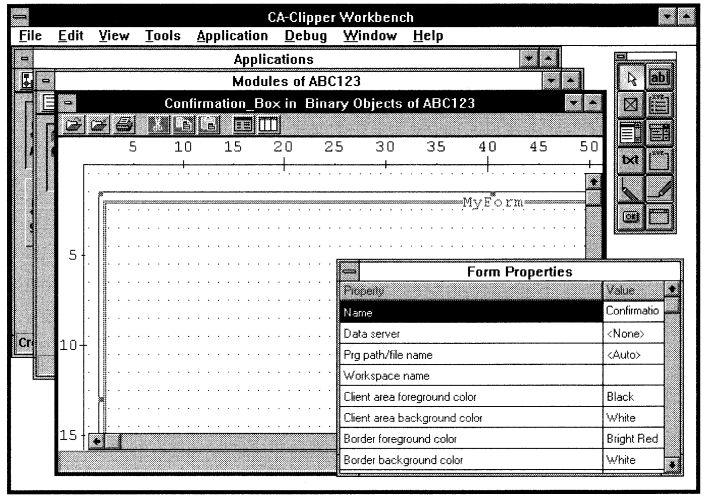
   Blank spaces within the name will be converted to underscores. This name will be used in the source code generated by the Form Editor to create a form entity and the .PRG and .CH files and their associated modules, so it must not conflict with other entity names in your application.

   **Note:** You can choose to edit an existing form instead of creating a new one by selecting the Open button. When clicked, the Form Editor dialog box changes so that it contains an Open Form list box, from which you can choose an existing form entity. Note that there is also a New push button that, when clicked, toggles you back to the above dialog box.

4. Optionally, specify a file name in the PRG File Name edit control for the .PRG file that will be generated. The default is <Auto>, which uses the default path defined via the System Options dialog box and takes the first eight (8) characters of the form name as the file name with .PRG as the extension.

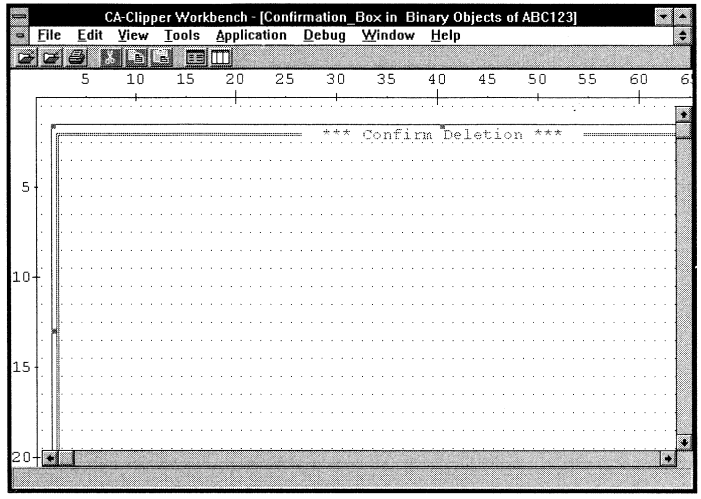
5. Choose OK.

   The Form Editor appears:



6. Specify properties for the form.

   For example, change the Caption property from "MyForm" to *** **Confirm Deletion** ***, and select Bright Red for the Border Foreground property and White for the Border Background property. The Form Editor now reflects these changes:




   See Specifying Form Properties later in this chapter for more information about each of the available form properties.

7. Customize the form by selecting controls from the tool palette (or using the Edit Select from Palette menu commands), placing them in the desired locations on the blank form, and specifying properties for them.

   See Placing Controls for more information using our example.

**Note:** You can also use the Auto Layout feature to place predefined controls on the form. See the Using Auto Layout section later in this chapter for more information.

8. Choose the Save toolbar button.

**Note:** This last step can be repeated whenever you make changes to the form and want to save your work without closing the Form Editor.

**Tip:** The Open toolbar button can be used at any time to begin editing another form without shutting down the Form Editor. Unless you save the form you are currently working with before starting a new session, you will be prompted to do so before the Form Editor opens the new form for editing.

## Defining Default Form Settings

From within the Form Editor, you can define *default* property settings for your forms and for the controls you place upon them using a series of dialog boxes. These default settings will apply to *all* new forms that you create.

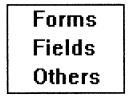
**Note:** You can *override* the system default settings for an individual form and/or its controls via the Properties window. See the Specifying Form Properties and Specifying Control Properties sections later in this chapter.

### New Forms

To define default property settings for all new forms:

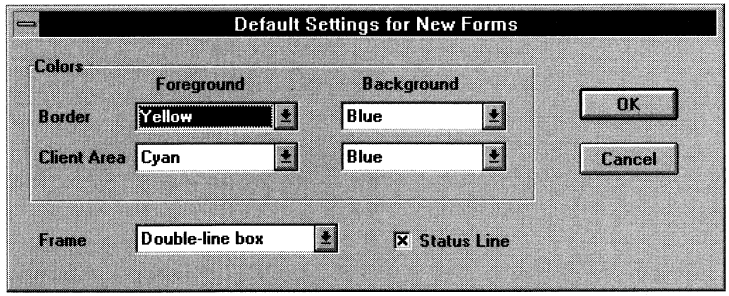
1. Choose Defaults from the File menu.

   A submenu appears:

2. Choose the Forms command from the submenu.

   The Default Settings for New Forms dialog box appears:

3. Choose foreground colors for the form's border and client area.

   The system default values are Yellow and Cyan, respectively. Choosing <Auto> from the drop-down list box indicates no specific color instruction.

4. Choose background colors for the form's border and client area.

   The system default in each case is Blue. Choosing <Auto> from the drop-down list box indicates no specific color instruction.

5. Select a border type for the form from the Frame drop-down list box.

   The system default value is Double-line Box.

6. Specify whether the form should have its own status line using the Status Line check box.

   The system default is checked (or Yes).

7. Choose OK.

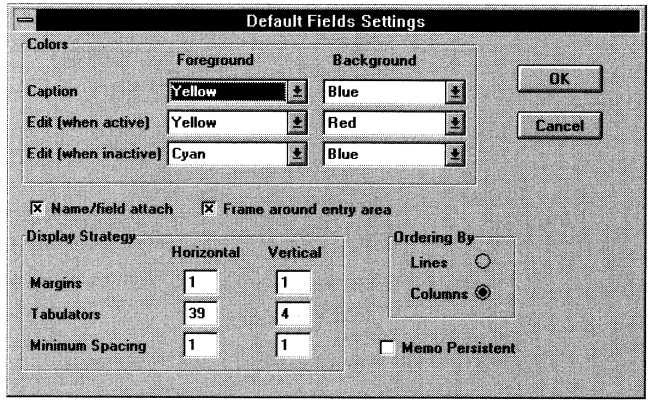
See Specifying Form Properties for more information about form properties.

## Fields

To define default color and display settings for your form's data entry fields:

1. Choose the Fields command from the File Defaults submenu.

   The Default Fields Settings dialog box appears:




2. Choose foreground colors for captions and for data entry controls when active and inactive.

   The system defaults in each case are shown. Choosing <Auto> from the drop-down list box indicates no specific color instruction.

**Note:** For printing reproduction purposes, black is used for the foreground and white for the background for forms, fields, text, and lines in our sample application, ABC123, instead of the default CA-Clipper DOS colors.

3. Choose background colors for captions and for data entry controls when active and inactive.

   The system defaults in each case are shown. Choosing <Auto> from the drop-down list box indicates no specific color instruction.

4. Specify whether each data entry field should be created as a single object, including caption and entry area.

   If the Name/Field Attach check box is unchecked, each data entry field is created as two different objects, i.e., one "text" object for the caption and one "edit" object with no caption for the entry area.

   The default setting is checked (or Yes).

5. Specify whether each data entry field should be framed using the Frame Around Entry Area check box.

   The default setting is checked (or Yes).

6. Specify whether memo fields should permanently display their contents using the Memo Persistent check box.

   The default setting is unchecked (or No).

7. Define the margins, tabulators, and minimum spacing for the fields within the form using the Display Strategy group box options.

   The default horizontal and vertical settings are 1 and 1, 39 and 4, and 1 and 1, respectively.

   **Note:** The Display Strategy group box settings are used by the Workbench for its Auto Arrange and Auto Layout features. See Customizing the Form Display for more information about the Auto Arrange feature, and Using Auto Layout in the Data Forms section for more information about the Auto Layout feature.

8. Specify whether fields should be ordered by lines or by columns using the Ordering By radio button options.

   **Note:** The Ordering By group box setting is also used by the Workbench for its Auto Arrange and Auto Layout features.

9. Choose OK.

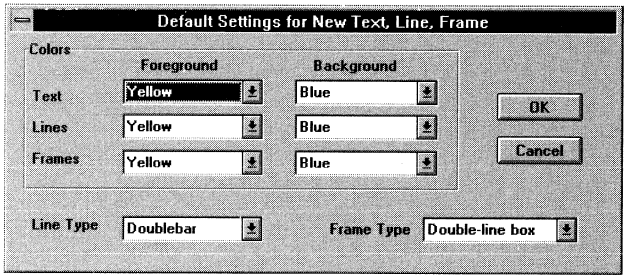
See Specifying Control Properties for more information about field and control properties.

### Text, Lines, and Frames

To define default settings for all new text, lines, and frames placed on your forms:

1. Choose the Others command from the File Defaults submenu.

   The Default Settings for New Text, Line, Frame dialog box appears:




2. Choose foreground colors for the form's text, line, and frame controls.

   The system default value in each case is Yellow. Choosing <Auto> from the drop-down list box indicates no specific color instruction.

3. Choose background colors for the form's text, line, and frame controls.

   The system default value in each case is Blue. Choosing <Auto> from the drop-down list box indicates no specific color instruction.

4. Choose a line type from the Line Type drop-down list box for the form's vertical and horizontal line controls.

   The system default is Doublebar.

5. Choose the type of frame from the Frame Type drop-down list box for the form's frame controls.

   The system default is Double-line Box.

6. Choose OK.

See Specifying Control Properties for more information about control properties.

## Specifying Form Properties

You define properties for a form using the Properties window discussed earlier. Each of the form properties is described below:

Name

Enter the name of your form *entity*. This property value is required and is automatically filled in with the name you entered in the Form Editor dialog box when you first created the form.

Data Server

Enter or choose the name of the data server that is associated with this form. The default is <None>.

**Note:** This property is required for *data entry fields*, and is automatically filled in when you choose the Edit Auto Layout menu command. All data servers available to the application are listed in alphabetical order in the Data Server property's drop-down list box. See the Using Auto Layout section later in this chapter for more information.

PRG Path/File Name

Specify a file name *prefix* (and the path for the form's .PRG file and .CH file if different from the default path for PRG files). CA-Clipper automatically adds the .PRG and .CH extensions.

The default is the name you entered in the PRG File Name edit control of the Form Editor dialog box when you first created the form.

| | |
|---|---|
| Workspace Name | Specify the name of a work area or the database file's *alias*. |
| Client Area Foreground Color | Select a color from the drop-down list box for the client area's foreground, which includes all controls, text, and the border. The system default is Cyan.<br><br>**Note:** You can customize your forms by overriding any or all of the system default form settings via the Default Settings for New Forms dialog box discussed earlier. |
| Client Area Background Color | Select a color from the drop-down list box for the client area's background. The system default is Blue. |
| Border Foreground Color | Optionally, specify a different color for the border only. The system default is Yellow. |
| Border Background Color | Optionally, specify a different color for the border's background area only. The system default is Blue. |
| Border Type | Select a border type from the drop-down list box for your form. Valid options are: Double-line Box, Single-line Box, Single-line Top Double-line Sides, and Double-line Top Single-line Sides. The system default is Double-line Box. |
| Status Line | Specify whether the form should have its own status line. The system default is Yes. |
| Caption | Enter the text that will appear on the form's title line. The default is "MyForm". |

## Placing Controls

Placing controls on a form in the Form Editor is easy. You have a choice of placement methods, as well as the option of using a grid to align them accurately.

### Placement Methods

To place controls on a form, you can use either the tool palette or the Select from Palette commands in the Edit menu.

Tool Palette

The tool palette supports two methods of control placement:

- Place the mouse pointer on an icon in the tool palette, press the left mouse button, hold it down and drag the mouse to the desired position on the form. When you reach the desired location, release the left mouse button.

  This is known as *drag-and-drop*. Note that as you start to drag the mouse, a cross-hairs positioning icon appears.

- Place the mouse pointer on an icon in the tool palette and press the left mouse button, this time releasing the button. Move the mouse pointer to the desired position in the form template and then click the mouse again.

After either method, note how the Properties window changes focus so that you can now specify properties for the newly created control.

Menu Commands

As an alternative to using the tool palette to place controls, you can use the Select from Palette commands in the Edit menu. After choosing one of these menu commands, move the mouse pointer to the desired position in the form template and click the mouse to place the new control.

## Displaying the Grid

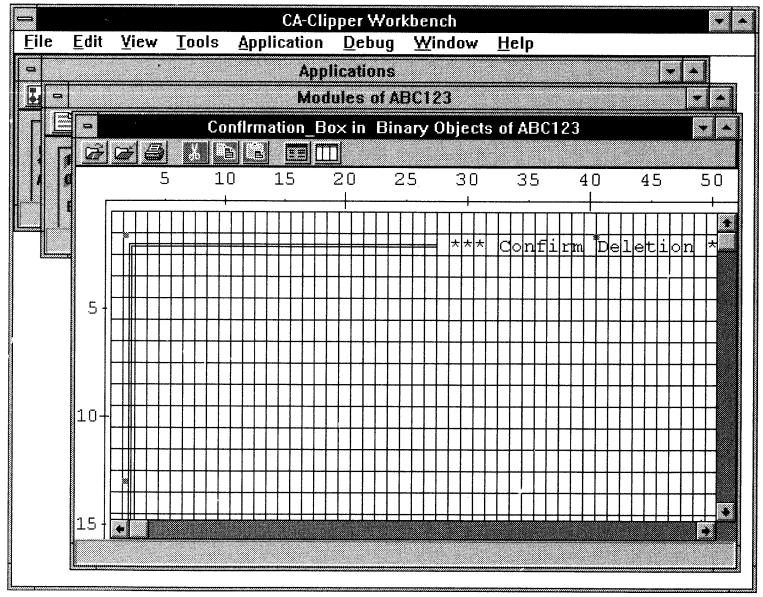
When placing or arranging controls on a form, you might find it useful to display the *grid*. This grid represents the DOS screen and allows you to align controls quickly and easily with accurate placement.

**Note:** Whether it is displayed, a form's controls are always aligned using the Workbench's grid. If the form is moved beyond line 25, the following warning appears: "You need to switch to 43 or 50 lines mode to use this form size." To change the number of rows and columns in the display mode, use the SETMODE() function. For more information, refer to the *Reference Guide, Volume 2*.

To display the grid:

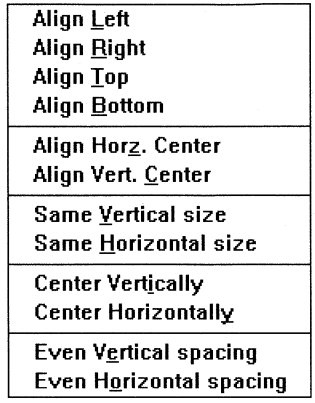
1. Choose the Grid command from the View menu.
2. Select the type of grid by choosing either the Dots or Lines command from the submenu that appears.

Below is an example using the Lines menu command:



### Aligning an Individual Control on the Grid

To align or center a specified control on the grid, you can either move it with the mouse; or you can right-click on it and then choose an appropriate command from the local pop-up menu that appears:




### Aligning All Controls

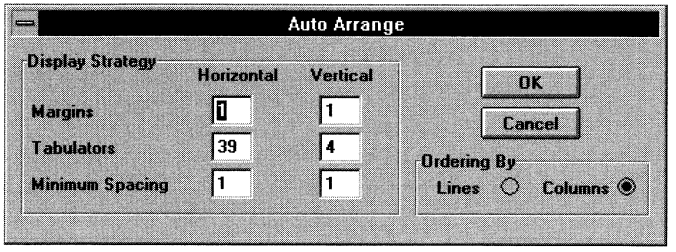
To align or center *all* controls on the grid:

1. Highlight all controls by choosing the Select All command from the Edit menu.
2. Right-click on any control.

   The same local pop-up menu appears.
3. Choose an appropriate menu command from the local pop-up menu.

## Customizing the Form Display

Note that you can also customize the form's display. To do so, choose the Auto Arrange command from the Edit menu. Selecting this command displays the following dialog box:




To rearrange the positions of all controls already defined for a form:

1. Choose either the Lines or Columns radio button in the Ordering By group box.
2. Enter different values in the Horizontal and Vertical edit controls for Margins, Tabulators, and/or Minimum Spacing.
3. Select OK.

**Note:** The default values that appear in this dialog box are those defined previously using the Default Fields Settings dialog box (discussed earlier in Defining Default Form Settings).

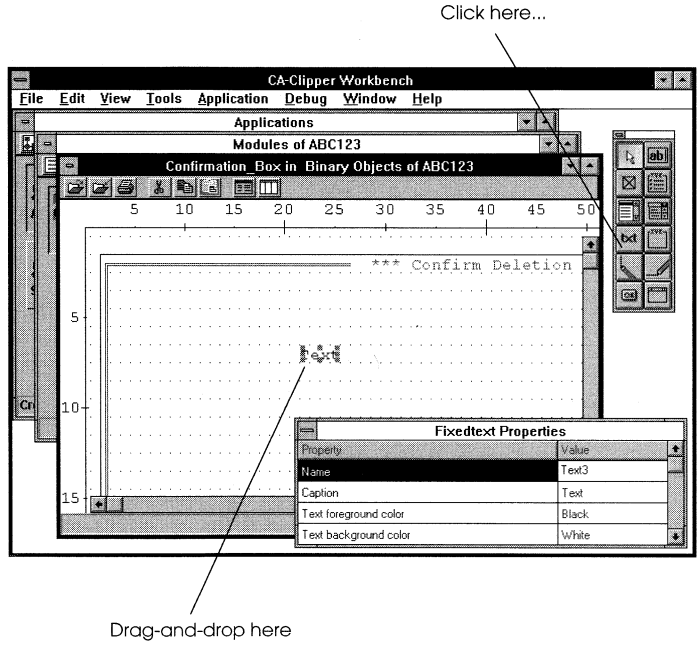
## Adding Controls

The remaining Form Editor features will be demonstrated using several examples, including our unfinished Confirmation_Box form entity which needs some text and push buttons.

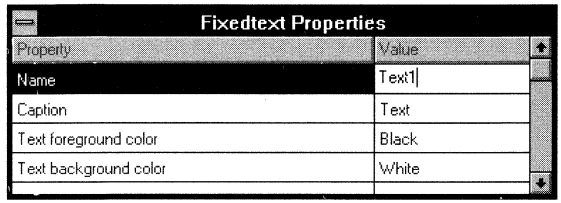
### Adding Text

To add a line of fixed text:

1. Click on the Fixed Text icon in the tool palette and drag-and-drop a fixed text control onto the form:



2. Click on the fixed text control. The Fixedtext Properties window appears:

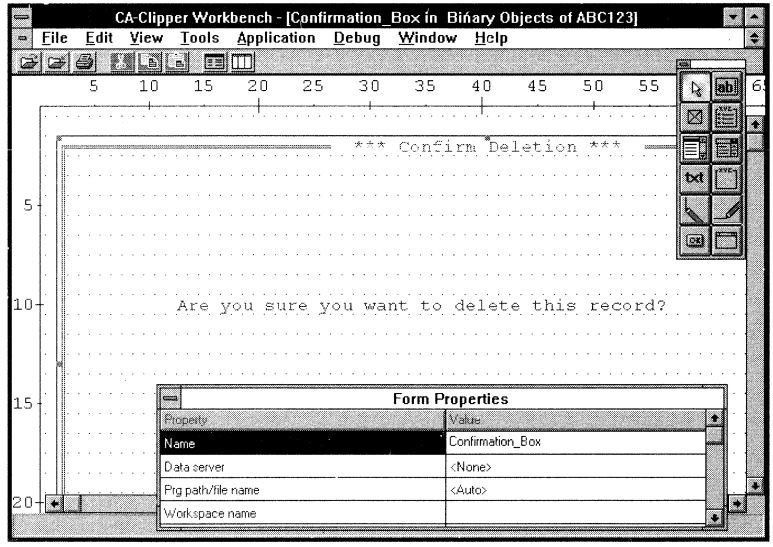



3. Next, specify property values for the fixed text control.

   For example, replace "Text" in the Caption value cell with **Are you sure you want to delete this record?** Also, select **Red** from the drop-down list box for the Text Foreground Color property.

   See the Specifying Control Properties section for more information about each of the available controls and their respective properties.

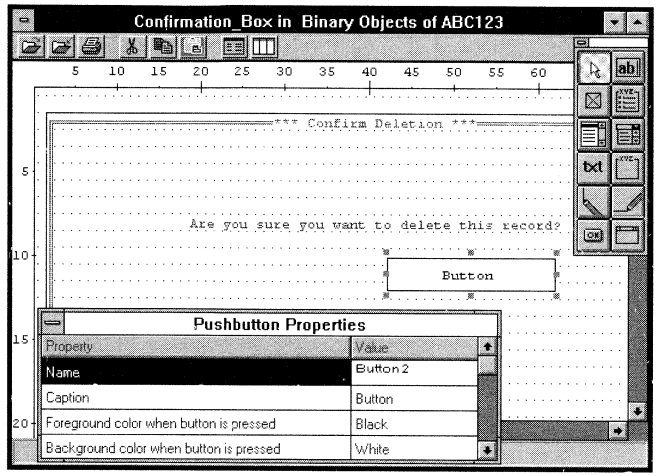
The Form Editor now reflects these changes:



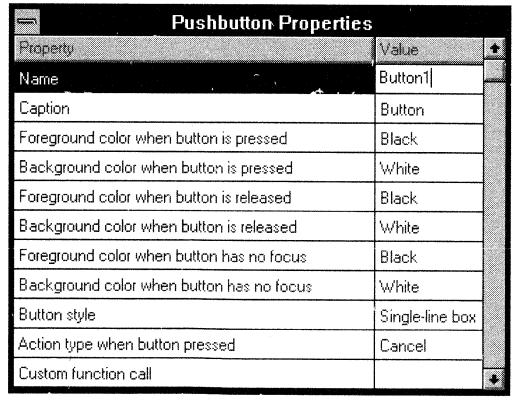
## Adding Push Buttons

To add two push buttons to our example:

1. Click on the Push Button icon in the tool palette and drag-and-drop a push button control onto the form under the fixed text control:




2. Repeat step 1.
3. Click on the first push button control. The Properties window changes accordingly:



4. Next, specify the first push button's property values.

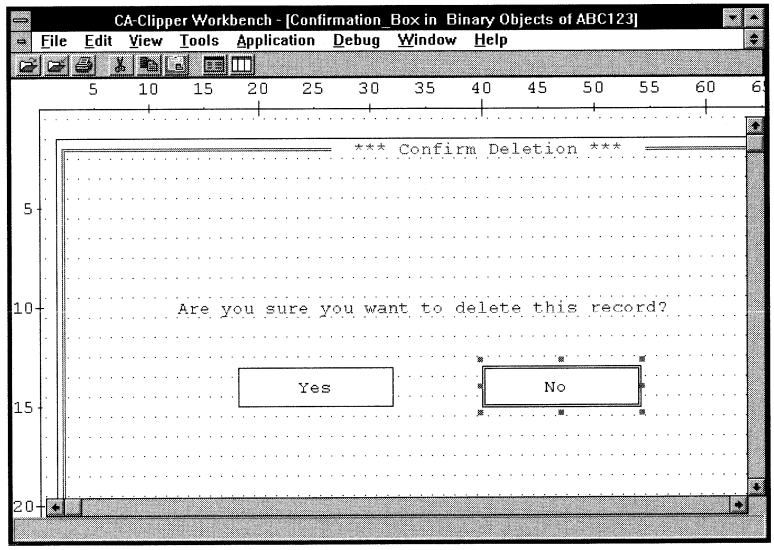
   For example, replace "Button" in the Caption value cell with **Yes**. Also, select Single-line Box from the drop-down list box for the Button Style property, and select OK for the Action Type When Button Pressed property.

   See Push Button Controls and Actions for more information about associating predefined actions with push buttons.

5. Repeat steps 3 and 4 for the second push button, replacing "Button" with **No** for the caption and selecting Cancel for the action type. Select Double-line Box for the button style.

6. Choose Save.

The Form Editor now reflects these changes:




See Modifying a Form later in this chapter for information about resizing forms and controls, and modifying other form and control properties. Also, see Using the Form in an Application for information about linking the Confirm Deletion dialog box to a data form.

## Specifying Control Properties

As you have just seen, defining a control's properties using the Properties window is the next step after placing the control on the form. This is not a requirement because the Form Editor will generate defaults that will be sufficient to allow you to save the form in all cases; however, there are properties that you may need to define to make certain controls operational.

**Tip:** Remember, once you begin to customize a form, the role of the Properties window changes depending on the currently selected item in the Form Editor. Also, if you have customized colors for forms, fields, text, lines, and/or frames using the Defaults command from the File menu, the corresponding settings default to the Properties window instead of the system default settings.

**Note:** Regarding all form controls, a default memory variable is associated with each control, unless the control is already associated with a field.

Below are property descriptions for each type of form control. Common to *all* controls are the Name and/or Caption properties.

Name

Enter the name of the control. This property value is required and automatically filled in with a default name when you place a control on a form. For example, when you place the first push button on a form, it is named "Button1."

Caption

Enter the text that identifies a control in the form display. For a push button, for example, this is the label that is displayed on the button. By default, the caption is defined by the control type when you place it on the window (for example, the default caption for all push buttons is "Button").

### Single-line Edit Controls

Placing a single-line edit control on the form allows the end user to enter textual data.

Message on Status Bar — Enter the text that should appear in the status bar when the single-line edit control has focus in the form display.

Field Spec — Enter or choose a field spec to be associated with the single-line edit control.

**Note:** When you associate a field spec with *any* control, all properties defined for the field spec (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, see the "Defining Data Servers and Field Specs" chapter.

Entry Type — Enter the type of data entry field. Valid choices are: Edit, Listbox, Combobox, RadioGroup, and Checkbox. The default is Edit.

**Note:** This property is common to the various types of data entry field controls; therefore, the Entry Type property value defaults from the type of control *initially* selected from the tool palette. You may, however, change the type of data entry field once the control has been placed on the form.

Caption Foreground Color — Enter or choose a color from the combo box for the caption text. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used.

**Note:** See Setting Default Form Settings earlier in this chapter for more information about overriding the system settings and customizing your default form and control settings.

Caption Background Color — Enter or choose a color to be used for shading the background of the caption display. The default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used.

| | |
|---|---|
| Active Edit Foreground Color | Choose a color from the combo box for the data entry text when the edit control is active. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Active Edit Background Color | Choose a color from the combo box for the data entry background area when the edit control is active. The system default value is Red. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Inactive Edit Foreground Color | Choose a color from the combo box for the data entry text when the edit control is inactive. The system default value is Cyan. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Inactive Edit Background Color | Choose a color from the combo box for the data entry background area when the edit control is inactive. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Frame Around Entry Area | Specify whether the control's data entry area should be framed. Valid options are Yes and No. The default is Yes. |
| Memo Persistent | Specify whether the display of a memo field is permanent. Valid options are Yes and No. The default is No. |
| Default Value for Field | Enter a default value for the field. |
| Additional Entry Field Properties | Not available for single-line edit controls. |

## Check Boxes

Placing a check box control on your form allows the end user to set conditions, providing an ON/OFF (or true/false) response to a condition.

Message on Status Bar — Enter the text that should appear in the status bar when the check box control has focus in the form display.

Field Spec — Enter or choose a field spec to be associated with the check box.

Entry Type — Enter the type of data entry field. Valid choices are: Edit, Listbox, Combobox, RadioGroup, and Checkbox. The default is Checkbox.

Caption Foreground Color — Enter or choose a color from the combo box for the caption text. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used.

Caption Background Color — Enter or choose a color to be used for shading the background of the caption display. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used.

Active Edit Foreground Color — Choose a color from the combo box for the data entry text when the check box is active. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used.

Active Edit Background Color — Choose a color from the combo box for the data entry background area when the check box is active. The system default value is Red. Choosing <Auto> means that the corresponding customized default setting for fields is used.

Inactive Edit Foreground Color — Choose a color from the combo box for the data entry text when the check box is inactive. The system default value is Cyan. Choosing <Auto> means that the corresponding customized default setting for fields is used.

| | |
|---|---|
| Inactive Edit Background Color | Choose a color from the combo box for the data entry background area when the check box is inactive. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Frame Around Entry Area | Specify whether the check box should be framed. Valid options are Yes and No. The default is Yes. |
| Memo Persistent | Specify whether the display of a memo field is permanent. Valid options are Yes and No. The default is No. |
| Default Value for Field | Enter a default value for the field. Valid options for check boxes are true (.T.) and false (.F.). |
| Additional Entry Field Properties | Not available for check boxes. |

### List Boxes

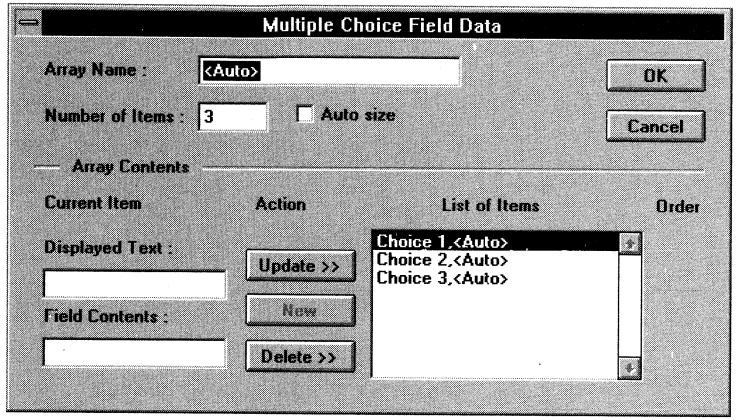
When you select the list box icon from the tool palette, a list box control is placed on your form which contains, by default, three choices. You may increase or decrease the number of choices simply by resizing the control. Note that "Choice1" is delineated as the default selection.

| | |
|---|---|
| Message on Status Bar | Enter the text that should appear in the status bar when the list box control has focus in the form display. |
| Field Spec | Enter or choose a field spec to be associated with the list box. |
| Entry Type | Enter the type of data entry field. Valid choices are: Edit, Listbox, Combobox, RadioGroup, and Checkbox. The default is Listbox. |
| Caption Foreground Color | Enter or choose a color from the combo box for the caption text. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used. |

| | |
|---|---|
| Caption Background Color | Enter or choose a color to be used for shading the background of the caption display. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Active Edit Foreground Color | Choose a color from the combo box for the highlighted item in the list box. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Active Edit Background Color | Choose a color from the combo box for the background area of the highlighted item in the list box. The system default value is Red. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Inactive Edit Foreground Color | Choose a color from the combo box for the unselected items in the list box. The system default value is Cyan. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Inactive Edit Background Color | Choose a color from the combo box for the background area of the unselected items in the list box. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Frame Around Entry Area | Specify whether the list box should be framed. Valid options are Yes and No. The default is Yes. |
| Memo Persistent | Specify whether the display of a memo field is permanent. Valid options are Yes and No. The default is No. |
| Default Value for Field | Enter a default value for the field. |

## Additional Entry Field Properties

Specify the array for a list box, combo box, or radio group box using the Multiple Choice Field Data dialog box, which appears when you click on the value cell for this property and then click on the *ellipsis button* that appears:




**Tip:** You can also access this dialog box by double-clicking on the control itself.

To enter an array of values:

1. Enter the name of an external array in the Array Name edit control. The default is <Auto>, which means the array is local and defaults from the form's name.
2. Enter the number of items in the array. The default is 3.

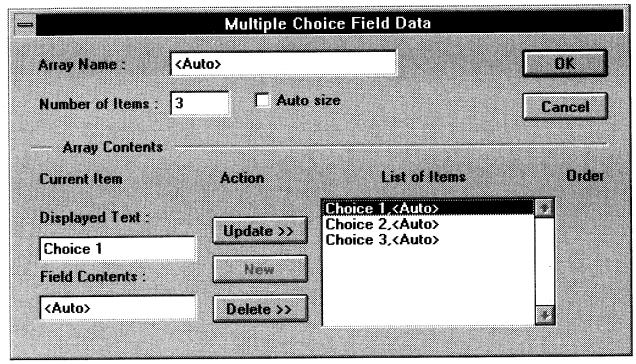
   Increasing the value of this option activates the New push button.
3. Optionally choose the Auto Size option.

   If checked, this option automatically sizes the array based upon the number of elements defined to it. The default setting is unchecked.

   **Note:** The Auto Size option is not available for radio button groups.

4. Click on the first item in the List of Items list box.

   The text "Choice 1" and "<Auto>" appear in the Displayed Text and Field Contents edit controls, respectively:



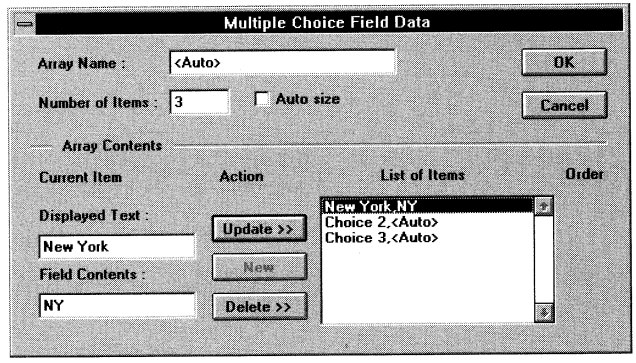

5. Replace "Choice 1" with the name of the first element in your array (for example, **New York**).

6. Replace "<Auto>" in the corresponding Field Contents edit control (for example, **NY**).

   The default is <Auto>, which means that the same text entered in the Displayed Text edit control is used for the field contents.

7. Click on the Update push button.

   The text is added to the List of Items list box:



8. Repeat steps 4 through 7 to define Choice 2 and Choice 3.

   **Note:** If you have specified a number greater than 3 in the Number of Items edit control or selected the Auto Size option, the New push button is activated. After defining the third element, click on New and an additional choice (, <Auto>) is added to the List of Items list box. Then, repeat steps 5 through 7 to define the new element.

9. Choose OK.

## Combo Boxes

Combo boxes, commonly referred to as *drop-down list boxes*, allow the end user to select either the currently selected item in a list or another item from the drop-down list. When you select the combo box icon from the tool palette, a combo box control is placed on your form which contains, by default, four choices. You may increase or decrease the number of choices simply by resizing the control. Note that "Choice1" is delineated as the default selection.

Message on Status Bar — Enter the text that should appear in the status bar when the combo box control has focus in the form display.

Field Spec — Enter or choose a field spec to be associated with the combo box.

Entry Type — Enter the type of data entry field. Valid choices are: Edit, Listbox, Combobox, RadioGroup, and Checkbox. The default is Combobox.

Caption Foreground Color — Enter or choose a color from the combo box for the caption text. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used.

Caption Background Color — Enter or choose a color to be used for shading the background of the caption display. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used.

| | |
|---|---|
| Active Edit Foreground Color | Choose a color for the highlighted item in the combo box. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Active Edit Background Color | Choose a color for the background area of the highlighted item in the combo box. The system default value is Red. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Inactive Edit Foreground Color | Choose a color for the unselected items in the combo box. The system default value is Cyan. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Inactive Edit Background Color | Choose a color for the background area of the unselected items in the combo box. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Frame Around Entry Area | Specify whether the combo box should be framed. Valid options are Yes and No. The default is Yes. |
| Memo Persistent | Specify whether the display of a memo field is permanent. Valid options are Yes and No. The default is No. |
| Default Value for Field | Enter a default value for the field. |
| Additional Entry Field Properties | See List Boxes for a description of this property. |

## Push Buttons

When you select the push button icon from the tool palette, a push button control is placed on your form. You must then associate an *action* with this push button control, so that when the push button is clicked, the form always calls the specified action. The Form Editor makes this task easy for you, providing two predefined actions—OK and Cancel. Additionally, you can select the CustomFunction action value and specify a custom function call.

Foreground Color When Button Is Pressed

Choose a foreground color from the combo box for the push button when it is pressed or clicked by the end user. The system default value is Yellow. Choosing <Auto> means no specific color instruction.

Background Color When Button Is Pressed

Choose a background color from the combo box for the push button when it is pressed or clicked by the end user. The system default value is Red. Choosing <Auto> means no specific color instruction.

Foreground Color When Button Is Released

Choose a foreground color from the combo box for the push button when it is released by the end user. The system default value is Cyan. Choosing <Auto> means no specific color instruction.

Background Color When Button Is Released

Choose a background color from the combo box for the push button when it is released by the end user. The system default value is Blue. Choosing <Auto> means no specific color instruction.

Foreground Color When Button Has No Focus

Choose a foreground color from the combo box for the push button when it has no focus. The system default value is Yellow. Choosing <Auto> means no specific color instruction.

Background Color When Button Has No Focus

Choose a background color from the combo box for the push button when it has no focus. The system default value is Blue. Choosing <Auto> means no specific color instruction.

| | |
|---|---|
| Button Style | Select a style from the combo box for the push button. Valid options are: None, Delimiter, Single-line Box, Double-line Box, Single-line Top Double-line Sides, and Double-line Top Single-line Sides. The default is Single-line Box. |
| Action Type When Button Pressed | Enter or select the type of action to be performed when the button is clicked. Valid values are: OK (i.e., Enter), Cancel (i.e., Esc), and CustomFunction. The default is Cancel. |
| Custom Function Call | Specify a function call for the push button if the type of action selected above was CustomFunction. |

### Radio Button Groups

When you select the radio button group icon from the tool palette, a radio button group box is placed on your form. By default, this group box contains three radio buttons with "Choice1" delineated as the default. Radio buttons provide the end user with mutually exclusive responses to a condition where only one choice is appropriate.

| | |
|---|---|
| Message on Status Bar | Enter the text that should appear in the status bar when the radio button group box has focus in the form display. |
| Field Spec | Enter or choose a field spec to be associated with the radio button group box. |
| Entry Type | Enter the type of data entry field. Valid choices are: Edit, Listbox, Combobox, RadioGroup, and Checkbox. The default is RadioGroup. |
| Caption Foreground Color | Enter or choose a color from the combo box for the caption text. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Caption Background Color | Enter or choose a color to be used for shading the background of the caption display. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used. |

| | |
|---|---|
| Active Edit Foreground Color | Choose a color for the radio button when the radio button is selected. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Active Edit Background Color | Choose a color for the background area of the radio button when it is selected. The system default value is Red. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Inactive Edit Foreground Color | Choose a color for the radio button when it is unselected. The system default value is Cyan. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Inactive Edit Background Color | Choose a color for the background area of the radio button when it is unselected. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for fields is used. |
| Frame Around Entry Area | Specify whether the radio button group should be framed. Valid options are Yes and No. The default is Yes. |
| Memo Persistent | Specify whether the display of a memo field is permanent. Valid options are Yes and No. The default is No. |
| Default Value for Field | Enter a default value for the field. Valid options for radio buttons are true (.T.) and false (.F.). |
| Additional Entry Field Properties | See List Boxes for a description of this property. |

## Fixed Text

Placing a fixed text control on a form allows you to define static text for your form, such as labels for controls or instructions to the end user.

Text Foreground Color

Enter or choose a color from the combo box for the fixed text control. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for text is used.

Text Background Color

Enter or choose a color to be used for shading the background of the fixed text control. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for text is used.

## Frames

Choosing the frame icon from the tool palette allows you to place a frame, or *group box*, around related controls.

Frame Foreground Color

Enter or choose a color from the combo box for the frame and its caption text. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for frames is used.

Frame Background Color

Enter or choose a color from the combo box for the frame's background. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for frames is used.

Frame Border Type

Select a border type for the frame control. Valid options are: Double-line Box, Single-line Box, Single-line Top Double-line Sides, and Double-line Top Single-line Sides. The default is Double-line Box.

### Vertical Lines

Choosing the vertical line icon from the tool palette allows you to draw vertical lines of varying lengths on your form.

Line Foreground Color — Enter or choose a color from the combo box for the vertical line. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for lines is used.

Line Background Color — Enter or choose a color from the combo box for the vertical line's background area. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for lines is used.

Line Style — Select a line style for the vertical line control. Valid options are: Doublebar and Singlebar. The system default value is Doublebar.

Top End Style — Select a style for the top end of a vertical line control. Valid options are: None, Single-line T, Double-line T, Single-line Cross, Double-line Cross, Single-line Right Turn, Double-line Right Turn, Single-line Left Turn, and Double-line Left Turn. The default is None.

Bottom End Style — Select a style for the bottom end of a vertical line control. Valid options are: None, Single-line T, Double-line T, Single-line Cross, Double-line Cross, Single-line Right Turn, Double-line Right Turn, Single-line Left Turn, and Double-line Left Turn. The default is None.

### Horizontal Lines

Choosing the horizontal line icon from the tool palette allows you to draw horizontal lines of varying lengths on your form.

Line Foreground Color — Enter or choose a color from the combo box for the horizontal line. The system default value is Yellow. Choosing <Auto> means that the corresponding customized default setting for lines is used.

Line Background Color — Enter or choose a color from the combo box for the horizontal line's background area. The system default value is Blue. Choosing <Auto> means that the corresponding customized default setting for lines is used.

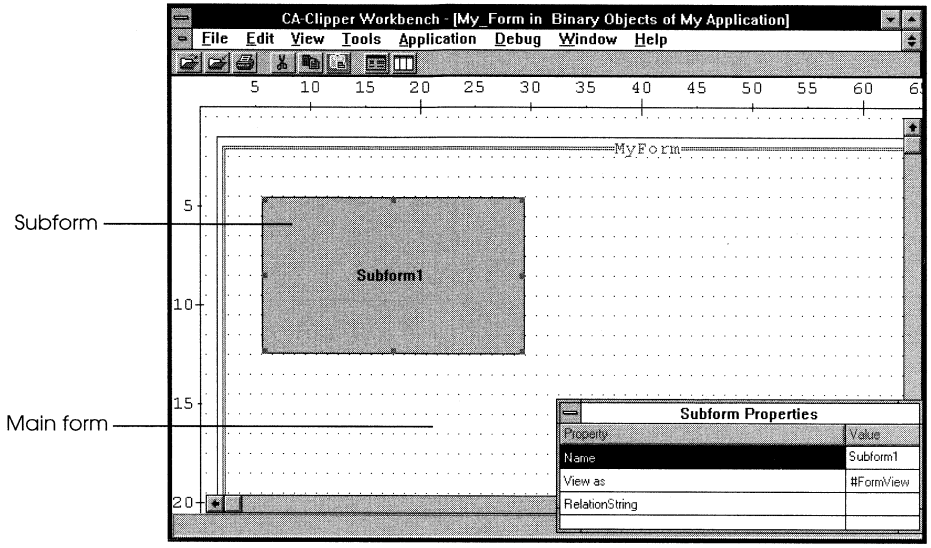
Line Style — Select a line style for the horizontal line control. Valid options are: Doublebar and Singlebar. The system default is Doublebar.

Left End Style — Select a style for the left end of a horizontal line control. Valid options are: None, Single-line T, Double-line T, Single-line Cross, Double-line Cross, Single-line Right Turn, Double-line Right Turn, Single-line Left Turn, and Double-line Left Turn. The default is None.

Right End Style — Select a style for the right end of a horizontal line control. Valid options are: None, Single-line T, Double-line T, Single-line Cross, Double-line Cross, Single-line Right Turn, Double-line Right Turn, Single-line Left Turn, and Double-line Left Turn. The default is None.

## Subforms

When you select the subform icon from the tool palette, a secondary data form, or *subform*, is placed on your main form:




A subform allows you to manually link data servers in a master-detail relationship.

**Note:** See Using Auto Layout later in this chapter for information about the Form Editor's Auto Layout feature, which automatically creates data forms and subforms with fixed text captions and data entry fields for every available field in the associated data server(s).

The following properties are required for subforms:

View As

Every data form can operate in either *form view*, displaying multiple fields for a single record as fixed text and fields, or *browse view*, using a spreadsheet-like data browser that displays multiple fields and records in a table in the data window.

Choose the default view to use when the data form is initially accessed. The valid choices are #FormView and #BrowseView. The default setting is #FormView.

**Note:** See the Browse and Form View section later in this chapter for more detailed information.

RelationString

The *relation string* is the common field used to relate the data server associated with the detail form to the data server associated with the main data form.

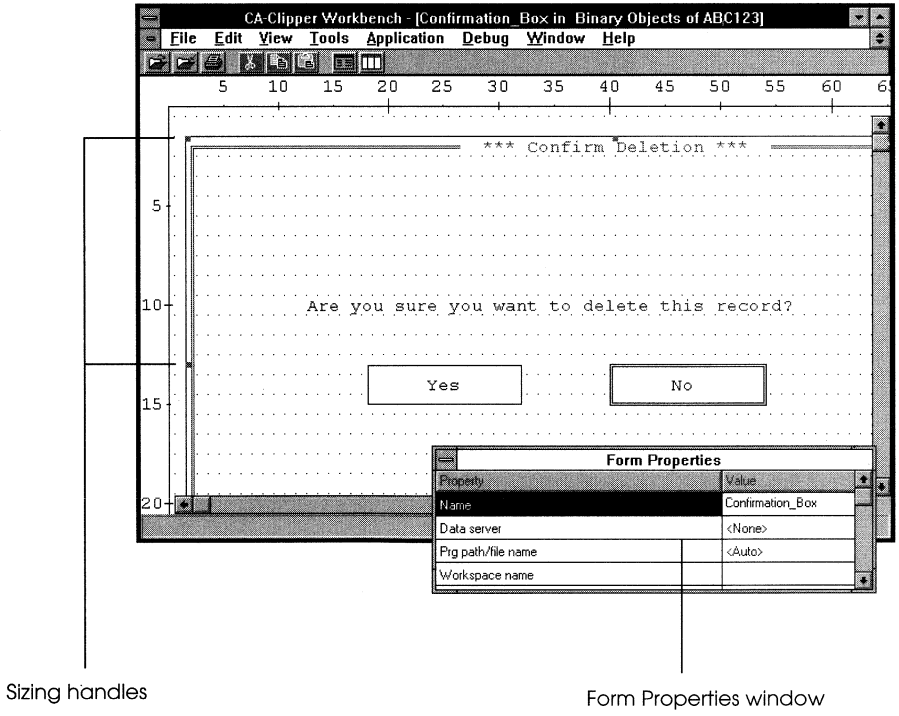
**Note:** When you select a subform control, it shows only the Name, View As, and RelationString properties. If you double-click on the subform control, however, a new copy of the Form Editor is launched in which you can view and change all the standard data form properties described earlier under Specifying Form Properties.

# Modifying a Form

After you have defined a form, you can modify it by changing its properties or the properties of any control associated with it. You can also move and size a form or any of its controls, and you can cut, copy, paste, and delete individual form controls.

## Editing Form Properties

To make any changes to a form, you must first select it by clicking any spot on the form that is not currently occupied. For example, you can click on the title bar or a blank space between two controls. When a form is selected, its perimeter is marked with several handles that are used for sizing, and the Properties window changes to show the form's properties:



### Changing Form Properties

To change any property associated with a selected form, click the property that you want to change in the Properties window and specify a new value. See Specifying Form Properties for more information.

### Sizing a Form

To change the size of a selected form:

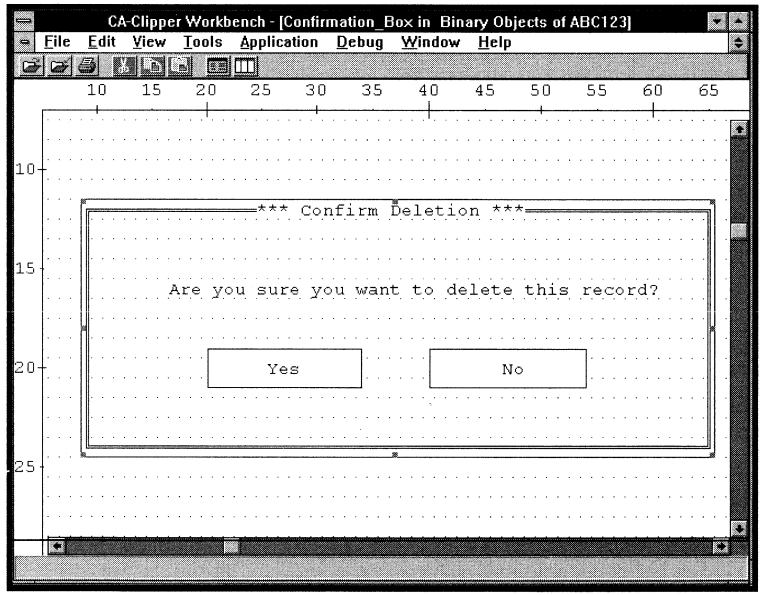
1. Place the mouse pointer on one of its handles.

   The mouse pointer will change to a double arrow.

2. Press the left mouse button and hold it down.

3. Drag the mouse to change the form to the desired size, and release the mouse button.

For example, below is our resized Confirm Deletion dialog box:

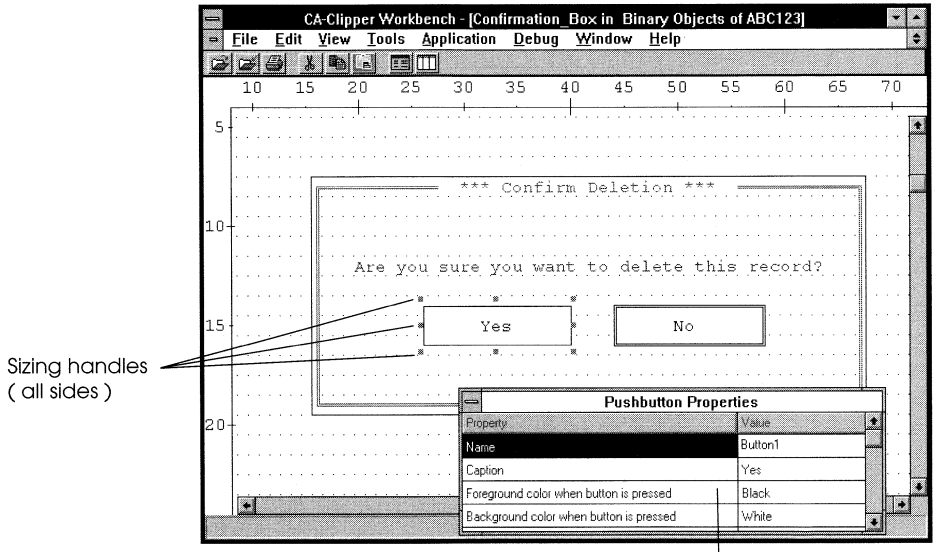


## Editing Controls

To make any changes to a control, you must first select the control by clicking on it.

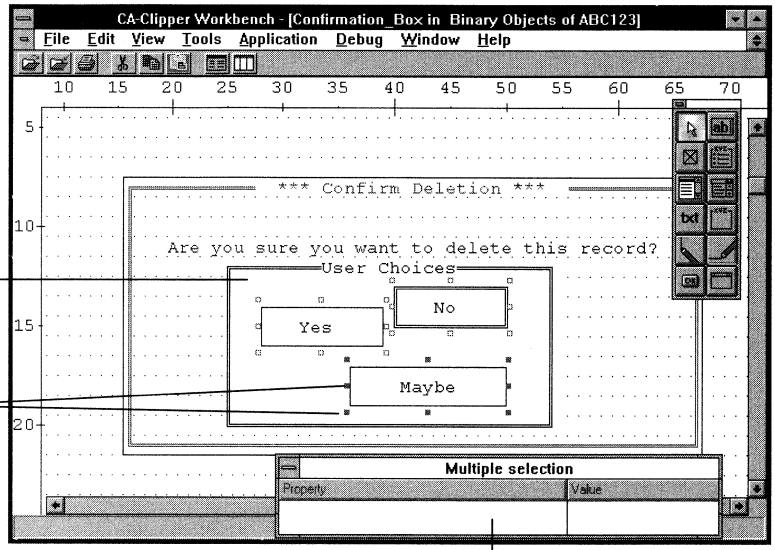
### Selecting a Single Control

To select a single control, place the mouse pointer on the control and click the left mouse button. Its perimeter is marked with several handles that are used for sizing and moving, and the Properties window changes to show the control's properties:



## Selecting Multiple Controls

To select more than one control, click on the first control, then hold down the Ctrl key as you click on the other controls. You can also drag a frame around the controls you want to select (hold down the left mouse button and drag until a square appears around the group of controls you want to select). A multiple selection might look like this:




**Note:** You cannot change properties for a multiple selection; this is indicated by an *empty* Properties window.

The last control selected has a different appearance; it has gray selection handles while the other controls have white selection handles. The last control selected plays a key role when you are using one of the Edit Arrange menu choices.

For example, if you have three controls, the first two controls will be modified to imitate one or more of the attributes of the last one. For example, the Align Left command would move the top three controls so their left borders align with the left border of the bottom control.

### Changing Control Properties

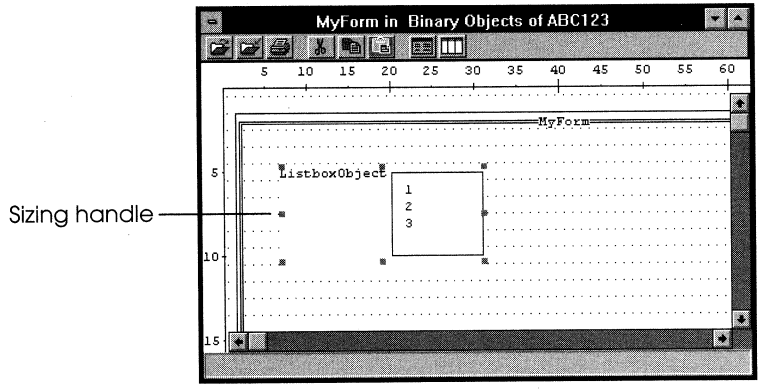
To change any property associated with a selected control, click the property that you want to change in the Properties window and specify a new value. See Specifying Control Properties for more information.

### Sizing a Control

All controls can be resized. Some can only be resized horizontally, such as single-line edit controls, check boxes, and push buttons. Horizontal lines can only be resized horizontally and vertical lines only vertically. Others like list and combo boxes can be resized in both directions, and fixed text controls adjust automatically to the length of the specified caption.

For example, to change the size of a list box:

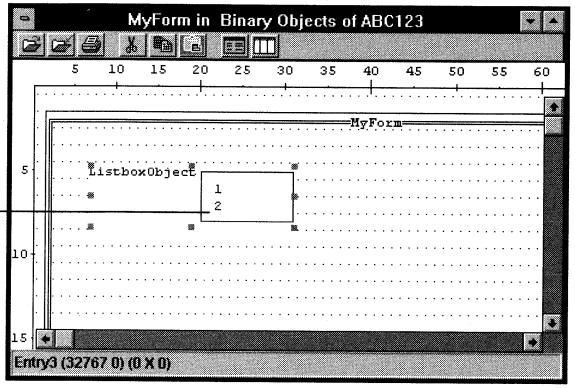
1. Place the mouse pointer on one of its handles. For example:




   The mouse pointer will change to a double arrow.

2. Press the left mouse button and hold it down.

3. Drag the mouse to change the control to the desired size, and release the mouse button. The result is:




## Moving a Control

To move a selected control to a new location in the form:

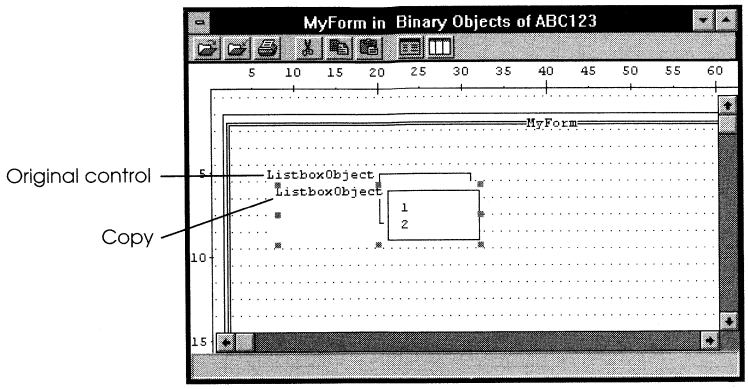
1. Place the mouse pointer anywhere on the control.
2. Press the left mouse button and hold it down.
3. Drag the mouse to move the control to the desired location, and release the mouse button.

## Copying a Control

To copy a selected control to a new location in the form:

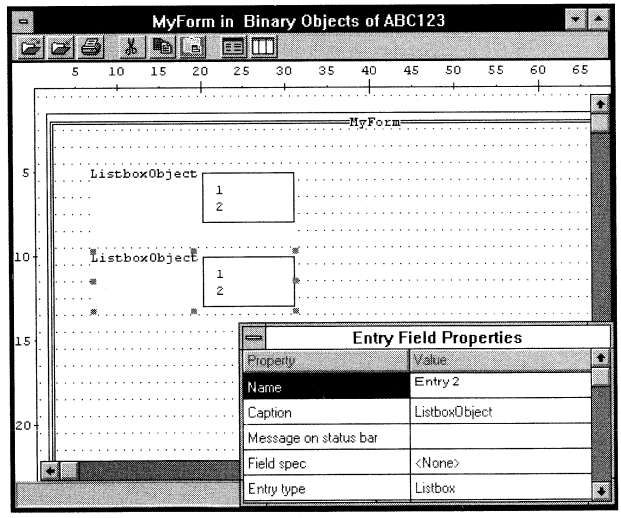
1. Place the mouse pointer anywhere on the control.
2. Choose the Copy toolbar button.
3. Choose the Paste toolbar button.

The new control will be pasted on top of the original. For example:




4. Click on the new control and move it to the desired location.

   The Properties window changes, reflecting the fact that the new control now has focus:




   Note that the copy has the same caption as the original, but a different name (for example, "Entry2" if the original was named "Entry1").

5. Enter a new caption for the new control, and specify any other properties for this control.

### Deleting a Control

To delete a control from the form:

1. Select the control by clicking on it.
2. Press the Delete key.

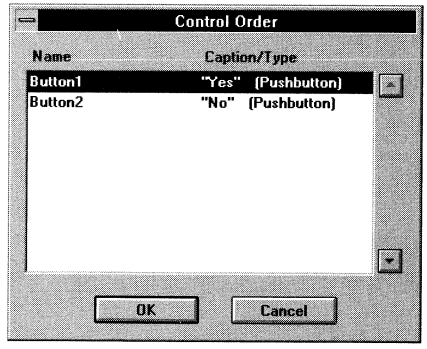
   Alternatively, use the Edit Delete menu command to remove the control.

**Tip:** You can also use the Cut toolbar button, if you plan to paste the control on this or another form.

## Changing Tab Order by Reordering Controls

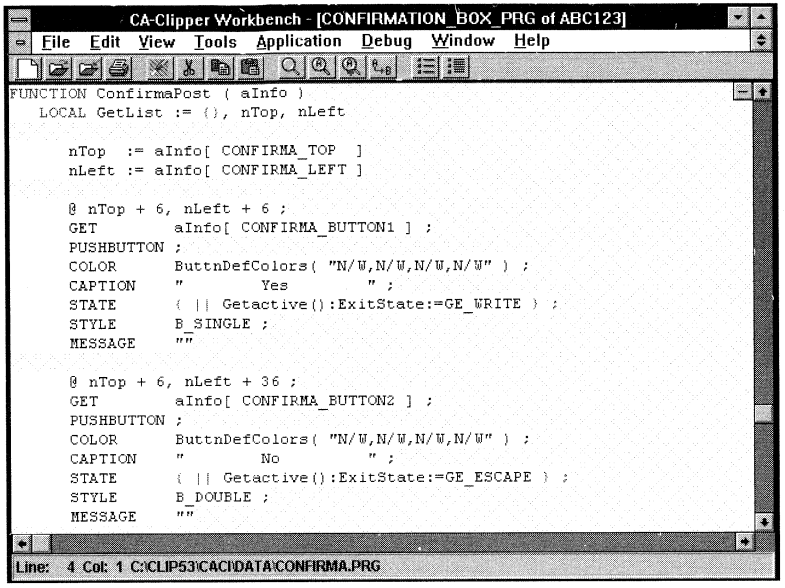
The Edit Control Order menu command allows you to change the cursor tabbing order for a form's controls by reordering the controls in the source code.

To view the cursor tabbing order for our Confirmation_Box form entity, for example, select the Control Order command while in the appropriate Form Editor. The Control Order dialog box appears:




This dialog box displays the names of all the controls, as well as their captions and control types, in *tab stop* order, as they appear in the source code.

To verify this, return to the module browser and select the .PRG module for the form. You can then access the module's source code using the Edit All Source toolbar button:



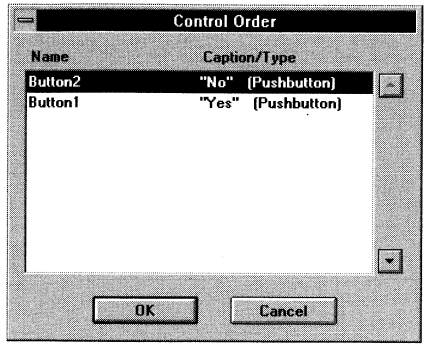

Tab stops are synonymous with the order of the GETs in the *GetList*. They are based upon the coordinates for the upper-left corners of the controls.

As this tabbing order may not be user-friendly, you may want to rearrange the controls to improve the logical flow of the cursor movement.

For example, perhaps you would rather have the cursor initially go to the "No" push button to avoid accidental deletion. Let's look at an example. To change the current tabbing order for our Confirmation_Box form entity:

1. Return to the Form Editor and choose the Edit Control Order command.
2. Select "Button2" in the Name list box.

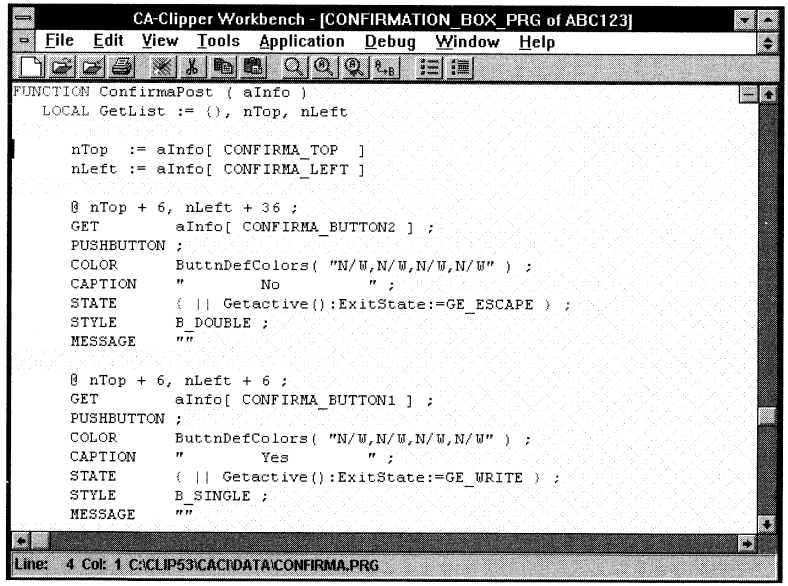
3. Click on the Up arrow twice to move it to the top of the Name list box:




4. Select OK.

**Note:** This command affects only the order in which the cursor moves from control to control within a form. It does *not* alter the controls' actual positions on the form.

The source code would then reflect these changes for the reordered controls:



# Data Forms

A data form is a form or data entry screen for a *data server*. The data form is *data-aware*—it "knows" about the data with which it is intended to operate via properties that you specify for each *data entry control* in the data form.

The Form Editor allows you to create data forms using either of two techniques:

- Customizing a form by manually placing controls on a blank form, choosing a data server, and specifying properties for those controls.
- Using the Auto Layout feature to choose a data server (or two related data servers) and letting the Workbench automatically place controls on the form whose properties are based on the data server.

Using either technique, the key to designing a data form is in linking individual controls to *field specs* defined in the data server. Because the field specs store information designed specifically for use with data forms (such as captions, status bar messages, picture formatting, and validation rules), these links are quite useful. Data forms automatically inherit and use the property values from the field specs, eliminating the need for you to specify this information repeatedly.

## Creating a Data Form

To create a form that is linked to a data server, you need to perform the following steps:

1. Start the Form Editor.

   The Form Editor dialog box appears. (See Creating a Form for details about this dialog box.)

2. Enter a name for the form entity (for example, **Customers**).

3. Associate a data server with the data form, using one of two methods: *Auto Layout* and *Properties*. For this example, we used the CUSTOMER_BASE data server.

   These methods are described next in Associating Data Servers.

4. Choose the Save toolbar button to save the form.

**Note:** This last step can be repeated whenever you make changes to the form and want to save your work without closing the Form Editor.

## Associating Data Servers

You can associate a data server with a data form in two ways:

- Using the Properties window

  Choose a data server using the Properties window and add your own controls manually one by one, creating a customized data form.

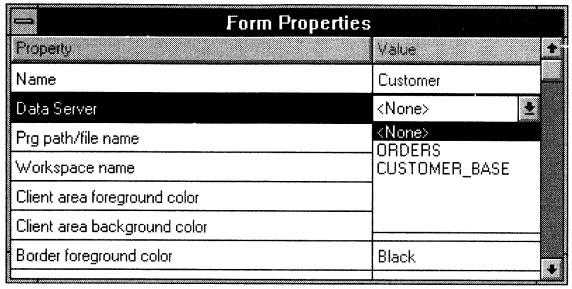
- Using the Auto Layout feature

  Choose the Auto Layout toolbar button to specify a data server (or two related data servers), creating a predefined data form based on the data server(s) definition. You are then free to move the predefined controls around within the form and modify them as you see fit.

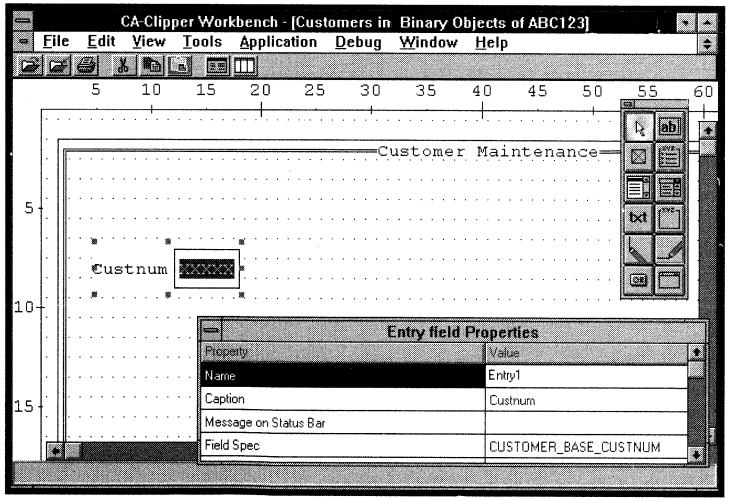
See Customizing a Data Form and Using Auto Layout, respectively, for more detailed information about linking data servers with a data form.

## Customizing a Data Form

To create a customized data form manually, you must first choose a data server using the Properties window. To do this, click on the Data Server property's arrow button, and select a data server from a drop-down list of all data servers defined for the current application. For example:




Then you can customize the blank form manually, selecting controls from the tool palette (or using the Select from Palette commands from the Edit menu), placing them in the desired locations on the form, and specifying properties for them, such as field spec, size, color, and caption. For example:




See the Placing Controls and Specifying Control Properties sections earlier in this chapter for details.

## Using Auto Layout

Using the Form Editor's Auto Layout feature is much easier than creating a data form "from scratch." At the touch of a button, Auto Layout automatically creates a fixed text caption and data entry control for every available field in the associated data server(s).

> **Tip:** If you have an Xbase database on disk that is not yet known to the repository, simply bring up the DB Server Editor using the appropriate command in the Tool menu, import the database structure, save it in the repository, and switch back to the Form Editor. See "Defining Data Servers and Field Specs" for more information on how to do this and on data servers and field specs, in general.

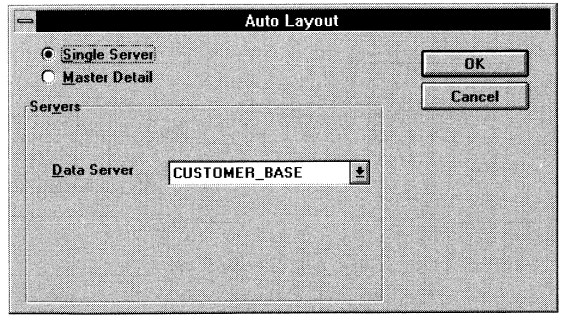
The Auto Layout feature presents you with two options for defining the type of data server link, *Single Server* or *Master Detail*.

### Single Server Option

The Single Server option is used to link a data form to a single data server. Our example below walks you through the creation of a customer data entry form for maintaining customer records:

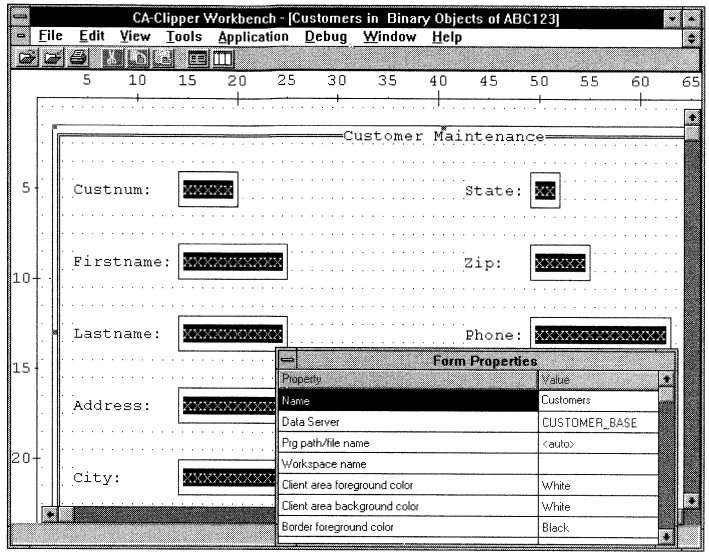
1. Click on the Auto Layout toolbar button.

   The Auto Layout dialog box appears:




2. From the Data Server drop-down list box, select the desired data server (for example, CUSTOMER_BASE).
3. Choose OK.

The result is a predefined form with data entry controls for each of the fields defined for the specified data server:




Note that the value cell for the Data Server property has been filled in automatically by the system.

**Tip:** You can edit or delete any of these controls. Additionally, you can rearrange their placement or manually add other controls using the tool palette. See Modifying a Form earlier in this chapter for more information.

Master Detail Option

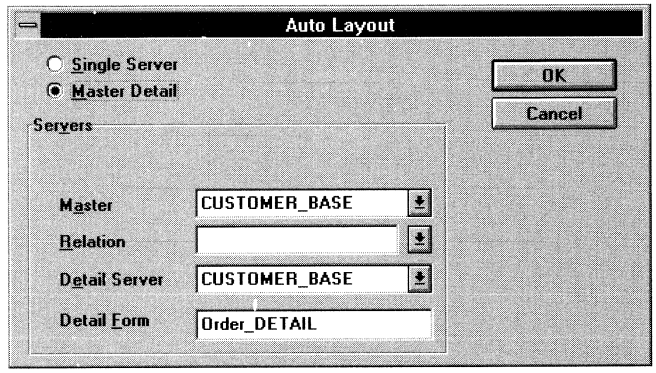
The Master Detail option allows you to link a form to two data servers in a master-detail relationship. In this case, a data form is created for the master server with a "nested" subform for the detail server.

Nesting data forms and subforms is a convenient way to update several forms that share common fields, as they are linked to the same data server(s). Thus, changes are automatically propagated between the forms regardless of where the changes originated. For example, if a user types in one form, that entry is immediately reflected in the other form. The code generated for both the form and the data server is designed to allow multiple instantiation.

We will now use the Form Editor's Auto Layout feature to create a new data form called "Order Entry." This data form will link order entry information to customer records in a master-detail relationship.

1. Create a new form, following the basic steps described earlier in the Creating a Form section, and name it Order Entry.
2. Click on the Auto Layout toolbar button.
3. Select the Master Detail radio button.

   The Auto Layout dialog box changes slightly, as shown below:




4. Select a data server from the Master drop-down list box (for example, ORDERS).
5. Select a field from the Relation drop-down list box (for example, CUSTNUM).

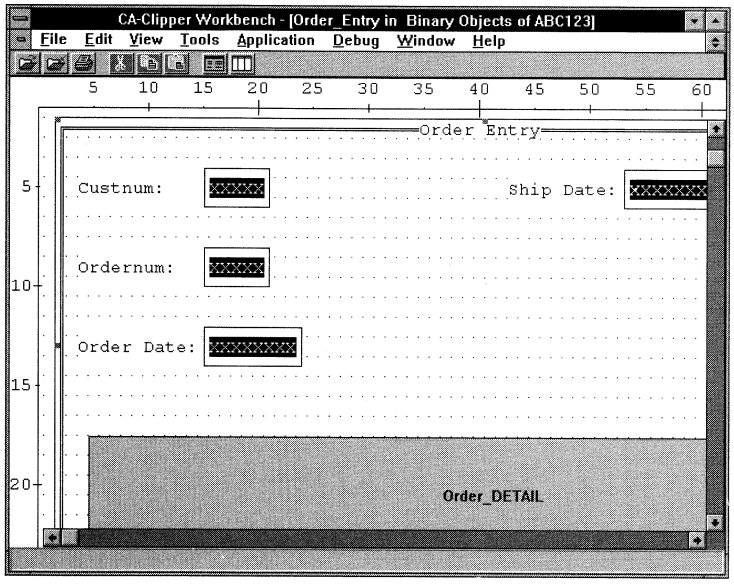
   The field list presented is taken from the master data server. The field that you choose will be used to perform a lookup operation in the detail server based on its controlling order, and all detail records with keys matching the field in the master record will be displayed in the detail form.
6. Select another data server from the Detail Server drop-down list box (for example, CUSTOMER_BASE).

7. Note that the system automatically generates a name for the detail form and displays it in the Detail Form edit control. (This same name will be displayed in the Properties window for the subform control.)

   If you like, enter a new name for the detail form.

8. Choose OK.

Using ORDERS as the master server and CUSTOMER_BASE as the detail server in our sample application, the following data form appears:




Since the ORDERS data server was designated as the master server, its fields are placed on the main data form as single-line edit controls with fixed text captions. Additionally, a detail form is placed on the data form beneath these fields for CUSTOMER_BASE. This detail form, or *subform control*, is actually a nested data form.

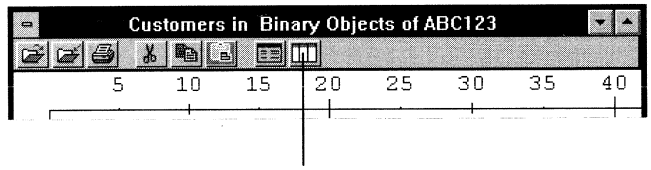
You can now edit the following: the data form itself, the controls for each of the data form's fields, or the detail form simply by clicking in the appropriate area. The appropriate Properties window appears, allowing you to modify the specified form or control properties.

See Modifying a Form earlier in this chapter for more details on editing form properties and controls.

## Browse and Form View

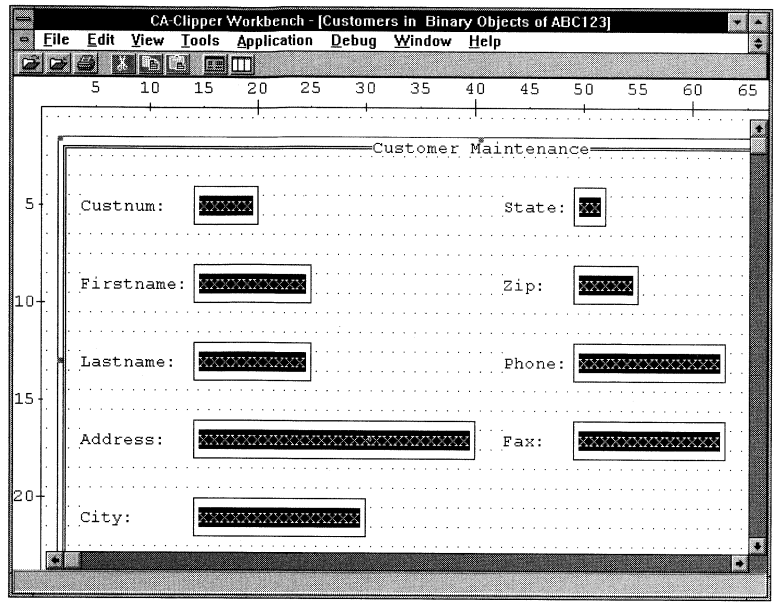
So far, you have been using the Form Editor to paint a data form using *form view*, but the editor has also created a corresponding *browse view* that you can work with if you choose. Browse view displays multiple records for each field, always maintaining one record as the "current" row, reflecting the data server's current position. The form view, on the other hand, displays only one record—the current one—at any given time.

To change from one view to another, use the Browse/Form View toolbar button:

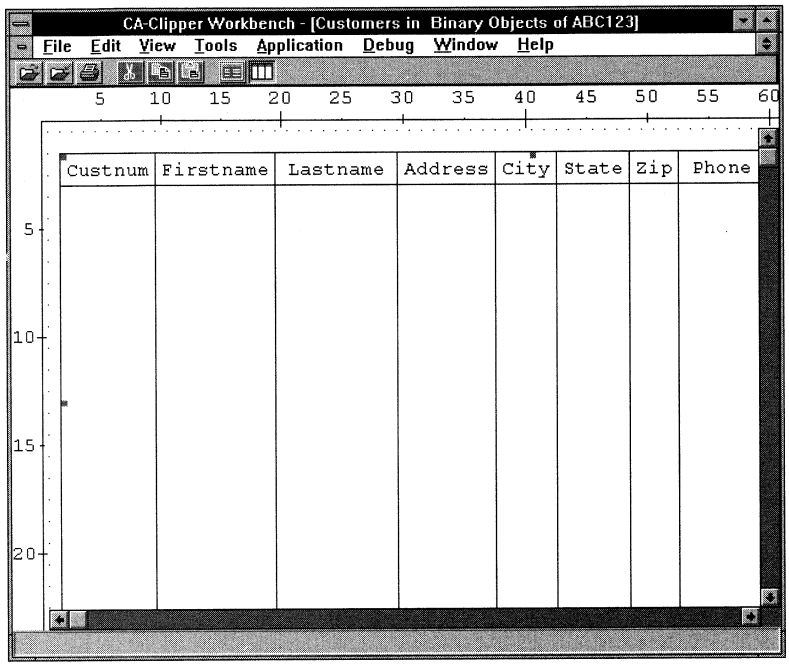



Browse/Form View toolbar button

For example, below is our sample single server form, Customers, in form view:



After clicking the Browse/Form View toolbar button, here is the same form in browse view:

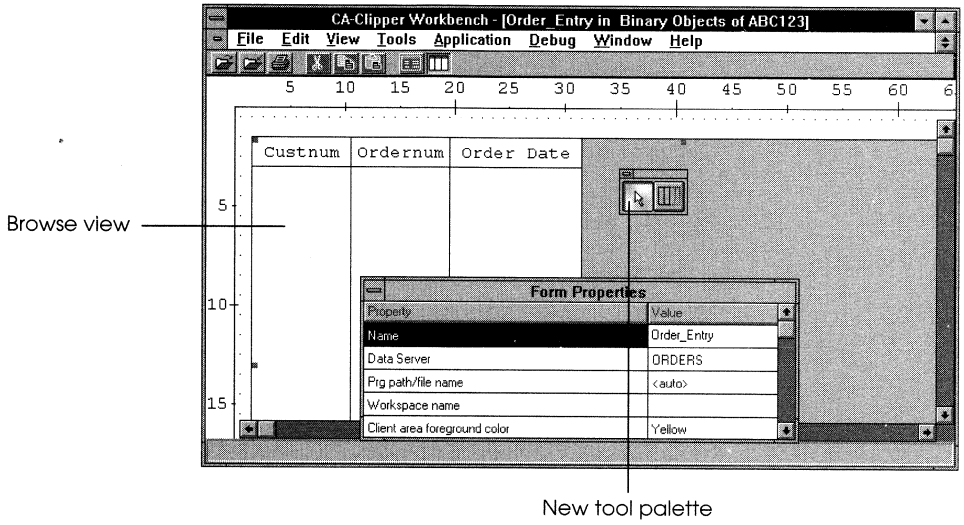



**Note:** Changing the view associated with a data form affects only the form template display and the operations you can perform within the Form Editor.

**Tip:** You can start out by designing the form in browse view, if you like. Similar to the way in which the Form Editor creates a default browse view for forms designed using form view, it will create a default form view for forms designed using browse view.

## Working in Browse View

For example, our sample Order Entry data form was created initially using ORDERS as the master server and CUSTOMER_BASE as the detail server (see Master Detail Option in the Using Auto Layout section earlier in this chapter). To change the data form so that it displays in browse view, click the Browse/Form View toolbar button. Note that the new tool palette icon that allows you to place additional columns in the table:




Sizing Columns

In this view, you can size a column as follows:

1. Place the mouse pointer on either the left or right edge of the column heading.

   A vertical bar, with left and right arrows, appears when the column is ready for sizing.

2. Press the left mouse button and hold it down.

3. Drag the border to the left or right until the column is the desired size, and release the mouse button.

### Moving Columns

You can also move a column:

1. Place the mouse pointer anywhere within the column heading.
2. Press the left mouse button and hold it down.
3. Drag the column to the new location, and release the mouse button.

   When you are moving a column, you will see a double-sided arrow with a column icon in between to indicate that the column is being moved.

### Cut, Copy, and Paste

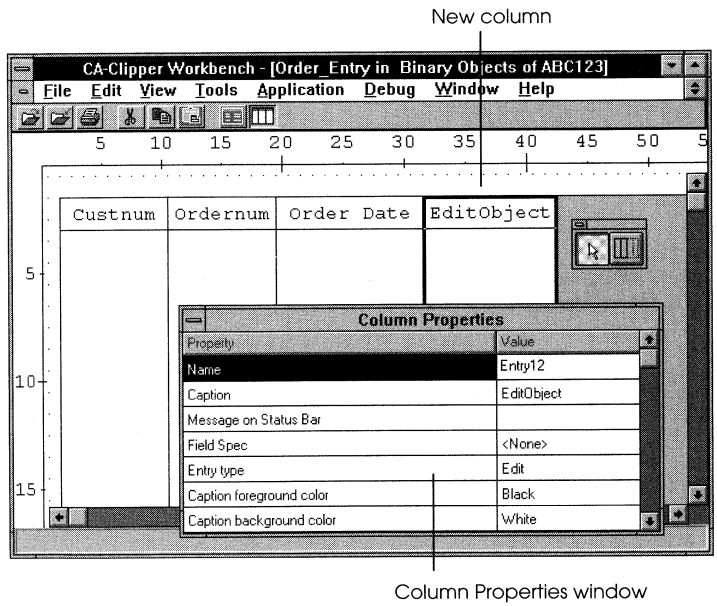
If you like, you can use the Cut, Copy, and Paste toolbar buttons to manipulate columns using the Clipboard in the standard way.

### Inserting Columns

Finally, you can add columns in this view using the column icon in the new tool palette. For example, to insert another column in our sample data form, Order Entry:

1. Click on the column icon in the tool palette and drag-and-drop the control onto the data form.

   A column is added as a single-line edit control, and an appropriate Properties window appears:



2. Specify the values for the new single-line edit control.

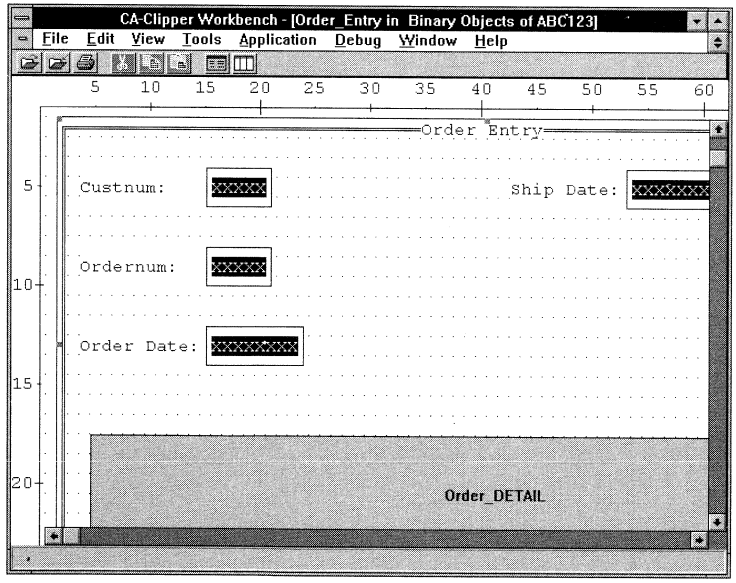
   The descriptions for the column's properties are the same as the other EntryField controls. See Specifying Control Properties earlier in this chapter for more details.

3. Choose Save.

Any changes that you make while in browse view are saved as part of the data form and will be apparent the next time you switch to the browse view in the Form Editor and when the user switches to browse view at runtime.

## Switching to Form View

You can return the data form to the original *form view* by clicking the Browse/Form View toolbar button once again. For example:

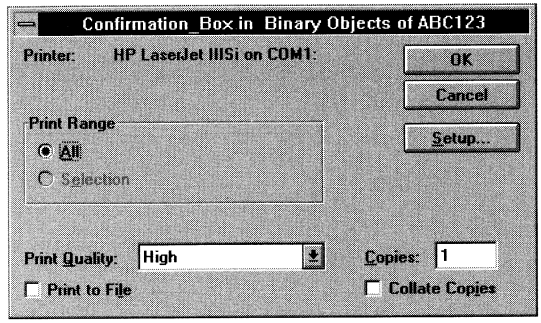


# Printing a Form

To print a screen shot of the form you are designing:

1. Choose the Print toolbar button.

   A standard Print dialog box appears:




2. Specify your printing options.
3. Choose OK.

   A copy of the form itself—and not the Form Editor containing it—is printed.

For descriptions of the available standard printing options, see Printing an Application List in the "Browsing Applications, Modules, and Entities" chapter.

# Using the Form in an Application

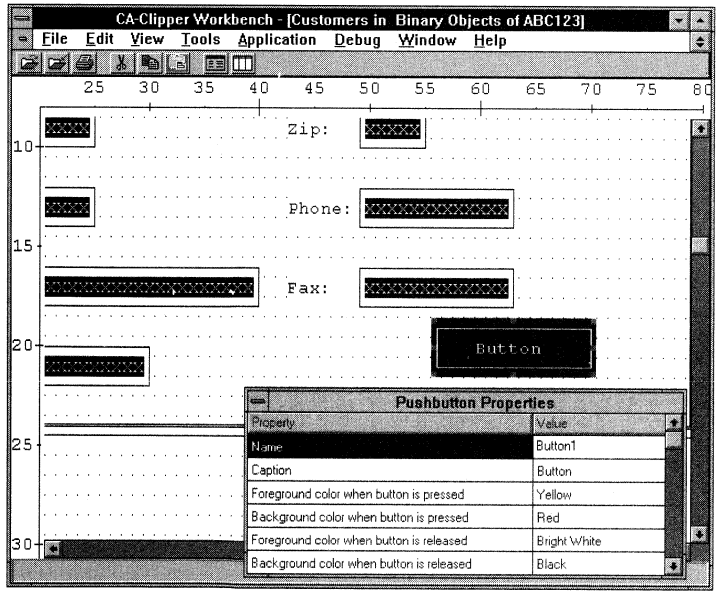
Once you have created a form using the Form Editor, you need to add one of two basic actions that will activate the form for the end user:

- Clicking a push button
- Using a menu command (or its equivalent toolbar selection)

Associating a form with either of these actions is relatively easy. Simply enter a function call in the value cell for the push button's Custom Function Call property or the menu item's Action property. Next, define the function in the .PRG module for the called form. Then, when the end user selects the push button or menu item, the form will be displayed.

For example, to link our sample Confirm Deletion dialog box to our sample Customer Maintenance form:

1. Place a push button control on the form. The result is:



2. Replace "Button" with **Delete** in the Caption value cell.
3. Select CustomFunction from the Action Type When Button Pressed drop-down list box.
4. Enter a function call in the Custom Function Call value cell (for example, **DelProc()**).
5. Choose Save.
6. Now, open the Confirmation_Box program module in the Source Code Editor and manually enter the code for the DelProc() function.

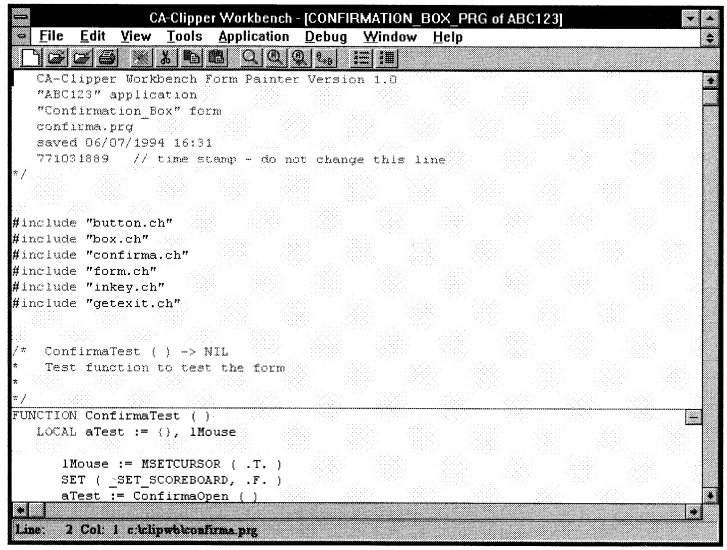
   **Note:** The code for the function call should be entered in a *different* module as we have just done here in our example. Otherwise, the added function will be deleted *if* you later modify the form using the Form Editor, as the Form Editor will regenerate the .PRG file.
7. Choose Save.

When the Delete push button is clicked in the final application, the Confirm Deletion dialog box will appear and prompt the end user to confirm that the current customer record should be deleted.

**Note:** See "Using the Menu Editor" for more information on associating a form with a menu selection.

# Generating Code

When you save a form, the Form Editor automatically generates a form entity, which is placed in the Binary Objects module, as well as separate program source file (.PRG) and header file (.CH) modules. You can subsequently view and/or edit the source code for your form in the Source Code Editor. For example, below is the source code for our very first form, the Confirm Deletion dialog box:




***Important!*** *If the source code for a form is edited manually, the changes will be lost* ***if*** *the form is later modified using the Form Editor, as the Form Editor in that case regenerates the .PRG file.*

## A Closer Look at the Source Code

Note that the Form Editor automatically generates predefined functions and static functions for any form entity. For example, if you created a form called XYZ, the following functions would be defined for it:

| Function | Purpose |
|---|---|
| XYZTest () | Allows testing of the form display and data entry. |
| XYZOpen () | Opens the form, saves what was previously at that location on the screen, and displays it (with empty values for the Get system variables).<br>Returns an array, which contains information that defines the form's properties and behaviors. It needs to be passed to all other generated functions. XYZ.CH contains constants that you use to access and modify the array. |
| XYZClose (*aInfo*) | Closes the form, and restores what was previously on the screen before the XYZOpen() function call. |
| XYZEmpty (*aInfo*) | Resets the variables of the Get system to empty contents (spaces or 0 values). |
| XYZInit (*aInfo*) | Resets the variables of the Get system to the default values defined in the Form Editor. |
| XYZAppend (*aInfo*) | Appends a record to the active workspace with the contents currently in the variables of the Get system. |
| XYZDataToDb (*aInfo*) | Copies the contents of the variables of the Get system into the current record of the work area. |

*Continued*

*Continued*

| Function | Purpose |
|---|---|
| XYZLoad (*aInfo*) | Copies the contents of the current record of the work area into the variables of the Get system. |
| XYZSave (*aInfo*) | Stores the contents of the variables of the Get system into the current record of the workspace. |
| XYZEdit (*aInfo, oMenu*) | Puts the form in the edit mode. The first parameter is the same as for the other functions (the array returned by XYZOpen() ); the second is an optional menu object for that menu to be active during the ReadModal() function call. |
| XYZViewForm (*aInfo*) | Displays the form with the current values of the Get system variables, including the scrollable portions of the form, such as subforms in browse mode. |
| XYZScreen (*aInfo, nMode*) | The second parameter may be either:<br>FORM_DISPLAYWINDOW<br>which displays the form window and all statics<br>-OR-<br>FORM_DISPLAYDATA<br>which displays the variable parts of the form (Get system variables). |
| XYZPost (*aInfo*) | Creates and displays the form's *GetList*. |
| XYZStatics (*aInfo*) | Displays the static portions of the form—typically the border text and lines. |
| XYZValid (*aInfo, nParam*) | Used internally by other functions for some specific valid clauses, including performing the search in a linked file in subforms. |

*Continued*

*Continued*

| Function | Purpose |
|---|---|
| XYZTOpen () | Same as XYZOpen(), but opens the form in browse view instead of form view. This function returns an array, which is passed to all other generated functions. |
| XYZTDisplay (*aInfo*) | Displays the form in browse view, including stabilization, and returns without waiting for a key press. |
| XYZTManage (*aInfo, bAppend, bUpdate, bDelete*) | Allows you to manipulate the browse (moving, scrolling, etc.) with the following additional parameters:<br>*bAppend*<br>Set to true (.T.) to allow the user to append database records at the end of the browse.<br>*bUpdate*<br>Set to true (.T.) to allow the user to update database records in the browse.<br>*bDelete*<br>Set to true (.T.) to allow the user to delete database records in the browse. |
| FUNCTION XYZTClose (*aInfo*) | Closes the browse view, and restores what was previously on the screen before the XYZTOpen() call. |

**Note:** If the form is not attached to a data server, the following functions are *not* generated: XYZAppend(), XYZDataToDb(), XYZLoad(), XYZSave(), and any browse view function.

# Chapter 5
# Using the Menu Editor

## In This Chapter

The Menu Editor makes it easy to design custom menus, providing a host of useful properties (such as associating an action code with a menu item) and immediate visual feedback by previewing menus as you design them. You can quickly add standard, predefined File, Edit, View, and Help menus at the click of a button.

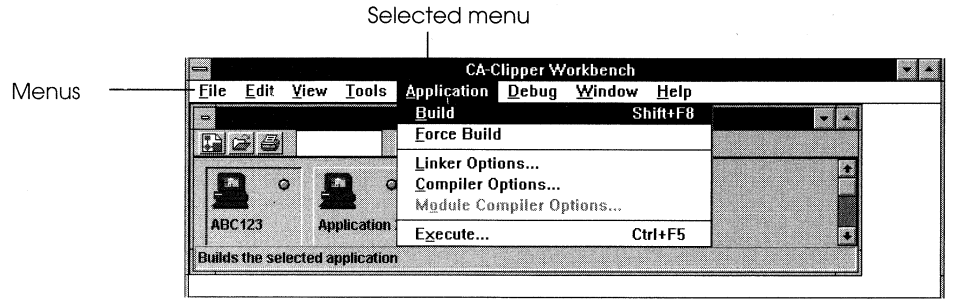
The Menu Editor will create a binary entity in the Binary Objects module of the current application, and generate the associated source file (.PRG) in a separate source file module.

This chapter describes how to use the Menu Editor, including how to:

- Create a custom menu structure
- Add predefined, standard menus at the touch of a button
- Modify an existing menu structure
- Print using the Menu Editor
- Use the menu in your application

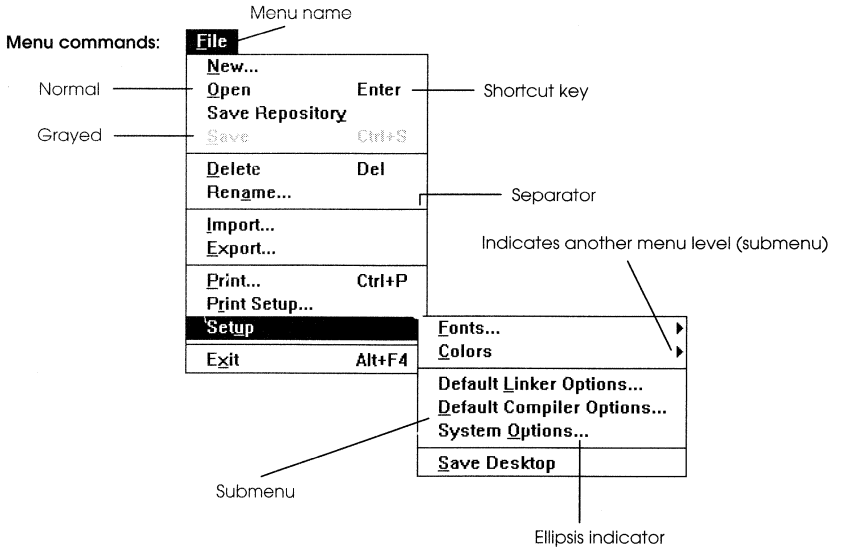
# Menu Terms

A *menu* is a user interface element that presents a list of choices. Menus appear in a window's *menu bar*; for example, below is the Workbench's menu bar:




Typically, when a menu is selected in a menu bar (with the mouse or keyboard), it displays a menu of *items* (menu commands, separators, other menus, etc.). However, it can also immediately execute an action (for example, an Exit menu that quits the application when selected).

The various types of menu items that you can include in a menu's structure are illustrated in the example below:

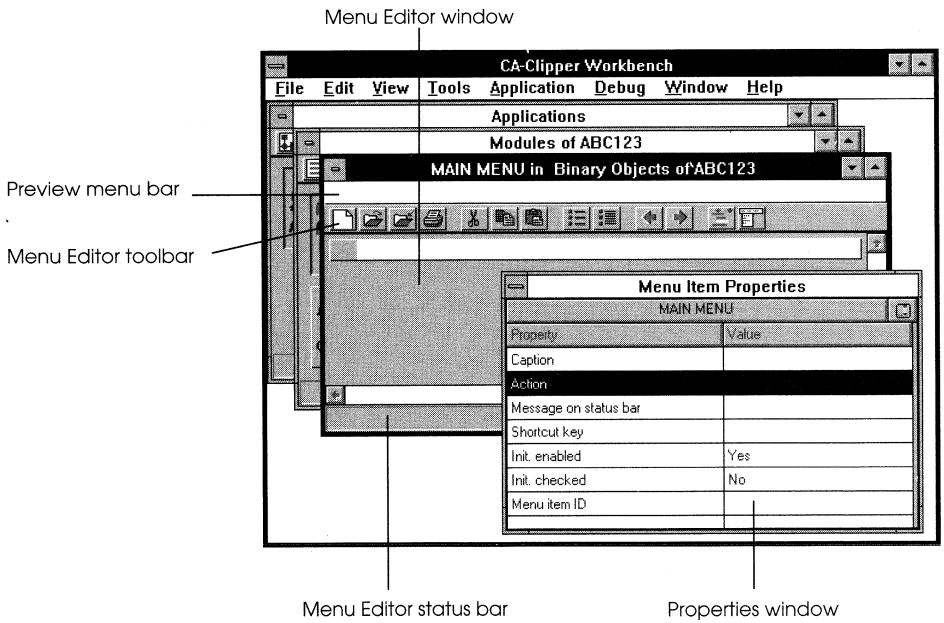


# Workspace Overview

The Menu Editor is the primary workspace in the Workbench for creating, viewing, and modifying menus. When you are in the Menu Editor, you can:

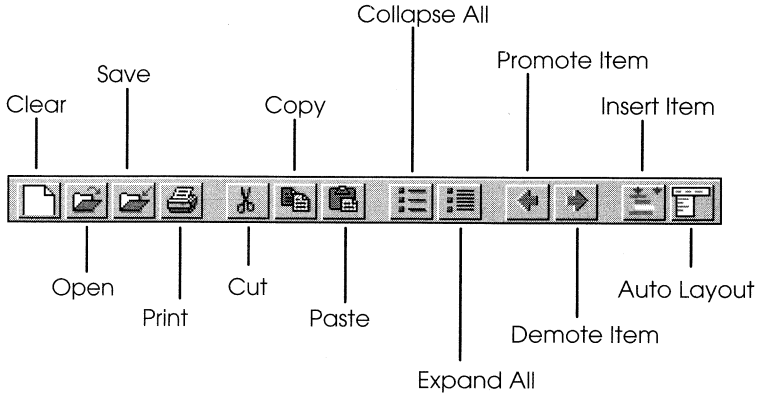
- Create, modify, preview, and print a menu structure
- Define properties for the menu structure and its entries
- Access other browsers and editors

The Menu Editor has its own toolbar, status bar, an associated Properties window, and an area for previewing defined menus. In the example below, a simple accounting application named "ABC123" is loaded for a new menu:



## The Toolbar

The Menu Editor toolbar contains the following buttons:

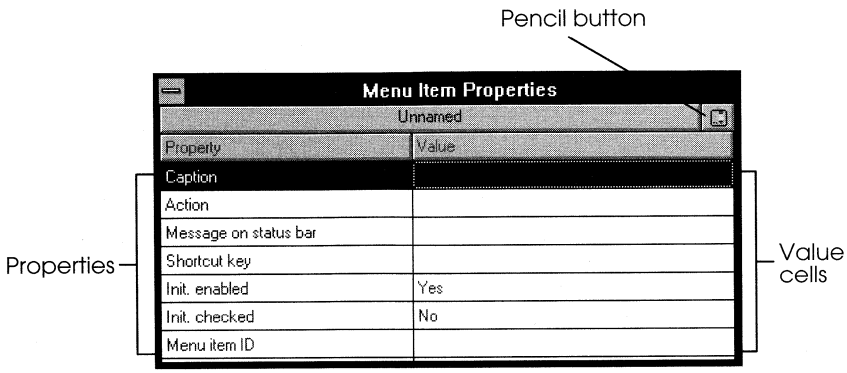



The buttons displayed here are discussed in this chapter.

**Tip:** For a quick description of these toolbar buttons, look at the Menu Editor status bar as the mouse pointer passes over the buttons. Also see your online Help reference.

## The Properties Window

When the Menu Editor is launched, a *floating* Properties window opens automatically. Initially, this window allows you to specify properties for the current menu item:




The *Property* column lists all properties that can be specified for the currently selected entry, and the *Value* column contains the corresponding cells where you specify a value. For example, you can specify text that should appear in the status bar when the menu item is selected, or an initial state for a control (such as checked or unchecked).

At the touch of a button, you can change the Properties window to reflect the properties of the menu structure, instead of its individual entries.

In either case, the window behaves in the same manner. To do this, simply highlight a property by clicking on it, then use one of the following techniques for specifying its value:

- Click on the *Pencil button* and
  - Enter a new value by typing directly into a single-line edit control
  - Choose a new value from a drop-down list
  - Fill in a corresponding dialog box
- Choose one of two possible values using a plus (+) or minus (–) button

The Properties window is discussed in greater detail in the Specifying Menu Properties and Specifying Menu Item Properties sections later in this chapter.

**Note:** The Properties window always remains open until explicitly closed (using the system menu) or until its owner, the Menu Editor window, is closed. If explicitly closed, you can reopen it at any time using the Open Property Window command on the Window menu. Also, the Properties window is affected by actions to its owner window. For example, if the owner window is minimized to an icon, the Properties window will also be minimized.

### The Preview Menu Bar

As you define a menu structure in the Menu Editor, each new entry is added to a prototypical menu bar, called the *preview menu bar*, at the top of the Menu Editor window. The preview menu bar is partially operational—showing description messages in the status bar and allowing submenus to be pulled down—but nothing actually happens when you make a selection. Its purpose is to give you visual feedback while you are designing a menu structure.

See Previewing the Menu Bar later in this chapter for details.

# Defining a Menu

Now that you have a general overview of the Menu Editor workspace, you are ready to use it to define a new menu structure. In this section, you will learn how to:

- Create a menu structure by manually adding entries
- Specify a menu hierarchy
- Specify properties for the menu structure and its entries

In later sections you will learn how to create a predefined menu using the Auto Layout feature, and to modify an existing menu structure.

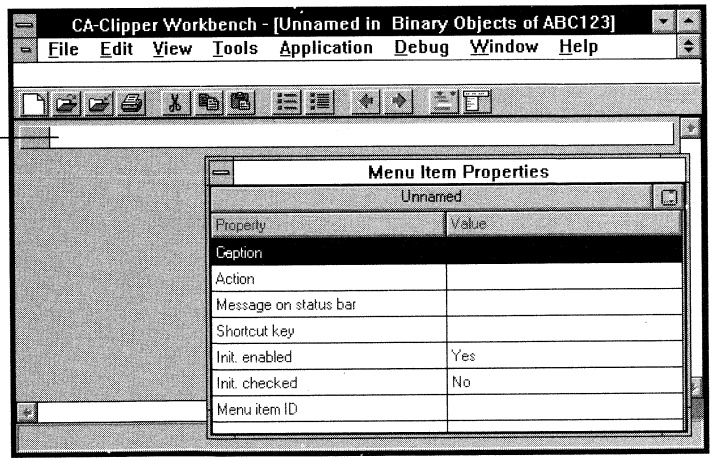
## Creating a Menu

To create a menu, perform the following steps:

1. Start the Menu Editor.

   The Menu Editor is accessed using the Tools menu or the New Entity toolbar button from within the Binary Objects module.

   The Menu Editor appears as follows, displaying a single, empty edit control:



2. Specify a name and, optionally, other properties for the menu structure. See Specifying Menu Properties for more information.
3. Enter the caption for each entry as you want it to appear either in the menu bar or on the menu. See Adding Menus and Menu Items for more information.
4. Promote/demote any entry necessary to create the hierarchy. See Creating the Hierarchy for more information.
5. Optionally preview the menu bar and each of the menus that you have defined. See Previewing the Menu Bar for more information.
6. Customize each entry by specifying properties for it. Certain properties, such as the captions that you have already entered, are required, while others are optional. See Specifying Menu Item Properties for more information.

7. Choose the Save toolbar button to save the menu.

**Note:** This last step can be repeated whenever you make changes to the menu design and want to save your work without closing the Menu Editor.

**Tip:** The Clear toolbar button can be used at any time to start a new editing session without shutting down the Menu Editor. Unless you save the menu you are currently working with before starting a new session, you will be prompted to do so before the Menu Editor opens the new menu for editing.

## Specifying Menu Properties

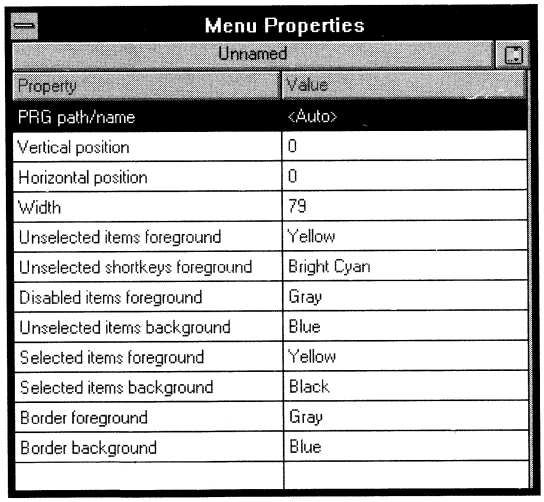
Before you get started with the menu design, you may want to go ahead and define a few properties for the menu structure, including its name and the name of the associated .PRG file.

### Menu Name

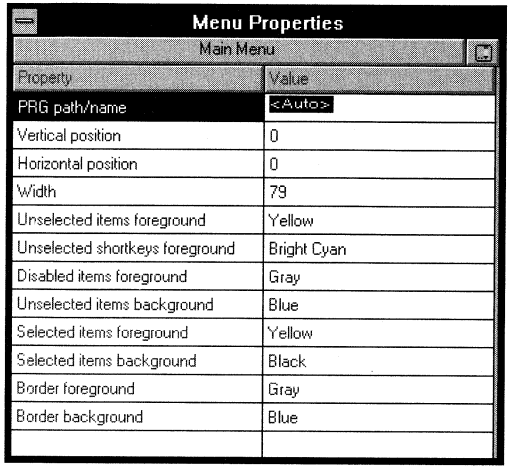
At a minimum, you are required to supply a name for the menu structure before you can save it. To do this, complete the following steps:

1. Click on the Pencil button to the right of the 3-D bar titled "Unnamed" at the top of the property list.

   The Menu Item Properties window becomes the Menu Properties window:



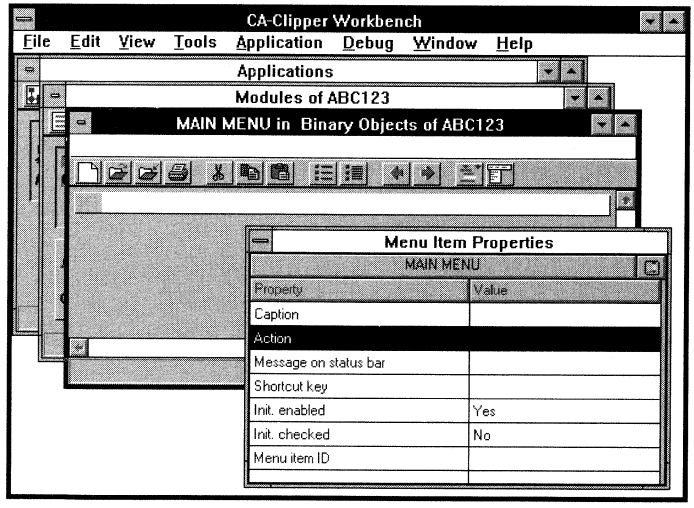
2. Enter a name in the edit control that appears (for example, **Main Menu**):




The name that you enter will be the name of the created entity in the Binary Objects module. This name, followed by "_PRG" will be the name of the module that will be created for the generated .PRG file. Therefore, it must not conflict with other entity names in your application.

3. Press Enter.

"MAIN MENU" now appears in the title bar of the Menu Editor:



**Note:** If you do not specify a name at this time, you will be prompted for a name the first time you save. At that time, you must enter a name or the Menu Editor will not be able to save the menu structure.

Specifying the remaining menu properties is optional.

PRG Path/Name

Specify the path name for the attached generated .PRG source file. The default is <Auto> , which uses the default path defined via the System Options dialog box and takes the first eight (8) characters of the menu name as the file name with .PRG as the extension.

Vertical Position

Enter a value (0 - 49) for the vertical position on the DOS screen where the menu bar is to appear. The default is 0.

Horizontal Position

Enter a value (0 - 79) for the horizontal position on the DOS screen where the menu bar is to appear. The default is 0.

Width

Enter a value [1 to (79 – Horizontal Position)] for the width of the menu bar on the DOS screen. The default is 79.

Unselected Items Foreground

Choose a foreground color from the combo box for unselected menu items. The default value is Yellow. Choosing <Auto> denotes no specific color instruction.

Unselected Shortkeys Foreground

Choose a foreground color from the combo box for unselected shortkeys. The default value is Bright Cyan. Choosing <Auto> denotes no specific color instruction.

**Tip:** You can save the current menu's colors as the *default color scheme* for any future menus by choosing the Save Colors as Default menu command.

Disabled Items Foreground

Choose a foreground color from the combo box for disabled menu items. The default value is Gray. Choosing <Auto> denotes no specific color instruction.

Unselected Items Background

Choose a background color from the combo box for unselected menu items. The default value is Blue. Choosing <Auto> denotes no specific color instruction.

| | |
|---|---|
| Selected Items Foreground | Choose a foreground color from the combo box for selected menu items. The default value is Yellow. Choosing <Auto> denotes no specific color instruction. |
| Selected Items Background | Choose a background color from the combo box for selected menu items. The default value is Black. Choosing <Auto> denotes no specific color instruction. |
| Border Foreground | Optionally, specify a different color for the border only. The default value is Gray. |
| Border Background | Optionally, specify a different color for the border's background area only. The default value is Blue. |

## Adding Menus and Menu Items

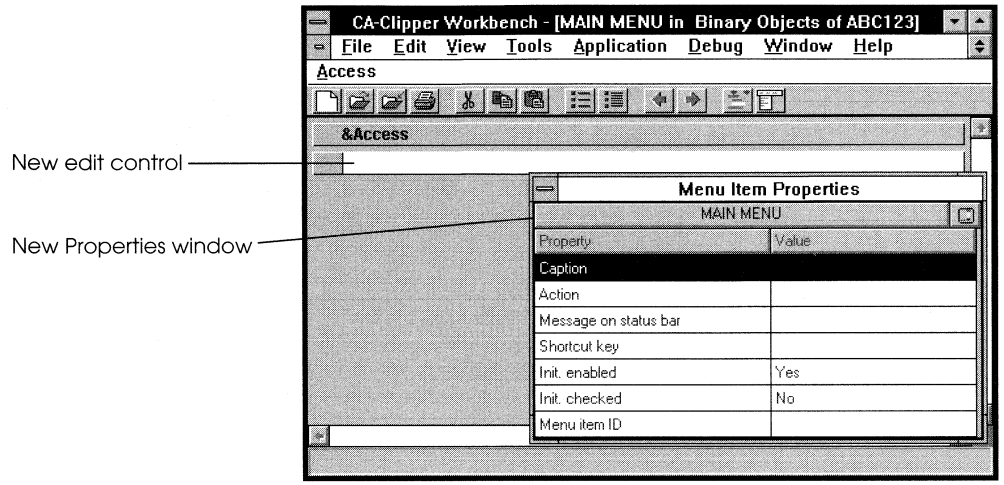
The remaining Menu Editor features will be demonstrated using the simple accounting application, ABC123, as our example. We will create a menu structure for it that includes a single menu called "Access," which allows the end user to maintain account, customer, vendor, and part records; invoices, purchase and sales orders. Additionally, the Access menu allows the end user to access financial transaction data entry screens using a submenu.

**Tip:** Each menu and menu item can feature an *accelerator key*, indicated by an underlined letter. Accelerator keys allow fast access to the pull-down menu itself by pressing Alt and the specified key together (or just the specified key in the case of a menu item on a pull-down menu). For example, the "F" in File menu and the "x" in Exit command could be underlined, indicating that you can select the File Exit command by pressing the Alt+F, X key combination. To add this type of functionality to your menus, simply preface the letter that is to be highlighted with an ampersand (&).

To create the Access menu for our sample application:

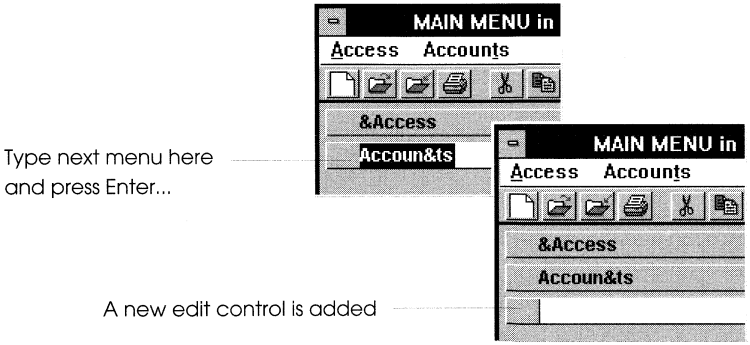
1. Click on the first empty edit control, and type **&Access**.
2. Press Enter.

   A new edit control and Menu Item Properties window appears for the next menu item:

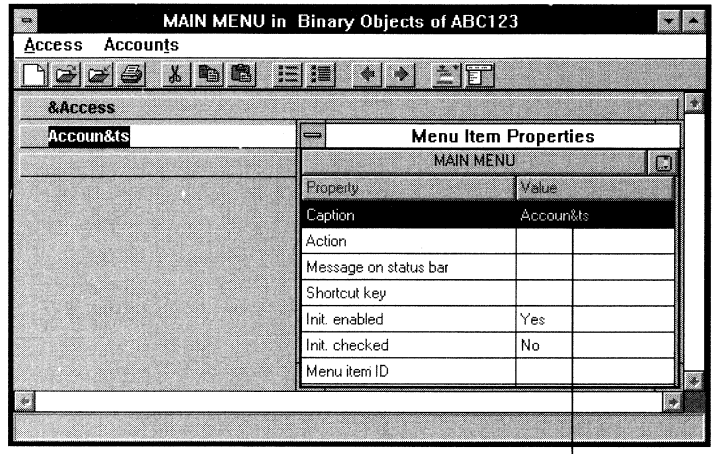



3. Click on the new edit control to open it.
4. Type the next entry (for example, **Accoun&ts**), and press Enter.

   Again a new line appears:



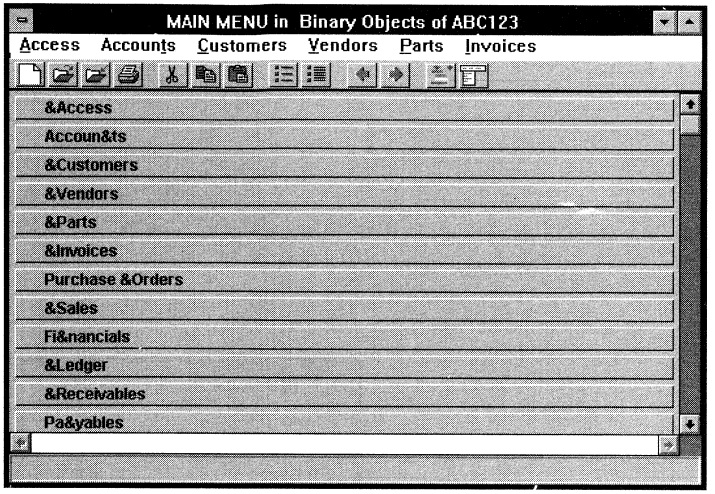
As each new entry is added to the menu, the Menu Item Properties window is also updated behind the scenes using each entry's text as the caption for that menu. For example, if at this point you click on "Accoun&ts," you will see the following:




Accoun&ts added as caption

5. Repeat steps 3 and 4 ten times, using the text **&Customers**, **&Vendors**, **&Parts**, **&Invoices**, **Purchase &Orders**, **&Sales**, **Fi&nancials**, **&Ledger**, **&Receivables**, and **Pa&yables**.

When you are finished, your menu structure should look as follows:




Notice that the Workbench automatically places corresponding entries in its preview menu bar for each item you enter (space permitting). Because no menu hierarchy has been defined, the Menu Editor assumes that all entries are menus.

## Creating the Hierarchy

Next, you will create the actual hierarchy of the menu structure to designate which entries are menus, which are menu items and, among the menu items, which have submenus. In this example, Access will act as the topmost menu, containing eight menu items (seven commands and a submenu). The submenu will contain three menu items, all of which are commands.

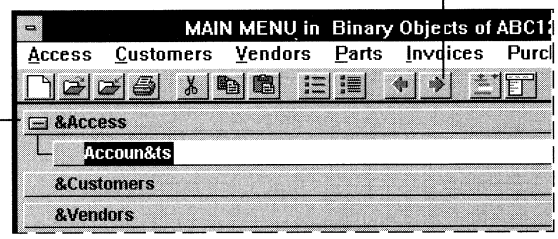
Creating a hierarchy out of all these items is easy in the Menu Editor—simply use the Promote Item and Demote Item buttons in the toolbar (or their corresponding commands on the Edit menu).

To create the menu hierarchy for the sample ABC123 application:

1. Select the Accounts entry by clicking on it in the Menu Editor window, and click the Demote Item toolbar button.

   The Menu Editor immediately indents the Accounts entry so that it is one level below Access, making it a *child*, or submenu, of Access. It also removes the Accounts entry from the preview menu bar—the preview menu bar is continually updated as you work, providing immediate visual feedback:

   Click the Demote Item button to make Accounts a menu item

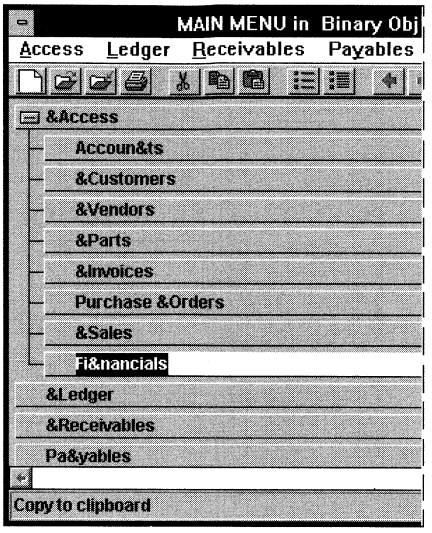
   
   

   Collapse/Expand (+/-) button

   **Note:** As you can see, the Menu Editor represents the hierarchy of the menu in a tree-like structure. The + / – button added to the left of the Access entry is a Collapse/Expand toggle button; single-clicking on this button allows you to alternately hide sublevels of a menu and then restore the full menu structure. The Collapse All and Expand All toolbar buttons, on the other hand, allow you to collapse and expand the entire menu structure.

2. Repeat step 1 seven more times to demote the Customers, Vendors, Parts, Invoices, Purchase Orders, Sales, and Financials entries.

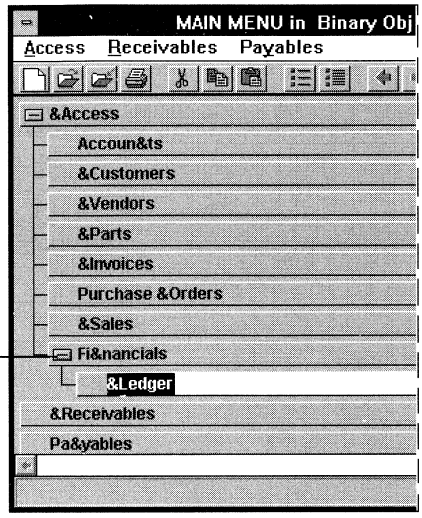
All demoted entries are now menu items on the Access menu. The menu structure looks as follows:




You now need to demote the remaining entries so that they become child menu items under Financials. By doing so, Financials automatically becomes a submenu.

3. Place the cursor in the Ledger entry and click the Demote Item toolbar button *twice*.

The menu structure is updated as follows:

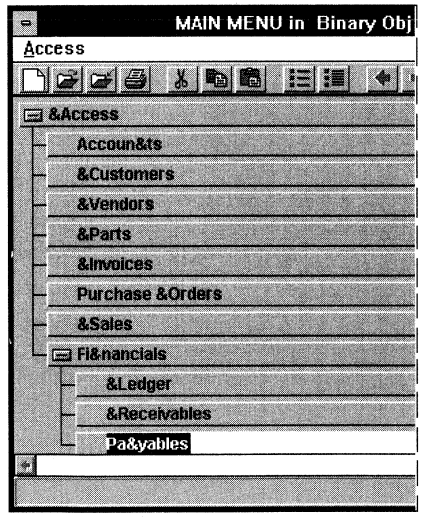



Collapse/Expand button for submenu

Because Financials is now a menu (albeit a submenu), it also gets a Collapse/Expand button.

4. Repeat step 3 two more times for the Receivables and Payables entries.

The menu structure now looks as follows:

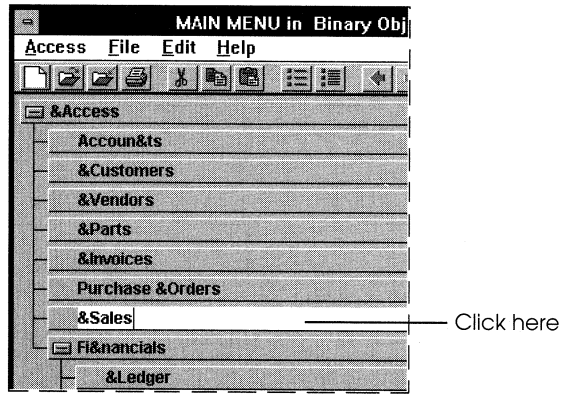


## Adding Separators

Separators are commonly used to group menu items logically within a menu. They can be added after you have created your menu hierarchy.

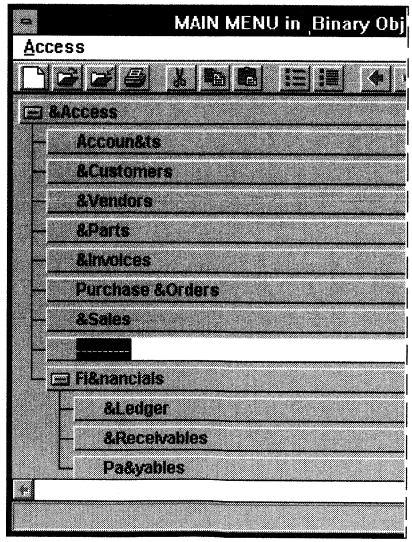
For example, you might want to separate record maintenance activities from transaction activities. So to separate Financials from the other entries in our Access menu:

1. Click on the **&Sales** entry:



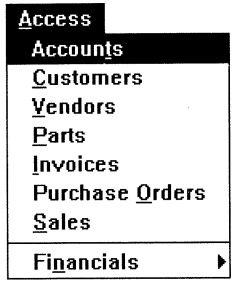

2. Choose Insert Separator from the Edit Insert Item menu.

   The menu structure now look as follows:



## Previewing the Menu Bar

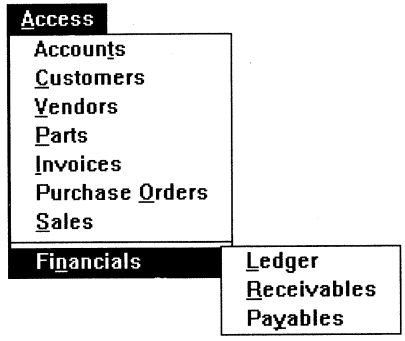
After completing the steps outlined in the previous section, you may notice that the Menu Editor's preview menu bar has been reduced to just the Access entry. At any time, you can select the entries in this menu bar (just as you would a real menu) to preview what your menus look like. For example, click Access to display the following:




Notice that the Menu Editor has added the underlining for the letters indicated using the (&) and the (▶) symbol to indicate a submenu for the Financials menu item.

**Note:** On your DOS window a different color is used for each of the highlighted letters instead of underlining.

If you then click on Financials, its submenu appears:



## Specifying Menu Item Properties

Once you have added an entry to a menu structure, you can then specify properties for it using the Menu Items Properties window. By simply typing the name of the entry in the Menu Editor window, you have already defined one property, the caption, but you can define additional properties using one of the techniques described earlier in this chapter. Each property is discussed in detail below.

Caption

Enter the text that will appear in the menu bar (for a menu), or on the menu (for a menu item). This property is required and is automatically filled in when you type a value in the Menu Editor window.

Captions can include text, blanks, punctuation, and the underscore character (_). Additionally, you can use an ampersand (&) in the caption to specify that the character immediately following it be underscored, indicating that it is the menu item's accelerator key. With the exception of the ampersand, all characters will appear exactly as typed.

**Note:** The shortcut key, check mark, and the right arrow symbol (▶), used to indicate a submenu, are not part of the caption. These indicators are added automatically by the system. However, the standard ellipsis (...) indicator, used to indicate that a menu item opens a dialog box, *must* be included as part of the menu item's Caption property.

**Tip:** Since you can add captions and edit them directly on the Menu Editor window, it is never necessary to use this property value cell; however, you can use it as an alternative means for entering and editing captions if you prefer.

Action

Enter a function call to be performed when this menu item is selected.

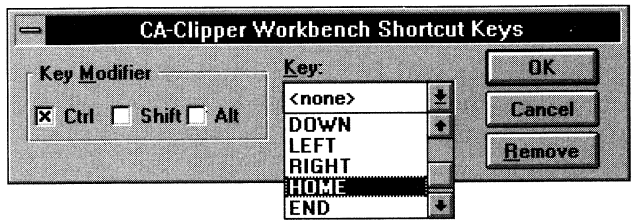
**Note:** The code for the function call should be entered in a *different* module in the current application. Otherwise, the added function will be deleted *if* you later modify the menu using the Menu Editor, as the Menu Editor will regenerate the .PRG file.

Message on Status Bar

Enter the text that will appear in the status bar when the entry has focus.

Shortcut

Define a shortcut key using the Shortcut Keys dialog box:




In this dialog box, the Key Modifier group box allows you to optionally choose Ctrl, Shift, or Alt as part of the shortcut key. (Note that combinations of key modifiers are not allowed.)

The Key combo box allows you to enter a key name or select one from a drop-down list. To define a shortcut, the key name is required.

OK saves the shortcut and closes the dialog box. Remove eliminates a previously defined shortcut and closes the dialog box.

At runtime, the shortcut that you define will appear to the right of the menu item caption on the menu and will be operational (that is, the user can press the shortcut to select the menu and then choose the item).

Init. Enabled

Specify whether the initial state of an entry will be enabled or disabled. This is a Yes/No option that you change using the plus (+) and minus (–) buttons. The default value of Yes indicates enabled.

An *enabled* entry can be selected by the user. A *disabled* entry appears dimmed (grayed) and cannot be selected. It remains unavailable until it is enabled by the application.

Init. Checked

Specify whether a check mark is displayed to the left of an entry when it is initially displayed. This is a Yes/No option that you change using the plus (+) and minus (–) buttons. The default value of No indicates that the entry will not be checked.

For those entries whose purpose is to indicate a *toggle setting* (for example, an ON/OFF condition), you can create them as checked or unchecked to denote the initial status of the toggle setting. When the user chooses the item, its status will change (for example, if it is currently checked it will become unchecked).

**Note:** Although the visual indication of initial checking and unchecking occurs automatically as part of the menu structure definition, the action code to make the underlying change in the application must be defined in the program.

Menu Item ID

Enter a numeric value for identifying the menu item if needed in the .PRG file.

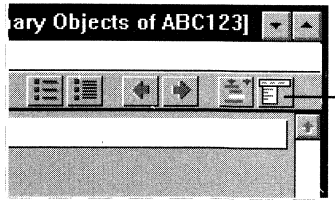
# Adding Predefined Menus

The Menu Editor allows you to quickly add standard, predefined File, Edit, View, and Help menus to your menu structure. Each predefined menu has *default* menu items. The functions to be called, however, must be added manually in the Menu Item Properties window.

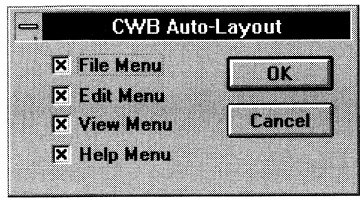
**Tip:** When adding one or more predefined menus to an existing menu, be sure to place the cursor in the entry *after* which you want the menus to be inserted.

To add one or more standard menus to your menu design:

1. Click the Auto Layout button in the Menu Editor toolbar.



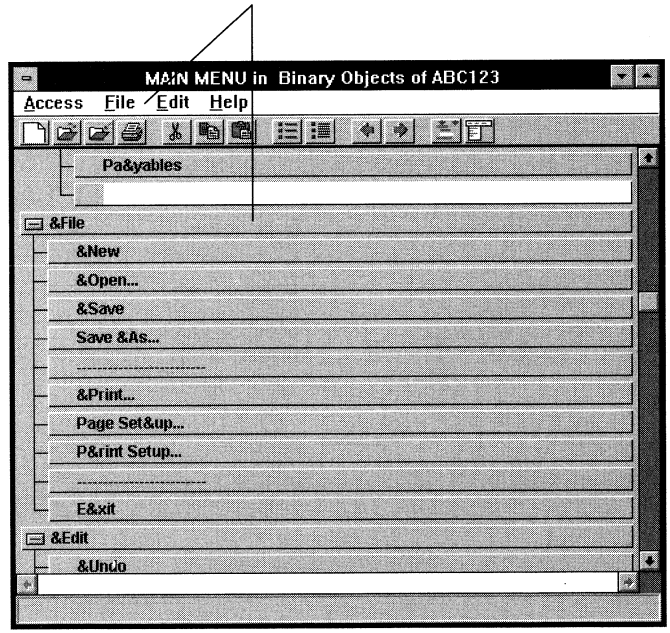
The Auto-Layout dialog box appears with all menus selected:




2. Make sure that all menus you want to add are checked.
3. Choose OK.

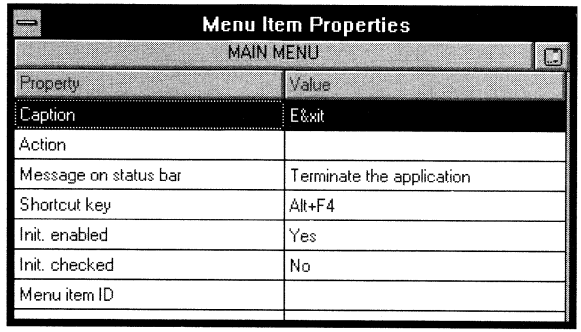
For example, if the File, Edit, and Help standard menus are added to the end of our earlier sample menu structure, Main Menu, the result would be:

Standard menus added to preview menu bar and menu structure



**Note:** If there is a blank line anywhere in your menu structure, you must either use this blank line to add another menu entry manually or delete it. Blank menu items are *not* allowed and, consequently, you cannot save your menu entity if they are not deleted.

Now, if you click on any one of these predefined menus and menu items, you can view its default properties. For example, below are the default properties for the standard File Exit menu command:




You can, of course, modify any of these default values (see Modifying Menu Item Properties). For more information about the Action and Menu Item ID properties, see Specifying Menu Item Properties. Also refer to the Using the Menu in an Application section later in this chapter for an example of a menu associated with a form via a function call in the Action property value cell.

# Modifying a Menu

After you have defined a menu structure, you can modify it by changing its properties or the properties of any entry associated with it. You can also delete existing entries and insert new ones.

**Note:** The Cut, Copy, and Paste commands apply only to the caption property. This means, for example, that if you copy the currently highlighted entry and paste it elsewhere on the menu, all properties besides caption will be reset to the original defaults for the newly pasted entry.

## Editing Menu Properties

To change any property associated with the menu structure, click the Pencil button in the Menu Items Properties window to the right of the menu name. The window will change to show the properties for the menu structure; you can change any property you want. See Specifying Menu Properties for more information.

## Editing Menu Items

To make changes to an entry on the Menu Editor window, you must first select it by clicking on it. When an entry is selected, it is highlighted and the Properties window changes to show its properties.

### Modifying Menu Item Properties

You can change the caption by simply editing the currently highlighted entry directly on the Menu Editor window. You can also change any property in the Properties window. See Specifying Menu Item Properties for more information.

## Inserting Menu Items

New menu items can be inserted at different levels of the menu hierarchy.

### Inserting a Sibling

To insert an entry at the *sibling* (or same) level after the currently selected entry:

1. Select the Insert Item toolbar button.

   Alternatively, choose the Edit Insert Item command and select Insert Sibling.

2. When the new edit control appears, type the caption name.
3. Define the menu properties for the new entry as discussed in Specifying Menu Item Properties earlier in this chapter.

### Inserting a Child

To insert an entry at the *child* (or submenu) level after the currently selected entry:

1. Choose the Edit Insert Item command and select Insert Child.
2. When the new edit control appears, type the caption name.
3. Define the menu properties for the new entry as discussed in Specifying Menu Item Properties earlier in this chapter.

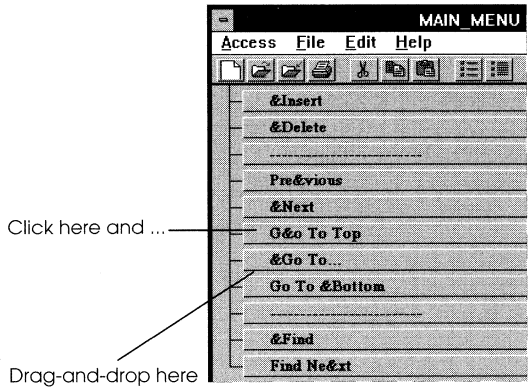
## Reordering Menu Items

You can rearrange the order of your menu items using the Menu Editor's *drag-and-drop feature*. For example, to move the G&o To Top entry in our example to a position following the &Go To entry:

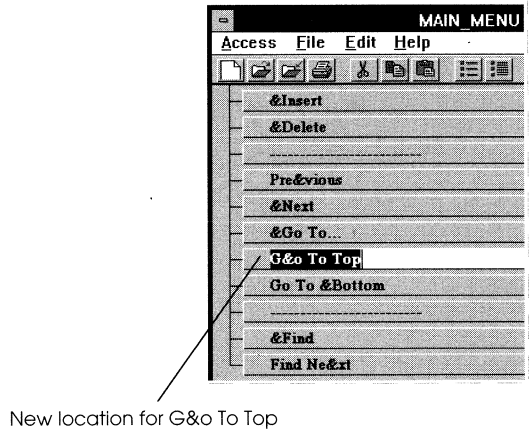
1. Click on G&o To Top.
2. Without releasing the left mouse button, begin dragging the selection.

   The mouse pointer changes to the  icon and a red guide line appears.

3. When the red guide line reaches the *top* of the Go To &Bottom menu item's 3-D bar, drop the icon by releasing the left mouse button:




The Menu Editor inserts the selection in the new location:




## Deleting Menu Items

To delete the currently selected entry, including all its properties and child entries, use the Edit Delete Item menu command.

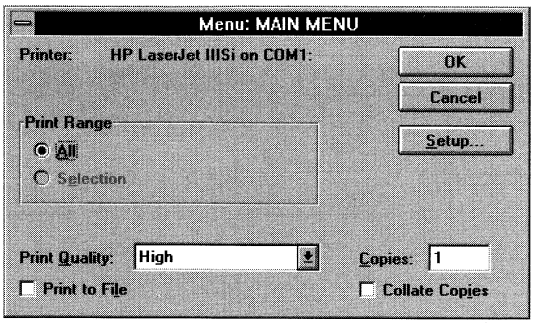
If you want to delete the caption only, choose the Edit Delete command (or press the Del key).

# Printing Menus

To print a copy of your menu design:

1. Choose the Print toolbar button.

   The Printing Menu dialog box appears:




2. Specify your printing options.
3. Choose OK.

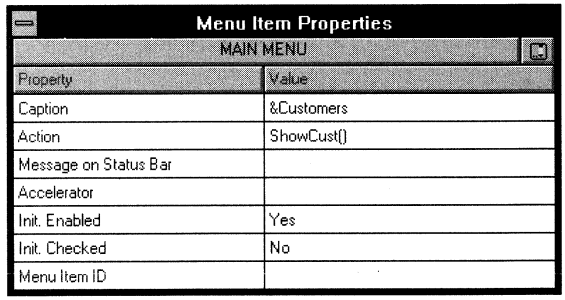
For descriptions of the available standard printing options, see Printing an Application List in the "Browsing Applications, Modules, and Entities" chapter.

# Using the Menu in an Application

Menus that you create using the Menu Editor are typically attached to forms. The easiest way to make this connection is to enter a function call to the form (or any other action) using the Action property of the menu item.

For example, to associate our &Customers menu item with our sample Customer Maintenance form:

1. Open our sample menu entity, Main Menu, in the ABC123 application.
2. Click on the &Customers menu item.
3. Enter a function call to the form in the Action property value cell (for example, **ShowCust()**):




4. Choose Save.
5. Now, open the Customer_Maintenance program module in the Source Code Editor and manually enter the code for the ShowCust() function.

   **Note:** The code for the function call should be entered in a *different* module as we have just done here in our example. Otherwise, the added function will be deleted *if* you later modify the menu using the Menu Editor, as the Menu Editor will regenerate the .PRG file.

6. Choose Save.

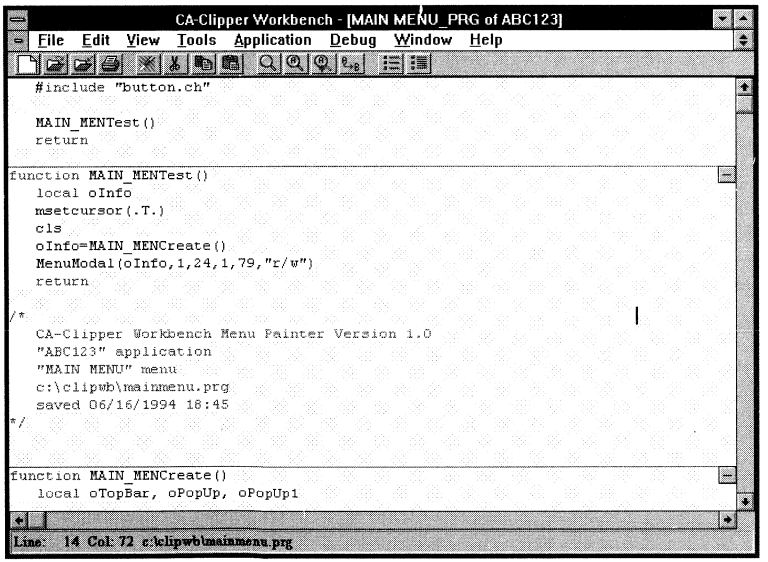
When the Access Customers menu command is selected in the final application, the Customer Maintenance form will appear, allowing the end user to view and maintain customer records.

**Note:** The MENUMODAL() function takes one menu event from the user and then terminates. To avoid this, you can manually edit your code so that your program continuously accepts menu events. To do this, use:

```
DO WHILE (MENUMODAL(themenu,.......) <.> ExitMenu)
ENDDO
```

# Generating Code

When you save a menu, the Menu Editor automatically generates a menu entity, which is placed in the Binary Objects module, as well as a separate program source file (.PRG) module. You can subsequently view and/or edit the source code for your menu in the Source Code Editor. For example, below is our code for the menu entity we just created:



## A Closer Look at the Source Code

Note that the Menu Editor automatically generates predefined functions and static functions for any menus that you create. For example, if you created a menu called XYZ, the following functions would be defined for it:

| Function | Purpose |
|---|---|
| XYZTest() | Allows testing of the menu display. |
| XYZCreate() | Creates the menu and returns a menu object.<br><br>**Note:** A menu object can be used while entering data in a form, in which case the menu object needs to be passed as the second parameter to the XYZEdit() function of the form (generated by the Form Editor). See Generating Code in the "Using the Form Editor" chapter for more information. |

For more information about generated functions, see the *Reference Guides, Volumes 1* and *2*.

# Chapter 6
# Using the Source Code Editor

## In This Chapter

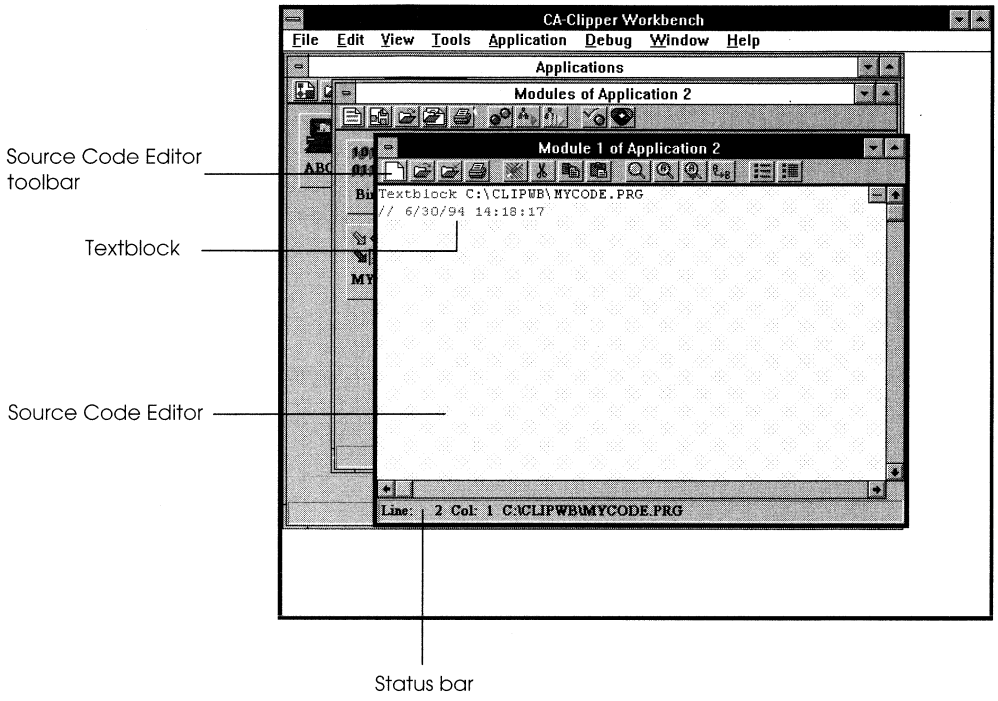
The Source Code Editor provides a powerful environment for writing and editing code, drawing much of its strength from its close integration with the Workbench's *repository*. It is the primary workspace in the Workbench for viewing, creating, and modifying source code entities, such as defines and functions.

This chapter describes how to use the Source Code Editor to:

- Create new entities when typing entity keywords
- Go immediately to a specific entity for editing
- Perform standard editing features such as cut, copy, paste, delete, search for text, replace text, undo, and redo using standard Microsoft Windows techniques for any entity in the module currently loaded
- Print the source code for all the file entities currently loaded
- Access other browsers and editors

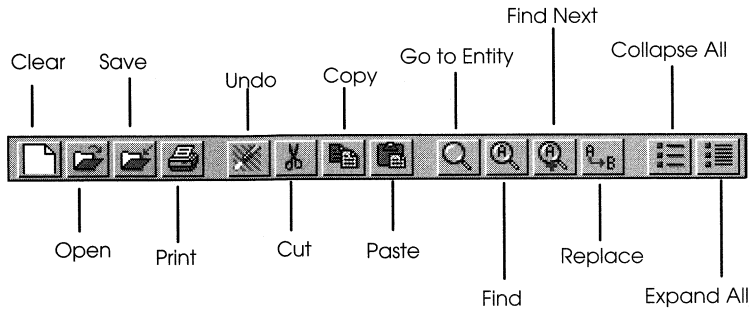
# Workspace Overview

The Source Code Editor has its own toolbar and status bar. When first loaded for a *new* source code entity, an empty Source Code Editor window with a predefined textblock is displayed:



## The Toolbar

The Source Code Editor toolbar contains the following buttons:




All the above buttons are described in this chapter.

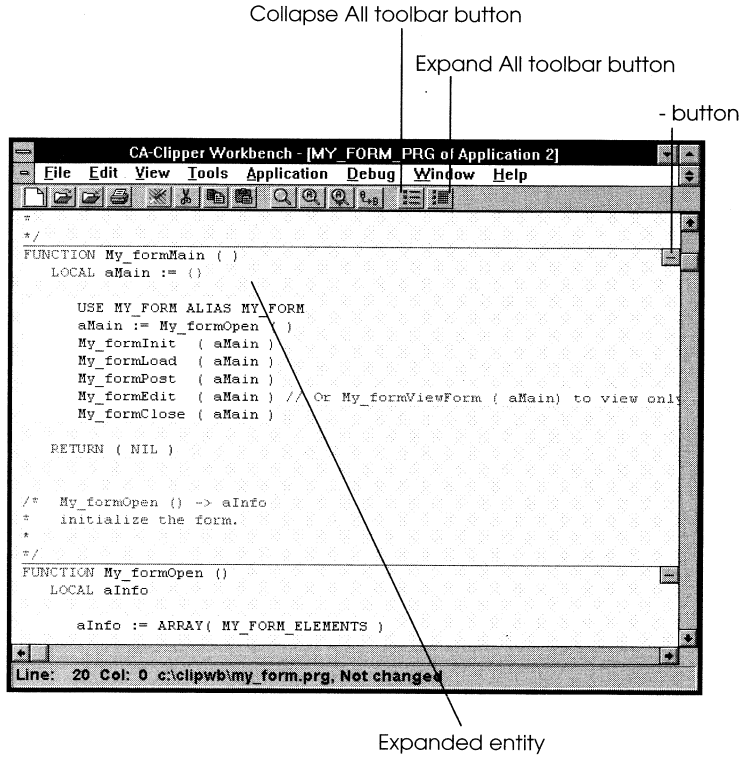
> **Tip:** For a quick description of these toolbar buttons, look at the status bar as the mouse pointer passes over the buttons. Also see your online Help reference.

## The Status Bar

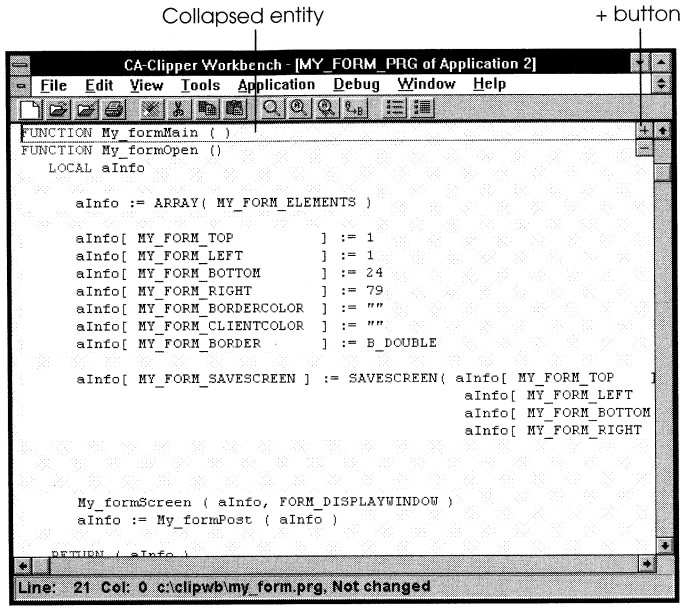
To facilitate your editing session, the status bar in the Source Code Editor displays the line and column number of the current cursor position, as well as related file information such as the file name and path of the associated external source file. It also indicates whether the contents of the editor have been changed or not.

## Collapsing/Expanding

As you begin to use the Source Code Editor, you will notice that each entity displayed in the editor has a minus (–) button to the right of its declaration statement as illustrated below:



By default, each entity is displayed in its entirety. Clicking on the minus (–) button collapses the entity, so that only its declaration statement can be seen, and changes the minus (–) button to a plus (+) button.

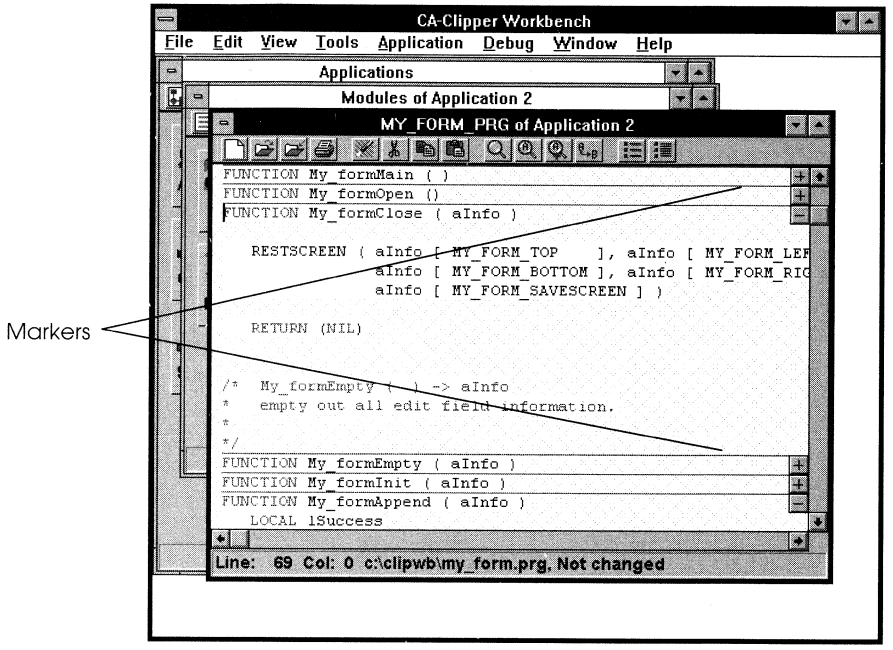



The plus (+) button indicates that the entity is collapsed. Clicking on it expands the entity, showing all the source code.

**Tip:** Instead of collapsing and expanding one entity at a time, you can use the Collapse All and Expand All toolbar buttons to collapse and expand the entire content of the Source Code Editor at once.

## Markers

Another feature is the *markers* that are displayed between entities to visually delimit one from the next:




You can toggle these markers on or off by alternately checking and unchecking the Entity Markers command on the View menu. The + /– buttons for the entities remain on display in either case.

## Color Coding

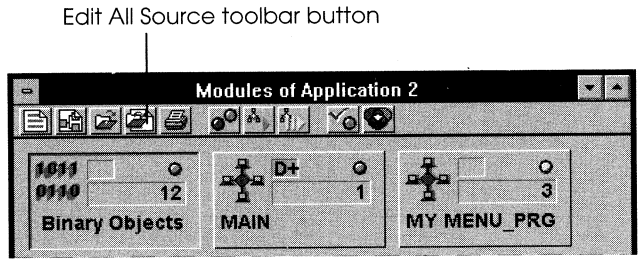
Additionally, the Source Code Editor continually parses each keystroke to color-code text based on its structure. Keywords, constants, and comments, for example, are all displayed in different colors for immediate visual feedback.

# Accessing the Source Code Editor

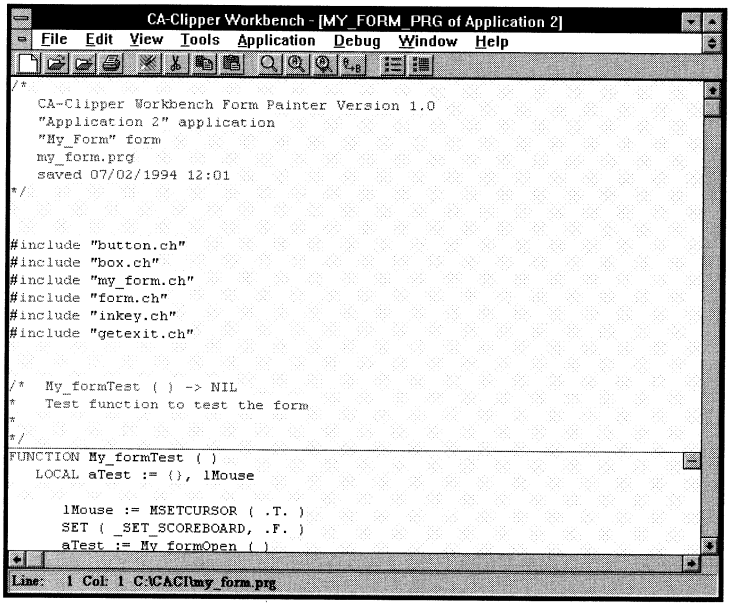
Now that you have a general overview of the Source Code Editor's features, you are ready to use it to create and edit source code entities. In this section, you will learn how to access your source code.

## Accessing All Entities Within a Source File

To access *all* existing entities for the current module, load them all at once in the Source Code Editor using the Edit Whole Source button in the Module Browser toolbar:



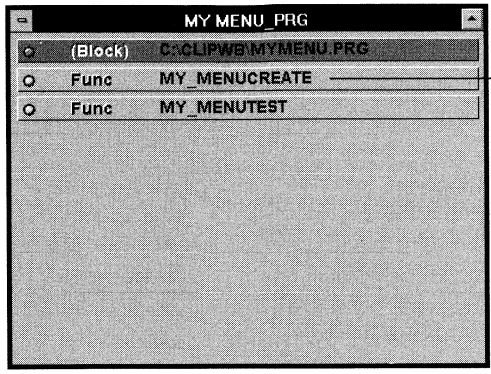
For example, selecting the MY MENU_PRG module and clicking on the Edit All Source toolbar button results in the following code:



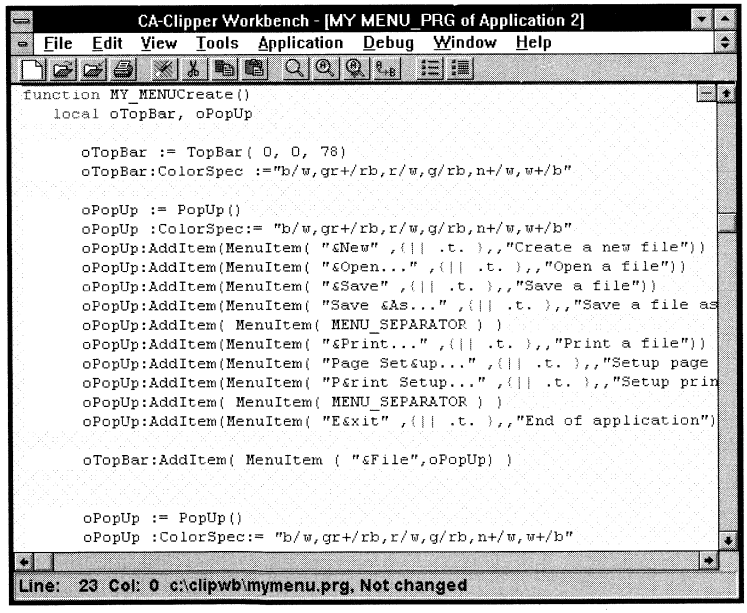

**Tip:** When you have several entities loaded in the Source Code Editor, use the Go to Entity toolbar button to go directly to a particular entity that you wish to edit. See Going Directly to an Entity later in this chapter for more information.

## Accessing an Individual Entity Directly

To access an *individual* entity, simply double-click on the specified entity in the Entity Browser. For example:




The code for the specified entity—in this case, the MY_MENUCreate() function—appears:



## Creating, Editing, and Saving Entities

Regardless of which technique you use, once the source code is loaded in the Source Code Editor, you can:

1. Create as many new entities as you want by manually entering declaration statements and source code for them.
2. Edit it using standard Microsoft Windows editing techniques. See Editing Source Code below for detailed information on editing techniques unique to the Source Code Editor.
3. Choose the Save toolbar button to save the source code. Any new entities that you have created during the editing session will be added to the Entity Browser for the appropriate module and attached DOS file.

**Note:** This last step can be repeated whenever you make changes to the source code and want to save your work without closing the Source Code Editor.

**Tip:** The Clear toolbar button can be used at any time to start a new editing session without shutting down the Source Code Editor. Unless you save the source code you are currently working with before starting a new session, you will be prompted to do so before the Source Code Editor is cleared.

# Editing Source Code

The Source Code Editor provides you with standard editing features. For example, there are toolbar buttons for cutting, copying, pasting, undoing editing changes, finding text, and replacing text. All of these operations are performed using standard Microsoft Windows techniques and, therefore, are not described in this section with the exception of finding and replacing text.

If you are unfamiliar with standard editing techniques such as the operations mentioned above, cursor movement, toggling between insert and overwrite mode, and selecting text, refer to your online Help reference.

The following editing features, however, are unique to the Workbench's Source Code Editor and are described in this section:

- Deleting individual lines of code
- Inserting a new line
- Going directly to an entity

Finding and replacing text is also discussed here, including the use of valid wild card expressions in the search for text.

## Deleting Lines of Code

Within the Source Code Editor, you can delete individual lines of code. To delete a *single line* of code, move the cursor to the line and press Ctrl+Y (or choose the Delete Line command from the Edit menu).

You can also delete several *contiguous lines* of code at once by selecting them and pressing the Delete key (or choosing the Delete command from the Edit menu).

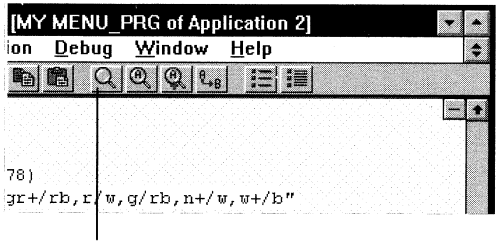
## Inserting a New Line

Use the Edit Insert Line menu command to insert a new line before the one in which the cursor is currently located. Then press Enter.

## Going Directly to an Entity

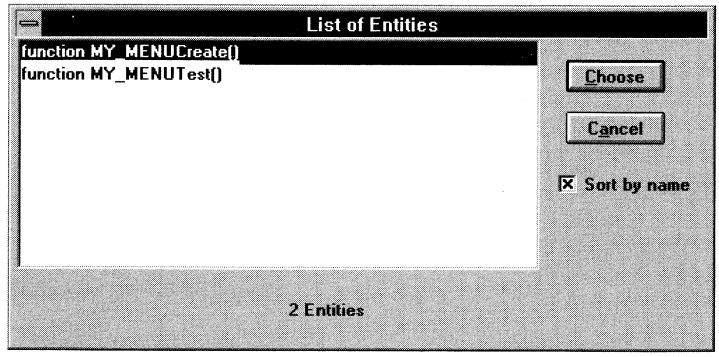
Since you may be editing many entities within the same module in the Source Code Editor, the Go to Entity feature enables you to access a particular entity quickly. To use this feature:

1. Select the Go to Entity toolbar button:




Go To Entity toolbar button

The List of Entities dialog box appears, showing the total number of entities currently loaded and their names. This example displays all entities defined for the MY MENU _PRG module in our example:



2. By default, the Sort by Name check box is checked and entities are displayed in alphabetical order by type, and then by name. If you like, you can uncheck Sort by Name to view the entities in the order that they appear in the Entity Browser (that is, the order in which they appear in the source file).

3. Highlight the entity you want to view, and select the Choose button.

   Alternatively, double-click on the entity.

   When the dialog box closes, you are returned to the Source Code Editor with the cursor located on the line of source code where the specified entity begins.

## Finding and Replacing Text

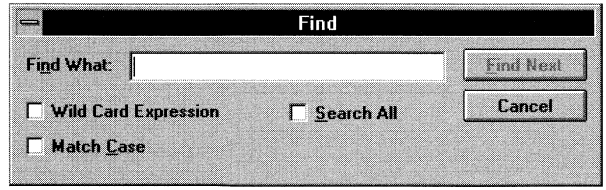
Finding and replacing text in the Source Code Editor utilizes standard editing techniques. The use of wild card expressions in the search for text, however, is restricted to valid CA-Clipper expressions.

### Searching for Text

To search for text while in the Source Code Editor:

1. Choose the Find toolbar button.

   The Find dialog box is displayed:




2. In the Find What edit control, enter the text that you want to find.

3. Choose an action:

   - To find the *next* occurrence of the specified text, choose the Find Next push button.

     The Workbench highlights the first occurrence of the specified text, leaving the Find dialog box open.

     If desired, you can move on to the next occurrence by clicking Find Next again. Or, if you closed the dialog box, you can choose the Find Next toolbar button or menu command.

     **Note:** If no match is found, a beep is sounded.

   - To find *every* occurrence of the specified text, check the Search All check box and then choose Find Next.

     The Workbench displays the Found the Following Lines dialog box, listing every line containing the specified text. If desired, double-click on the line you want to go to.

     **Note:** You can quickly redisplay this listing dialog box after exiting it by choosing the Find Next toolbar button or menu command.

The following options are available:

Find What

Enter the characters you want to search for. (You can also paste text from the Clipboard into this edit control using the Shift+Insert key combination.)

**Note:** If you have already issued either the Edit Find or Edit Replace command in this work session, the text that was previously entered in this edit control is still displayed. Type over this text to begin a different search.

Wild Card Expression

If checked, allows you to enter a wild card expression to search for similar text, or for a word whose correct spelling is unknown.

Valid CA-Clipper expressions are: * and ?. Use * to match any number of letters or characters; use ? to match any single letter or character.

Match Case

If checked, finds the specified text only where the case of each letter matches. If not selected, all occurrences of the specified text are found.

Search All

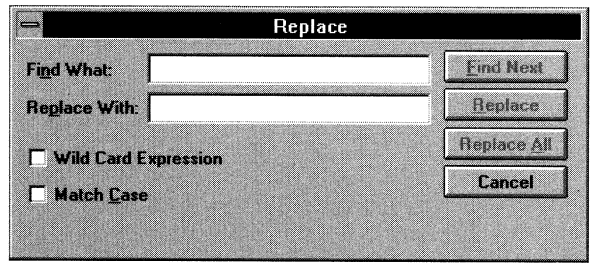
If checked, searches the entire module for every occurrence of the specified text. If not selected, only the next occurrence is found.

## Replacing Text

To find and replace text while in the Source Code Editor:

1. Choose the Replace toolbar button.

   The Replace dialog box is displayed:




2. In the Find What edit control, enter the search text.
3. In the Replace With edit control, enter the replacement text.

   **Note:** If you have already issued the Edit Replace command in this work session, the text that was previously entered in this edit control is still displayed. Type over this text to begin a different search.

   ***Important!*** *If you do not specify replacement text, the Workbench will* ***delete*** *the string specified in the Find What edit control from the text being searched.*

4. Choose an action:

   - To replace on a *case-by-case* basis, choose the Find Next push button to display each occurrence, and then choose the Replace push button each time you want to replace that occurrence.
   - To *globally* replace every occurrence, choose Replace All.

   **Note:** If no match is found, a beep is sounded.

With the exception of the following option, all available options for this dialog box are described in the preceding section.

Replace With

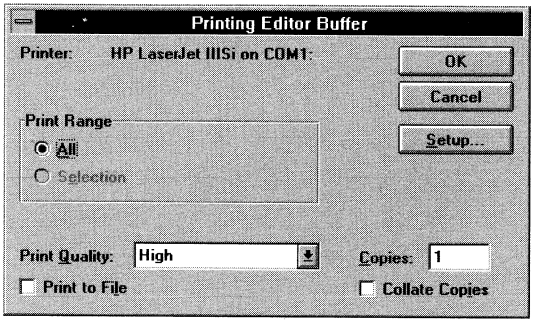
Enter the text which should be used to replace the text specified in the Find What edit control. (You can also paste text from the Clipboard into this edit control using the Shift+Insert key combination.)

# Printing Source Code

To print all the source code in the current Source Code Editor window:

1. Choose the Print toolbar button.

   The Printing Editor Buffer dialog box appears:




2. Specify your printing options.
3. Choose OK.

For descriptions of the available standard printing options, see Printing an Application List in the "Browsing Applications, Modules, and Entities" chapter.

# Chapter 7
# Defining Data Servers and Field Specs



## In This Chapter

The CA-Clipper Workbench provides a pair of GUI database editors that let you define, maintain, and utilize ancillary information concerning your external database files. The *DB Server Editor* allows you to create and modify *data servers* that are based on the traditional Xbase model of a database file; the *FieldSpec Editor* allows you to define field specifications, or *field specs*, independent of any data server.

When you create a data server or field spec, the respective database editor creates a corresponding entity in the Binary Objects module of the current application.

This chapter describes how to use the DB Server and FieldSpec Editors to:

- Create and modify data servers and field specs
- Specify field properties
- Print data server and field spec entities

**Note:** The properties that you define for a data server and its field specs are designed to be used by forms that you design and link to the data server. See "Using the Form Editor" for more information on creating forms and how to link field spec properties to data entry fields.

# What Is a Data Server?

Basically, a data server is a high-level, abstract entity designed to give you a consistent GUI interface for your database files and, more importantly, to allow them to interact with forms. The data server acts as a database describer, defining its file name, specifying the order in which it is accessed, and providing a mechanism for extending field definitions beyond the basic name, type, and length information stored in the database file structure—this last part is accomplished using a field spec.

## Data Server vs. Catalog

A data server should not be confused with a database *catalog*, which describes the structure of files on disk. While a catalog describes a file as it *exists*, a data server describes how the application intends to use the file.

As you work your way through the development cycle, you can make "on-the-fly" changes to a data server (for example, changing the validation or formatting rules for one or more fields). CA-Clipper ensures that these changes are reflected in all appropriate places, such as a form that is associated with that data server.

# What Is a Field Spec?

In many cases, the different data servers your application uses contain similar, if not identical, fields. For example, all zip code fields are the same regardless of where they are used. Other common examples are customer, account, and employee numbers. You can either define the properties of these common fields each time you create a new data server, or you can create a field spec that you reuse in each data server that needs it.

A field spec is essentially a template, that is, a set of properties (such as validation and formatting rules) that are related to a field but are *independent* of any particular data server. Thus, multiple data servers can access the same property values for common fields. For example, if you create a Salary field spec, you can simply reuse its properties in an EmpSalary field in a data server for an Employee database. You can then reuse its properties again when creating a similar field for a data server for a Payroll database.

Using field specs, therefore, saves you both time and resources. Another advantage is that any changes made to a field spec are automatically propagated to all the appropriate places. For example, if you add extra validation to the AcctNum field spec to require that its second digit must be zero or 5, then all data servers that use that field spec would perform that validation.

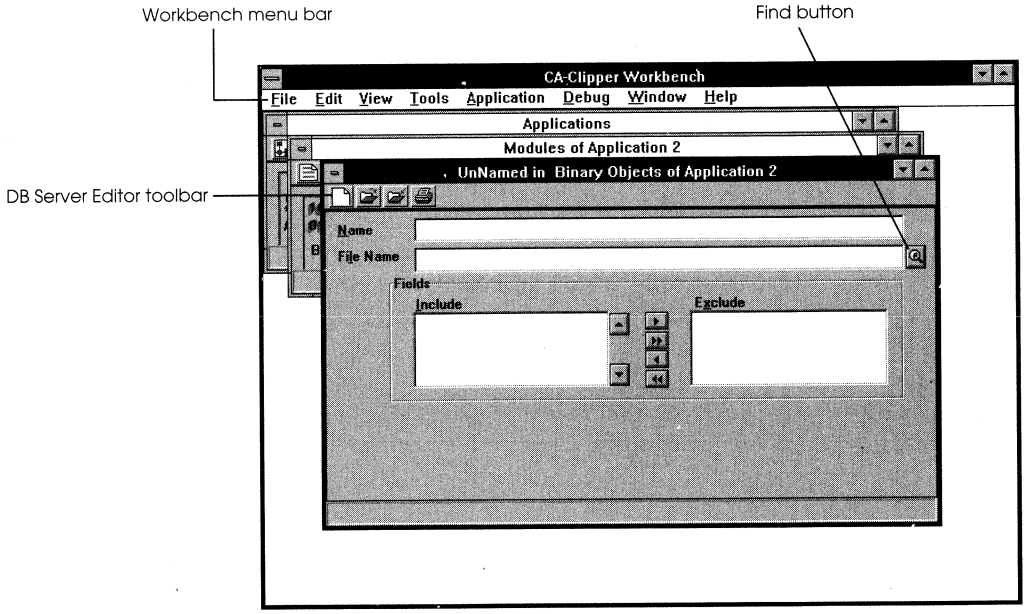
All data servers that use this field spec will be changed accordingly.

# Using the DB Server Editor

In this section, you will learn how to define a data server, import the structure of an existing database file, and specify its properties and fields.

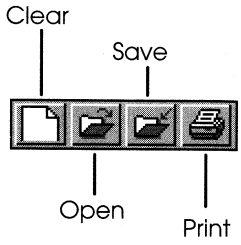
## The DB Server Editor Workspace

Similar to the Form and Menu Editors, the DB Server Editor has its own toolbar and an associated Properties window, in addition to the menu commands available on the Workbench menu bar. Note that, unlike the other visual editors, the DB Server Editor's FieldSpec Properties window does *not* appear initially:



## The Toolbar

The DB Server Editor toolbar contains the following buttons:




All of these buttons are discussed further in this chapter.

**Tip:** For a quick description of these toolbar buttons, look at the status bar as the mouse pointer passes over the buttons. Also see your online Help reference.

## FieldSpec Properties Window

The DB Server Editor's floating FieldSpec Properties window opens automatically *after* a .DBF file is associated with the data server, and allows you to specify properties for each field:

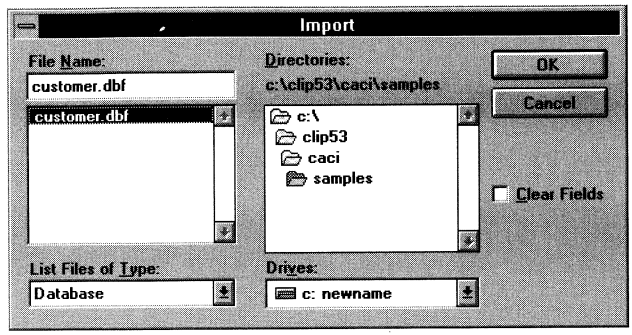
**Fieldspec properties**

| Property | Value |
|---|---|
| Name | CUSTNUM |
| Fieldspec | CUSTOMER BASE_CUSTNUM |
| Message on status bar | |
| Type | Numeric |
| Length | 5 |
| Decimals | 0 |
| Picture | |
| Min length | |
| Required | No |
| Minimum | |
| Maximum | |
| Validation | |
| | |

For example, you can enter text that should appear in the status bar when the field is selected, specify a validation rule, or associate the field with an existing field spec and use its properties. See Specifying Field Properties for more information about this window and available field properties.

## The Import Dialog Box

The DB Server Editor dialog box allows you to import a database file into the repository.




This dialog box contains the following options:

- File Name

  Enter or select the name of the .DBF file to be imported. This combo box displays files with the extension listed in the List Files Of Type box. By default, it is set to *.DBF.

  **Note:** To display a list of files that matches a certain criterion, type standard DOS wildcard characters in this combo box. For example, enter **new*.dbf** to display a list of files whose names begin with the letters "new" and have the file extension DBF.

- Directories

  Displays all available directories in the current path. You can use this option to select another directory.

- List Files of Type

  Displays the type of the files currently listed in the File Name combo box.

- Drives

  Lists all available drives. You can use this option to select another drive.

- Clear Fields

  Optionally, check the Clear Fields check box to clear any existing field definitions from the current DB Server Editor session. If you do not check Clear Fields, the fields in the imported file will be added to the existing field definitions.

- OK

  Confirm the choices and close the dialog box.

- Cancel

  Close the dialog box without taking any action.



## Defining a Data Server in the DB Server Editor

Now that you have a general overview of the DB Server Editor workspace, you are ready to use it to define a data server. In this section, you will learn how to:

- Create a data server and specify the associated .DBF file
- Optionally exclude fields from the data server's associated database file
- Specify properties for the data server and its fields

In later sections, you will learn how to modify an existing data server and its field properties.

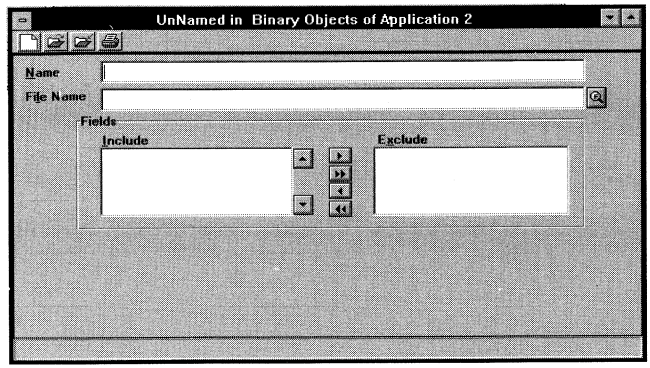
### Creating a Data Server

To create an Xbase-style data server, you need to perform the following steps:

1. Start the DB Server Editor.

   The DB Server Editor is accessed using the Tools menu or the New Entity toolbar button from within the Binary Objects module.

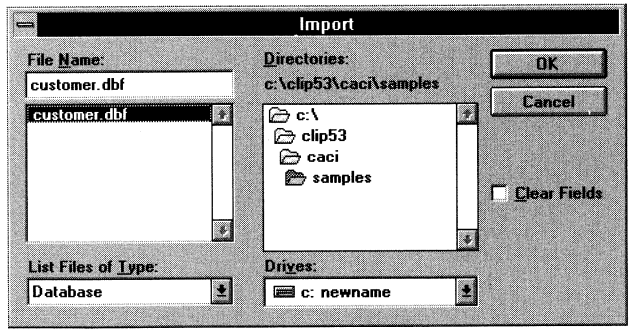
The DB Server Editor appears as follows:




2. You may type the name of the data server in the Name edit control (for example, **Customer Base**). This name is used to create the server *entity*. If left blank, the name of the associated .DBF file is used.

   You can enter a name using a maximum of 64 characters. The first character must be alphabetic or an underscore; the other characters can be alphanumeric and can include the underscore character. Blanks within the name will be changed to underscores when you save the data server.

3. Click on the *Find button* (the one with the magnifying glass graphic). A standard Import dialog box appears:




   Highlight a database file (for example, CUSTOMER.DBF) and then click OK.

## Importing a Database File

Use the File Import menu command while in the DB Server Editor to import a database file into the CA-Clipper repository.

You can import an existing database file as follows:

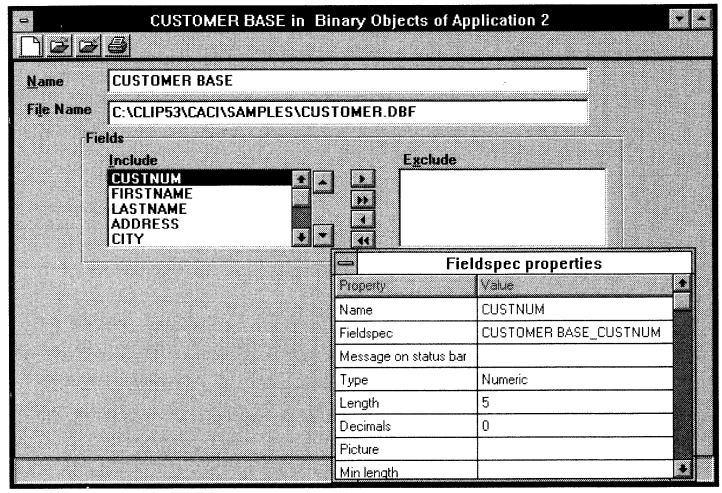
1. Start the DB Server Editor.
2. Choose the Import command from the File menu to display the standard Import dialog box.

   **Tip:** Alternatively, click on the Find button to the right of the File Name edit control.

3. Optionally, check the Clear Fields check box to clear any existing field definitions from the current DB Server Editor session. If you do not check Clear Fields, the fields in the imported file will be added to the existing field definitions.
4. Select the desired drive and directory, and double-click on the file name that you want to import.

   The Import dialog box closes, the File Name edit control is updated with the imported file name, and all the fields in the database file structure are now in the Include list box.

The data server name replaces "UnNamed" in the title bar of the DB Server Editor; and the database file name, drive, directory, and extension are added to the FileName edit control. Lastly, the database field names are added to the Include list box. For example:




**Note:** Once you enter or select the associated .DBF file, the *FieldSpec Properties window* appears. Also, if you left the Name edit control blank and selected a .DBF file using the Find button, the Find button disappears. (It reappears if the Clear toolbar button is used subsequently.)

5. Optionally, exclude one or more fields from the data server's Include list box. (For more information, see Including and Excluding Fields presented later in this chapter.)
6. Specify field properties using the FieldSpec Properties window.
7. Choose the Save toolbar button to save the data server.

**Note:** This last step can be repeated whenever you make changes to the data server and want to save your work without closing the DB Server Editor.

When you save a data server, the Workbench automatically generates a data server entity in the Binary Objects module of the current application. You can double-click on this entity in an Entity Browser to begin editing it with the DB Server Editor.

**Tip:** The Clear toolbar button can be used at any time to clear the DB Server Editor workspace and start a new editing session. Unless you save the data server you are currently working with before starting a new session, you will be prompted to do so before the workspace is cleared.



## Including and Excluding Fields

When a database file is initially associated with the data server, all of its fields are included in the data server's Include list box. You can continue to define the data server by excluding fields from the data server definition, if you wish.

### Excluding Fields

To exclude a field from use by the data server:

1. Click on the field name in the Include list box.
2. Click the Right arrow button to move the currently highlighted field to the Exclude list.

Fields that are in the Exclude list box will be inaccessible when using this data server.

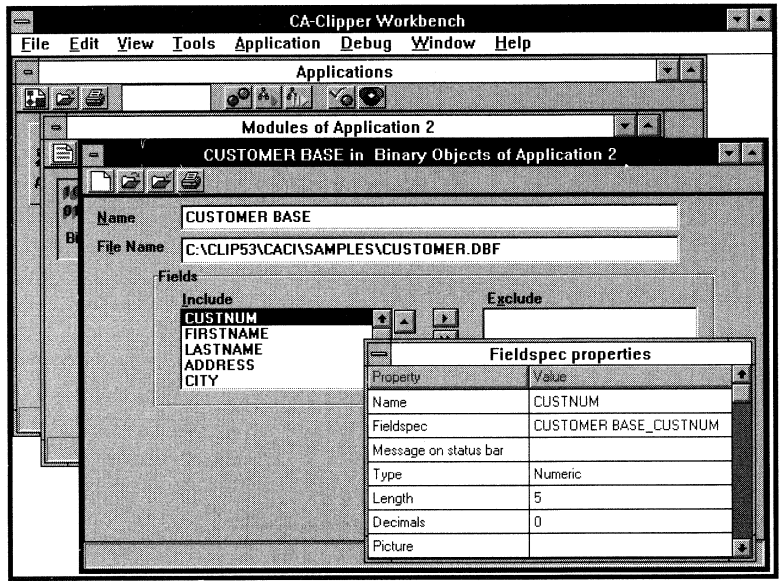
### Including Previously Excluded Fields

This process works the other way around using the Left arrow button to move a highlighted field from the Exclude list to the Include list.

**Tip:** As a shortcut, you can move all fields from one list to the other by clicking the appropriate double arrow button.

## Specifying Field Properties

Now you can proceed to specify each field's properties using the FieldSpec Properties window:




Fields can have the following properties in the DB Server Editor. Note that there are several *required* properties that must be specified. Otherwise these properties are optional.

Name

The name of the field. This property is required for every field.

FieldSpec

This name is used to create the FieldSpec entity. You can type in a value or choose one from this property's drop-down list box. If the field spec already exists, its associated properties are loaded into the Properties window.

The default is *<DataServerName>_<FieldName>* (for example, CUSTOMER BASE_CUSTNUM).

Message on Status Bar

The descriptive text to be used for this field in the specified field spec. This information will be displayed on the status line when this field has focus in your application at runtime.

| | |
|---|---|
| Type | The field's data type: Character, Numeric, Logical, Date, or Memo. This property is required for every field.<br><br>**Note:** The Type field cannot be set to a different value than that defined in the .DBF file. If a field spec is used in two different data servers, then the control of this field should be the same in their corresponding .DBF files. |
| Length | The length of the field. This property is required for every field.<br><br>**Note:** The Length field cannot be set to a different value than that defined in the .DBF file. If a field spec is used in two different data servers, then the control of this field should be the same in their corresponding .DBF files. |
| Decimals | The number of decimal places to be used for this field if it is numeric. Defaults to 0 for numeric data, unused otherwise.<br><br>**Note:** The Decimals field cannot be set to a different value than that defined in the .DBF file. If a field spec is used in two different data servers, then the control of this field should be the same in their corresponding .DBF files. |
| Picture | The picture clause to be used to format this field. This is a standard Xbase picture clause, such as "@!" or "999-99-9999". |
| Min Length | The minimum number of characters that can be entered in the field (for example, a state field can be defined with Length and Min Length of 2, while a password field can have a Length of 10 and a Min Length of 4). |
| Required | Select Yes or No, respectively, to indicate whether the field is required or not. |
| Minimum | A minimum value for a numeric or date field. |
| Maximum | A maximum value for a numeric or date field. |

Validation

The rule to be used to determine if entered data is valid. This is a condition, or logical expression, that must evaluate to TRUE before the data entered will be accepted in the field. You might, for example, use a table lookup method to determine if a key value is valid.

## Modifying a Data Server

After you have defined a data server, you can modify it by changing its properties or the properties of any fields associated with it. You can also change the data server name and its associated database file.

### Editing Data Server Properties

Name

To change the name of the data server, click on the Name edit control and type a new name.

File Name

To change the associated database file, click on the File Name edit control and either type a new name or import a new file using the Find button.

### Editing Fields

If there are fields in the Include list box, you can change their properties, change their order, or exclude them from use by the data server.

Changing Field Properties

To change any property associated with a field:

1. Click on the field name in the Include list box.

   The DB Server Editor highlights the field and updates the Properties window accordingly.

2. Click on the property that you want to change in the Properties window and specify the new value.

See Specifying Field Properties for more information.

### Ordering Fields

To change the order of the fields within the Include list box:

1. Click on the field name in the Include list box.
2. Click the Up arrow button to move the currently highlighted field up one position in the Include list, or click the Down arrow button to move the currently highlighted field down one position in the Include list.

The order in which the fields appear in the Include list box determines the order in which they will appear when you use the Auto Layout feature to define a data form for this data server. (See "Using the Form Editor" for information on Auto Layout.)



# Using the FieldSpec Editor

As discussed earlier in this chapter, FieldSpec entities allow you to associate additional properties with database fields without affecting the underlying database structure. They also allow multiple data servers to access the same property values for common fields.

The DB Server Editor generates a default field spec for each field included in the data server. You can modify any of these field specs using the FieldSpec Properties window. You can also choose an existing FieldSpec to associate with the current field using this window.

The FieldSpec Editor, on the other hand, provides a way to create and modify field specs *independent* of any field or data server with which they may be associated using an interface that is almost identical to the FieldSpec Properties window. The only difference is that the data server name of the field is not present.

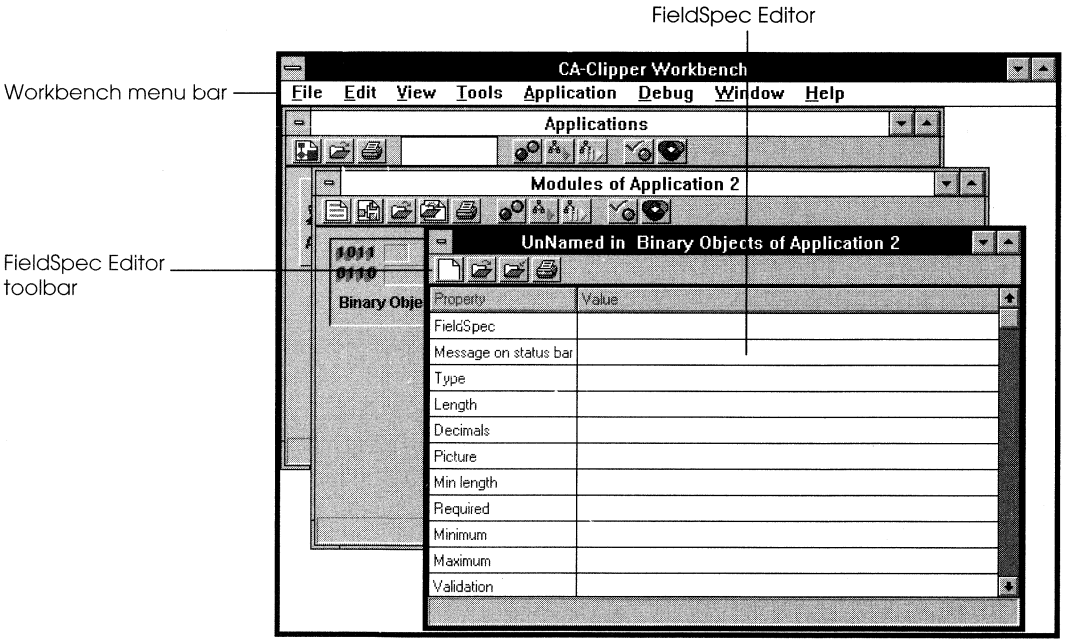
You can later associate the field specs that you create with the FieldSpec Editor with fields in a data server using the FieldSpec Properties window. If there are existing data servers in the system already using field specs that you modify, the changes that you make in the FieldSpec Editor will be automatically propagated to them.

This section describes the FieldSpec Editor and explains how to:

- Define a new field spec
- Modify an existing field spec
- Create a copy of an existing field spec and rename it

## The FieldSpec Editor Workspace

The Workbench's FieldSpec Editor has its own toolbar (which is identical to the DB Server Editor toolbar, described earlier), in addition to the menu commands on the Workbench's menu bar. When first loaded for a new FieldSpec entity, it looks as follows:



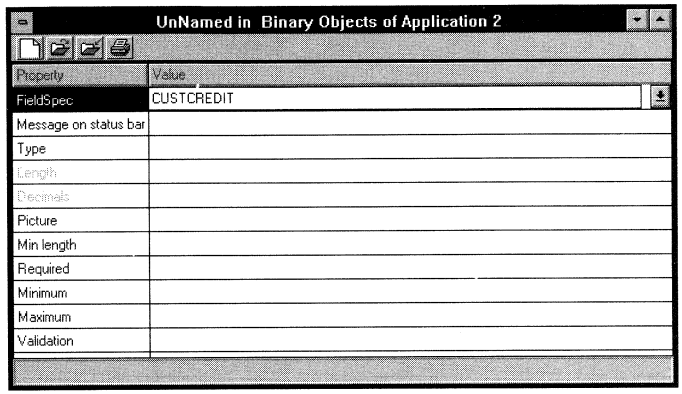
## Defining a Field Spec

To define a new field spec that will be stored in the Binary Objects module of the *current* application, you need to perform the following steps:

1. Start the FieldSpec Editor.

   The FieldSpec Editor is accessed using the Tools menu or the New Entity toolbar button from within the Binary Objects module.

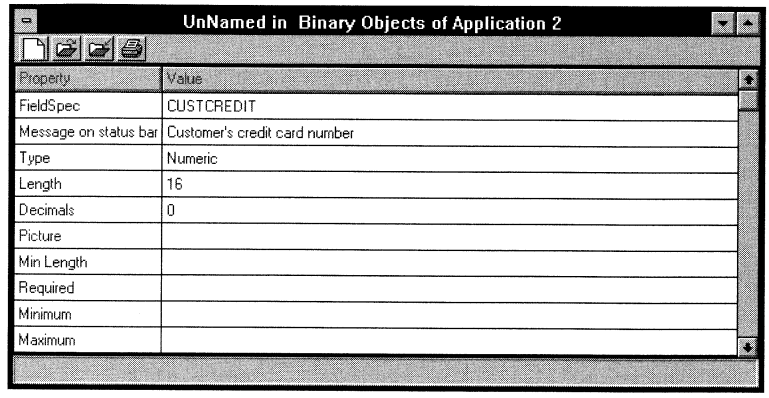
2. In the FieldSpec value cell, type the name of the new field spec (for example, **CustCredit**):




   You can enter a name using a maximum of 64 characters. The first character must be alphabetic or an underscore; the other characters can be alphanumeric and can include the underscore character. Blanks within the name will be changed to underscores and lowercase letters are changed to uppercase when you save the field spec.

   This name is used to create the FieldSpec entity, and to associate it with a field in a data server.

3. Enter the remaining properties for the field spec by clicking on the appropriate cell and specifying a value. For example:




   Note that Type and Length are required, and all other properties are optional. For a complete description of these properties, refer to Specifying Field Properties in the Using the DB Server Editor section earlier in this chapter.

4. Choose the Save toolbar button to save the field spec.

**Note:** This last step can be repeated whenever you make changes to a field spec and want to save your work without closing the FieldSpec Editor.

When you save a field spec, the Workbench automatically generates a FieldSpec entity in the Binary Objects module of the current application. You can double-click on this entity in an Entity Browser to begin editing it with the FieldSpec Editor.

## Editing Field Specs

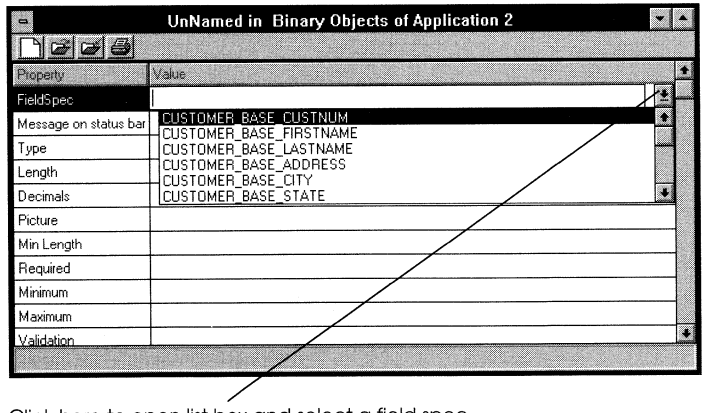
Within the FieldSpec Editor, you can edit, or modify, a field spec and copy properties from one field spec to another. Also, a field spec defined for the current application can be deleted just like any other entity.

### Modifying a Field Spec

To modify any field spec defined for the current application:

1. Start the FieldSpec Editor.
2. Click on the FieldSpec value cell.
3. Enter the name of an existing field spec.

   Alternatively, click on the Down arrow button to display a drop-down list box and choose the one you want:



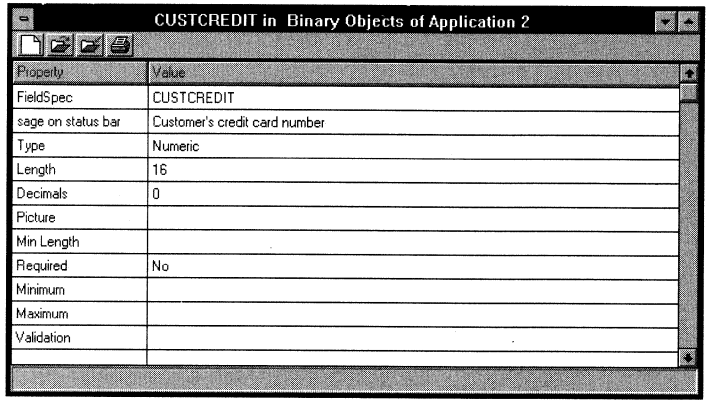

4. Change any property by clicking on it, and entering or selecting a new value.
5. Choose Save.

## Copying a Field Spec

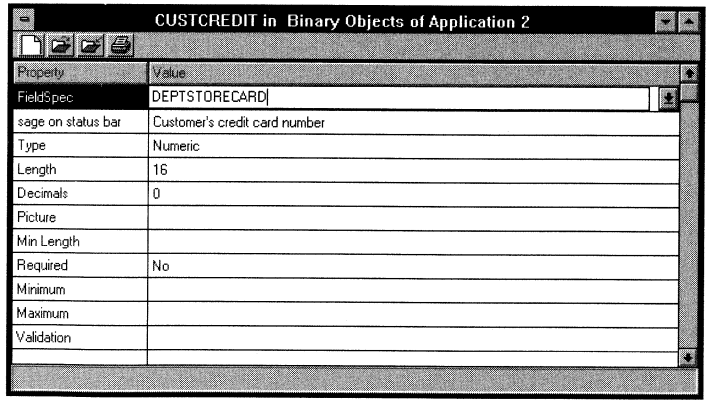
You can create a copy of a field spec and rename it easily using the FieldSpec Editor.

For example, if you want to create a *new* field spec called DeptStoreCard, copying the property values from our sample CustCredit field spec:

1. Load the CustCredit field spec following steps 1-3 in Modifying a Field Spec above:




2. Click on the FieldSpec value cell again, and replace CustCredit with **DeptStoreCard**:



3. Choose Save.

The properties for the existing field spec, CustCredit, are copied to the new field spec, DeptStoreCard, but will remain intact in the original. At this point, you can change some of the properties in the copy, if you wish.



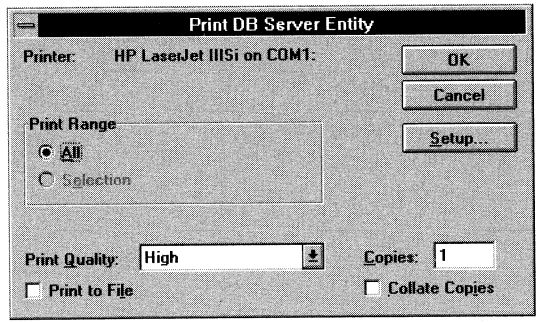
# Printing Data Servers and Field Specs

## Data Server Entities

To print both the data server properties and the field properties generated by a data server:

1. Choose the Print toolbar button from within the DB Server Editor:

   The Print DB Server Entity dialog box appears:



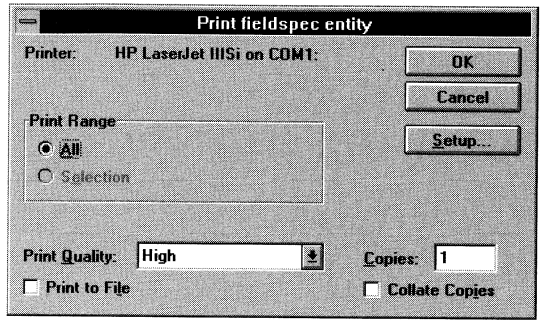

2. Specify your printing options.
3. Choose OK.

FieldSpec Entities

To print the properties for a field generated by a field spec:

1. Click on the Print toolbar button from within the FieldSpec Editor:

   The Print FieldSpec Entity dialog box appears:




2. Specify your printing options.
3. Choose OK.

For descriptions of the available standard printing options, see Printing an Application List in the "Browsing Applications, Modules, and Entities" chapter.

# Chapter 8
# Debugging Your Applications

## In This Chapter

The Workbench's debugger provides advanced tools for debugging your applications. You can set debugging on and off for each module, if you choose. This means that if you have a successful, stable application and decide to add new features to it, you can save invaluable time by testing and debugging only the new module.

Additionally, you can view an application's source code, global and local data, and database work areas *simultaneously* in separate windows.

This chapter is intended to help you recognize and resolve errors at every stage. In addition, you will be shown how to:

- Set debugging options at the application and module levels.
- Use the Error Browser and the visual indicators in the various browsers to pinpoint and resolve compiler errors.
- Recognize and resolve runtime errors using both the comprehensive runtime error reporting system that is built into every Workbench CA-Clipper application and the debugger for less obvious runtime errors.
- Use the debugger to track down and correct logic errors in your application.

# A Sample Debugging Application

Before showing you how to set debugging options and how to recognize and resolve errors in your own applications, you should first create our sample debugging application:

```
LOCAL cOldSalesRep, nTotal

USE Sales
SET INDEX TO Sales

DO WHILE ! EF()
    cOldSalesRep = Sales->Sale
    DO WHILE cOldSalesRep = Sales->SalesRep
        ? Sales->SalesRep Sales->Amount
        nTotal += Sales->Amount
        SKIP
    ENDDO
    ? "Total: ", nTotal, "for", cOldSalesRep
ENDDO

CLOSE
?
?
WAIT
```

This sample code is stored in the Sales.prg file in the CLIP53\CACI\DATA directory. Note that there are two additional files in the \CACI\DATA directory that are used in conjunction with this example—SALES.DBF and SALESREP.NTX.

The above code is an over-simplified example, designed with intentional errors to demonstrate the tools available for debugging an application in the Workbench. It is refined throughout the rest of this chapter as you walk through the various stages of debugging.

To create our sample application:

1. Create a new application, and optionally name it **Sales**.
2. Click on the New Module toolbar button.

   The Create Module dialog box appears.
3. Select PRG Module from the Module Type group box, if it is not already selected
4. Select the Create Module from Existing PRG or CH File option and choose OK.

   A standard Open dialog box appears displaying available .PRG files.
5. Open SALES.PRG from the CA-Clipper \CACI\DATA directory.

   You are returned to the Module Browser where a new module button named SALES appears.
6. Click on the Application Compiler Options toolbar button and select the following application compiler options: Debug Information, Create Default Procedure, and Enable Warnings. Also, select the Create .OBJ if Warnings and Cont. build if Warnings options.
7. Choose OK.
8. Click on the Application Linker Options toolbar button.
9. Type **SALES.EXE** in the .EXE File Name edit control and select SALES from the Main Module for Clipper App drop-down list box.
10. Type **CLD.LIB** in the Additional .OBJ Files edit box to include the CA-Clipper debug library.
11. Choose OK.

Now, use this sample application as you read through this chapter and work through the accompanying debugging examples.

# Setting Debugging Options

In the Workbench, you can set debugging options at the application or module level. This enables you to debug a new application piece-meal by setting the application-level debug option on, and selectively turning off the debug options for modules that you are not currently interested in debugging.

***Important!*** *Whenever you change a debug setting, you must rebuild the application to make it take effect. Because the system only rebuilds what you have changed, the rebuild is fast and efficient. You may also force the system to build your application* ignoring *date and time stamps. To do this, choose the Force Build command from the Application menu. However, the Force Build command should be used only under special circumstances.*

## Application Level

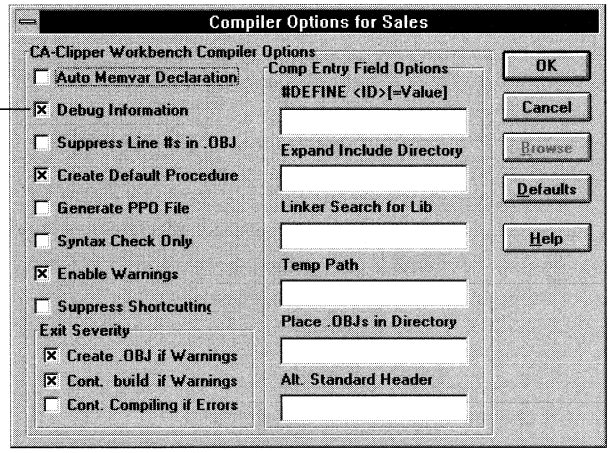
As discussed earlier in this guide, when you initially create an application, you can turn the debug option on or off. Each module within the application then automatically inherits the application's debug flag setting.

With the debug option turned on, the compiler automatically includes debugging information in each new module so that you can run the application using the debugger (via the Debug toolbar button) and, by default, debug any module defined in the application.

If, however, an application's debug option is not turned on, all modules will have debugging turned off. Furthermore, any attempt to run the application using the debugger will simply run it straight through without giving you the opportunity to use any of the debugger features.

**Note:** At any point during the lifetime of an application, you have the option of changing the debug option using the Application Compiler Options dialog box (accessed via the Application Compiler Options toolbar button or the Application Compiler Options menu command). See the Setting Compiler Options at the Module Level section in "Browsing Applications, Modules, and Entities" for more information about setting application compiler options.

An application's Debug Information option




***Important!*** *Note that after changing any application or compiler options, you must rebuild the application for the change to take effect.*

## Module Level

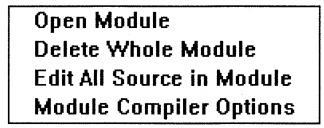
Since you may at times want to override the application debug setting for a particular module, the system allows you to change the debug setting at the module level. For example, if you are debugging an application and are not interested in debugging entities in a particular module, you can turn the module debug option off.

Conversely, you may have an application with the debug option off and want to turn on individual modules for debugging. For example, an application contains five modules: A, B, C, D, and E. You want to debug modules C and D only. To do this, you must change the Debug Information option for modules C and D using the Module Compiler Options dialog box. You must also enable the module options by selecting the Enable Module Options check box for *all* modules in the application in order for their respective options to take effect.

To change the debug option for an individual module from the Module Browser (for example, the SALES.PRG module in our sample Sales application):

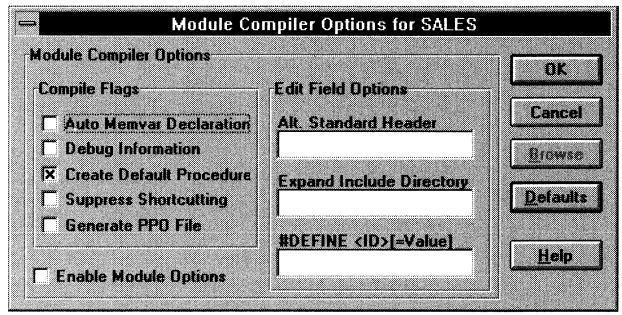
1. Highlight the module.
2. Click on the right mouse button.

   A local pop-up menu appears:




3. Choose Module Compiler Options.

   The Module Compiler Options dialog box appears:

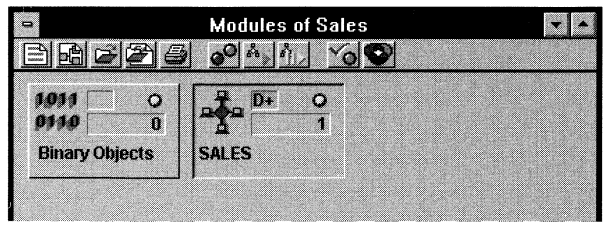



4. Select Debug Information from the Compile Flags group box.

   See "Browsing Applications, Modules, and Entities" for more information about the available compiler options.

5. Select the Enable Module Options check box.
6. Choose OK.

The SALES.PRG module button now displays a "D+" debugging indicator:



The debugging indicators "D+" and "D–" are used to denote that individual modules have been explicitly enabled or disabled, respectively. These indicators will appear on the module button only if the module-level compiler options are enabled. If not enabled, the application compiler options are in effect and no "D+" and "D–" indicators will be displayed. In other words, if the module is "blank," then the application options are in effect. This feature allows you to see at a glance which module level options are enabled. When you have an application with many modules, there is no need to clutter the screen needlessly with debugging indicators for every module.

# Using the Debugger



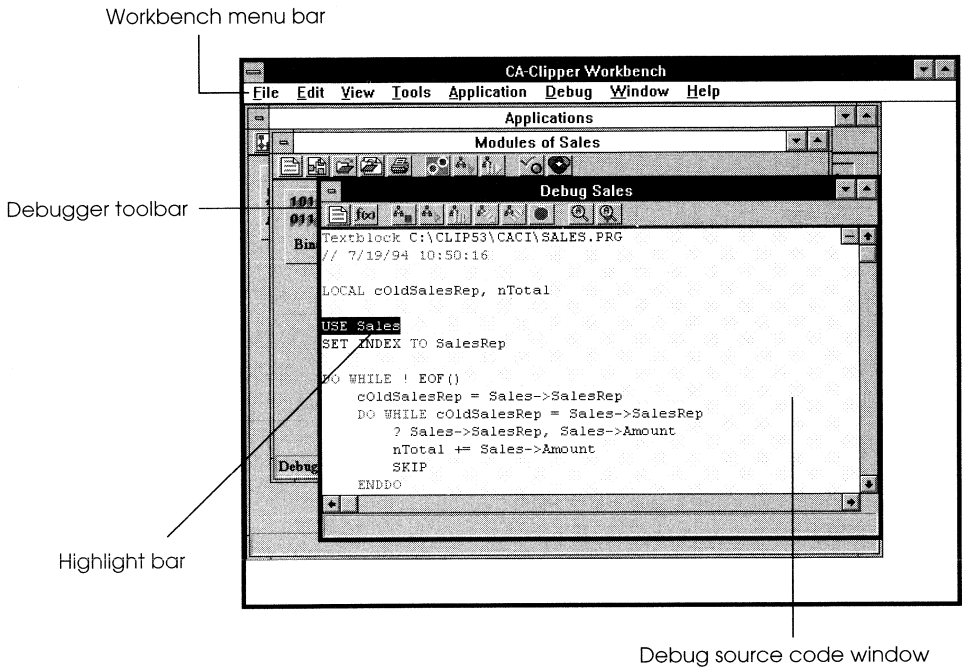
The debugger is the primary workspace in the IDE for tracking and correcting errors that occur at runtime. It can be accessed or started from almost anywhere, including the various browsers and editors.

The debugger is an advanced tool for debugging an application. Using it, you can:

- Use one of several execution modes to control the execution of your application while viewing the source code in the Debug source code window.
- Conditionally stop program execution using breakpoints, the AltD() function, or by pressing Ctrl+Break at any time.
- Monitor watch expressions in a separate window.
- Evaluate expressions on the fly.
- View and modify variables.
- View the call stack.
- View database and other work area information in a separate window and modify database field values.
- View and modify system settings.

## The Debug Source Code Window

The *Debug source code window* is the main debugger window in the Workbench. It is used to display the application's source code as you debug it. Note that when the Debug source code window is open, the commands on the Debug menu become active.




Debug Source Code Window

The Debug source code window owns all other windows that you can open using the Debug menu. Therefore, actions that you perform on the Debug source code window (such as maximize, minimize, or close) also affect the other debugger windows. Closing this window terminates the debugging process.

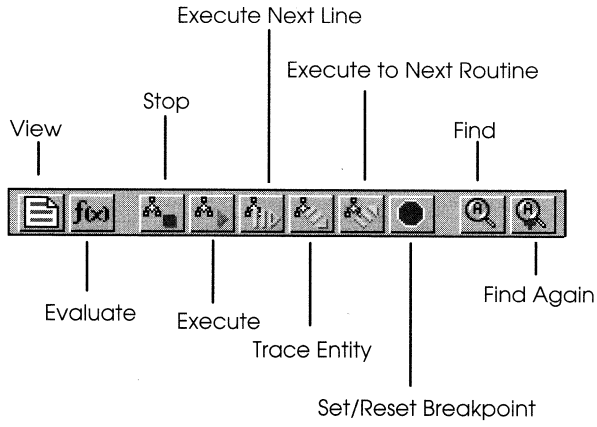
**Tip:** If there are other windows open when using the debugger (for example, other browsers and editors), you may want to minimize or rearrange those windows to reduce the clutter in the Workbench desktop. For example, choose the Window Tile menu command to conveniently arrange the remaining open debugger windows.

Highlight bar

The *highlight bar* in the Debug source code window indicates the current point of execution in the source code. You can scroll this window using the scroll bars and position the cursor using the mouse.

Toolbar

In addition to the Debug menu commands available in the Workbench's menu bar, the Debug source code window has its own toolbar. This toolbar contains the following buttons (which correspond to some of the commands available on the Debug menu):




All of these buttons, except Find and Find Again, are discussed in this chapter. See "Using the Source Code Editor" for information on searching for text.

**Tip:** For a quick description of these toolbar buttons, look at the status bar as the mouse pointer passes over the buttons. Also see your online Help reference.

## Execution Commands

The debugger allows you to run an application in several execution modes using various toolbar buttons and menu commands, as summarized in the following table:

| Menu Command | Toolbar Button |
|---|---|
| Run |  Execute |
| Step |  Execute Next Line |
| Step In |  Trace Entity |
| Next Routine |  Execute to Next |
| Step To Cursor | Not available in toolbar |
| Stop |  Stop |

Each execution command is described in more detail below.

**Note:** Use the Stop toolbar button to halt execution of the application.

### Execute

To execute an application in *run mode*, use the Debug Run menu command or the Execute toolbar button. The application returns control to the CA-Clipper Debug source code window when:

- A breakpoint is encountered
- You press Ctrl+Break to manually invoke the debugger
- The AltD() function is executed

Step

To execute an application in *single step mode*, use the Step command or the Execute Next Line toolbar button.

Single stepping lets you execute a single line of code at a time, viewing the output and examining variables as you go. Called entities are executed in run mode and are not displayed in the Debug source code window.

Step In

To step into an entity, or execute in *trace mode*, use the Step In command or the Trace Entity toolbar button.

Tracing an entity is similar to the single step mode in that it allows you to execute one line of program code at a time. However, the trace mode displays the code for called entities in the Debug source code window and allows you to single step through them.

Next Routine

The Next Routine command (or Execute to Next toolbar button) executes in run mode until the first line of the next procedure or function call.

Step To Cursor

The Step To Cursor command executes the application in run mode up to the current cursor position. It behaves as if a temporary breakpoint were set at the cursor.

**Note:** See Using Breakpoints later in this chapter for more information on breakpoints.

# Customizing Your DOS Window

Whenever a non-Windows application is started from within Windows, it searches for a Program Information File (PIF) which contains information that it can use to customize this non-Windows application.

When you select the Run command from the Debug menu or click on the Debug toolbar button, a DOS window is created automatically by the Workbench. Before creating this DOS window, however, Windows will search for a PIF in your Windows directory with the same name as your application. While using the Workbench debugger, CACI.PIF is used by default.

## Creating a PIF

You may want to create your own PIF using the PIF Editor provided by Microsoft Windows. To do this, follow these basic steps:

1. Choose the File New menu command in the PIF Editor.
2. Select the PIF options.

   See Debugging Considerations below for information about specific options that may affect debugging.
3. Save the PIF using your application's name with a .PIF file extension.

## Debugging Considerations

When selecting options for a new PIF, two in particular affect the debugging process: Display Usage and Execution.

### Display Usage

The PIF Editor's Display Usage option determines how an application is initially displayed—either Full Screen or Windowed. For debugging purposes, it may be a good idea to start your application in a window instead of full screen, so that you can view both the application and the debugger at the same time. To do this, simply click on the Windowed radio button under the Display Usage option.

### Execution Options

The PIF Editor's Execution options determine resource usage. The Background check box, if checked, allows the CA-Clipper application to run in the background while you are using another application.

The Exclusive option, if selected, gives exclusive use of resources to the current application; that is, no other application can run when this application is active.

Note that you may see a lapse in the display of your CA-Clipper application if this option is *not* checked. For example, while single-stepping through your code, you may notice that the output from the CA-Clipper application will be one line behind what the Debug window is showing as the currently executing code.

For more information about PIF files, please refer to your Microsoft Windows documentation.

# Detecting and Correcting Errors

With a Workbench application, detecting errors and correcting them is a three-stage process with which you are probably familiar as a software developer—the stages are similar to those that you may have gone through with other applications. The Workbench provides you with a comprehensive way to detect and resolve errors at every stage.

## Build Stage

In the first stage, when you attempt to build the application, you may receive *compiler errors* for such things as incorrect syntax usage. You also may receive linker errors for such things as unresolved external functions. Detection of certain error conditions at this stage can be controlled with compiler options. For example, whether the use of undeclared variable names is considered a compiler error is controlled by the Auto Memvar Declaration option. See "Browsing Applications, Modules, and Entities" for more information about setting compiler options.

The resolution of errors at this stage is fairly straightforward. You simply read the error message to pinpoint the cause of the error, locate the error, and fix it. However, in large applications with many modules and entities, there may be many errors at this stage and locating them in the source code can be difficult. To make locating compiler errors easy, the Workbench provides two tools: the Error Browser and visual indicators in the Entity, Module, and Application Browsers.

## Runtime Stage 1: Runtime Errors

After all compiler and linker errors are resolved, you will get a successful build for your application and can run it to see what happens. However, the first time you do this, the program may not run perfectly because of additional programming errors—*runtime errors*—that could not be detected at compile time. Some examples of runtime errors are a missing file, or a variable that is referenced before it is initialized.

**Tip:** You can minimize the errors that occur during this stage by strongly scoping and typing variables (and function and procedure declarations) and by *deselecting* the Auto Memvar Declaration compiler option. Doing this will trap any error involving the misuse of a variable or function call at compile time. See "Basic Concepts" in the *Programming and Utilities Guide* for more information about strong typing, scoping of variables, and calling conventions for functions and procedures.

Resolving errors at this stage is also not difficult. The application will display an error message to indicate exactly where the application failed and why, and you can resolve the problem appropriately. For example, make sure the missing file is properly located on the disk, or correct the source code by putting in an initialization statement for the variable in question. Occasionally, however, the particular cause of a runtime error message will elude you and you may need to use the debugger to track it down.



### Runtime Stage 2: Logic Errors

Once you have run the application through its paces and no runtime error messages are apparent, you may encounter additional errors in your application. These errors are known as *logic errors* and manifest themselves in less obvious ways than displaying an error message.

This final stage of error resolution, or *debugging*, is definitely the most difficult because the only way to detect logic errors is to recognize that the application is not behaving as intended. For resolving errors at this stage the Workbench provides you with the debugger, a comprehensive tool designed for just this purpose.

For a list of compile-time and runtime error messages and their descriptions, see Appendix A.

# Using the Error Browser to Resolve Compiler Errors

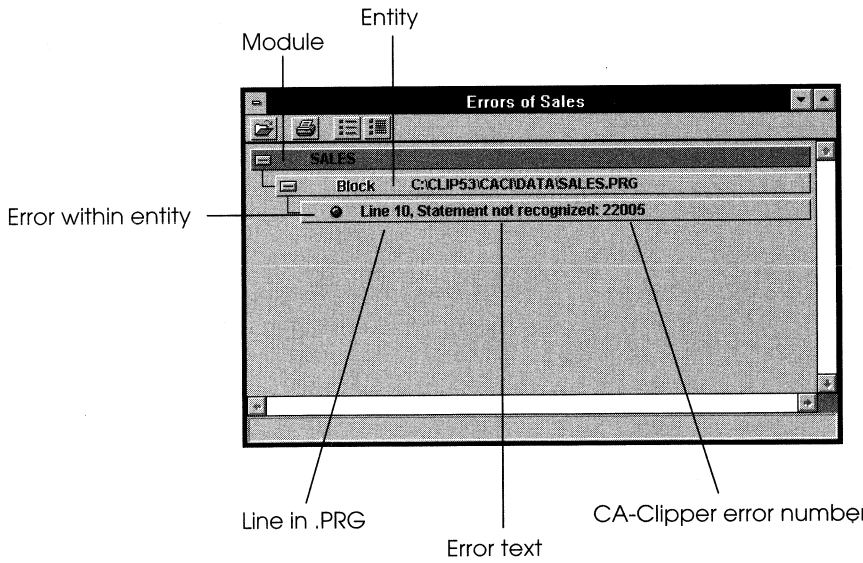
The Error Browser is the primary tool for locating *compiler* errors and warnings, and is automatically displayed after each build if there are errors. (If you close this window, you can reopen it at any time by choosing the Error Browser command from the Tools menu.)

1. From the Module Browser, build the sample application using the Build toolbar button.

   The CA-Clipper Workbench Link Diagnostics dialog box appears, indicating the type of error.

2. Choose OK.

   You are launched immediately into the Error Browser, which indicates a compiler error in line 10:

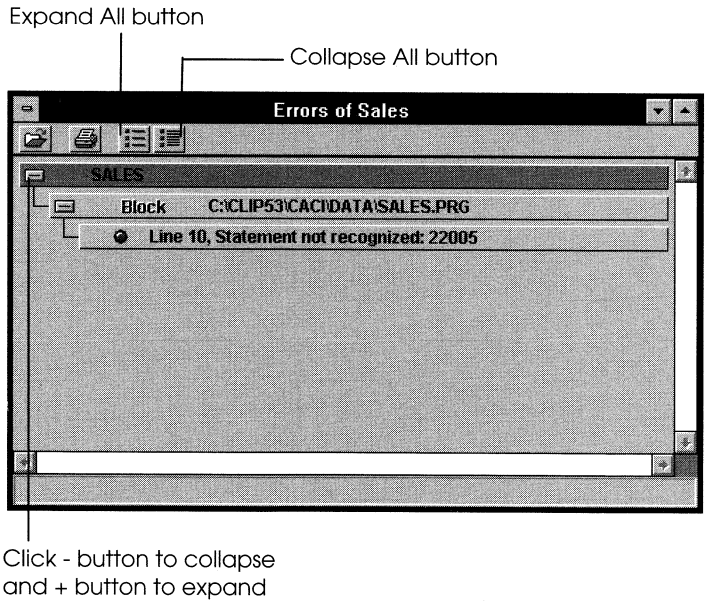



**Note:** The Error Browser also displays warning messages. The level of error warning messages can be changed via the Default Compiler Options dialog box.

## The Tree-Like Structure

The Error Browser displays errors in a tree-like structure that can be expanded or collapsed. In this structure, the first level indicates the module, the second level indicates the entity, and the third (and final) level indicates the line number and error message. This structure is a convenient mechanism for zeroing in on the error you want to correct.

Similar to the application-wide Entity Browser, you can manipulate the tree by clicking on the – / + button to collapse or expand, respectively, a particular level (or branch), and the Expand All or Collapse All toolbar button to expand or collapse the entire tree.

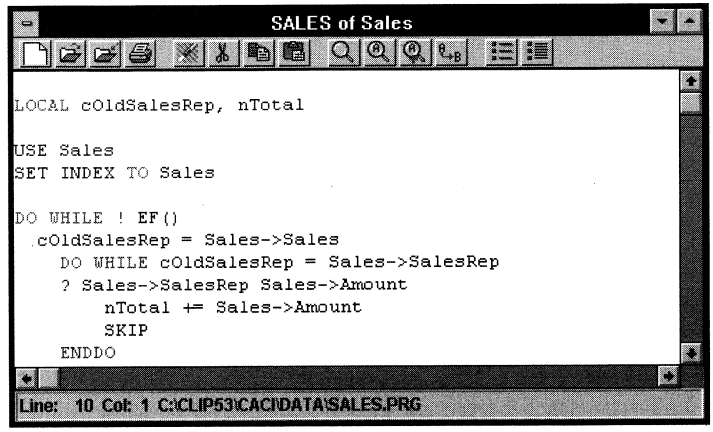



For example, if your application contains several modules, you might want to collapse all the branches, and then expand one module at a time to view and correct its errors.

## Getting to the Error

To immediately go to the source of an error, simply double-click on the error message in the Error Browser. This action takes you directly into the Source Code Editor, with the cursor located on the line where the error occurred.

For example, double-clicking on the first error message in your current Error Browser would open the Source Code Editor as follows:



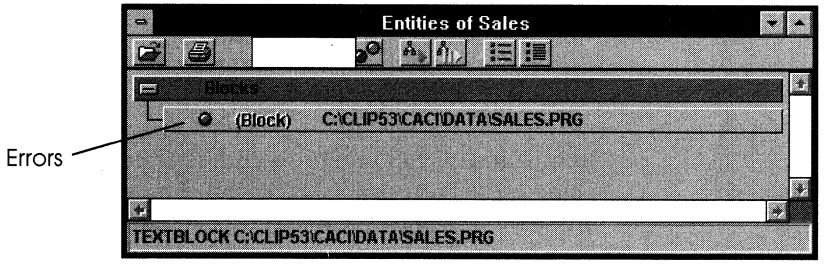

**Note:** Leave this window open as is—you will use it again in just a little while.

## Error Indicators in the Entity, Module, and Application Browsers

The Entity, Module, and Application Browsers have LED-style indicators to let you know the current status of any application component (that is, an entity, a module, or the application itself) as of the last build. In each of these browsers, *green* means the component has been compiled successfully; *red* indicates that there are compilation errors associated with the component; *blue* indicates the component has been compiled, with warnings; and *yellow* indicates that the build status is unknown, because the component is new or has been modified since the last build.

**Note:** If you have overridden the default Color LEDs option when setting up your system, compilation status is indicated respectively by a check mark , an "X", an exclamation point, or a question mark. (See Setting System Options in "Working in the Workbench" for more information.)

For example, choose the Tools Entity Browser menu command—you can see that the BLOCK entity in your sample application compiled with errors:

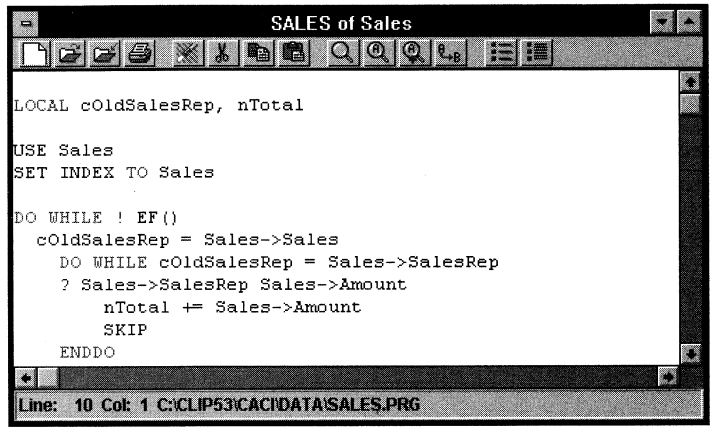



If there are red LED indicators (or X's) anywhere within an application, you can launch the Error Browser, discussed earlier, at any time (as long as one of the application's components has focus) to view the errors. The LED indicators give you a quick view of the status of your application, but cannot pinpoint the exact location of the error within an entity—you need the Error Browser for that.

When you are finished viewing the entities, close the Entity Browser window.

## Correcting the Error

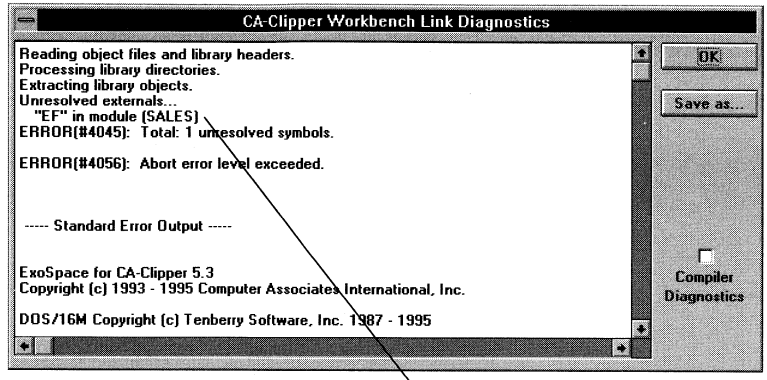
In our example, the problem is a syntax error. If you have been following along, you should have an open Source Code Editor window in which you are positioned at line 10:




**Note:** If this window is not currently selected, you can simply switch to it by clicking on the window with the mouse, or by choosing its name from the Window menu. If you already closed the window, repeat the steps outlined earlier in this chapter to redisplay it.

In this line of code, there should be a *comma* separating Sales->SalesRep and Sales->Amount. Add the missing comma and close the Source Code Editor window, saving your changes.

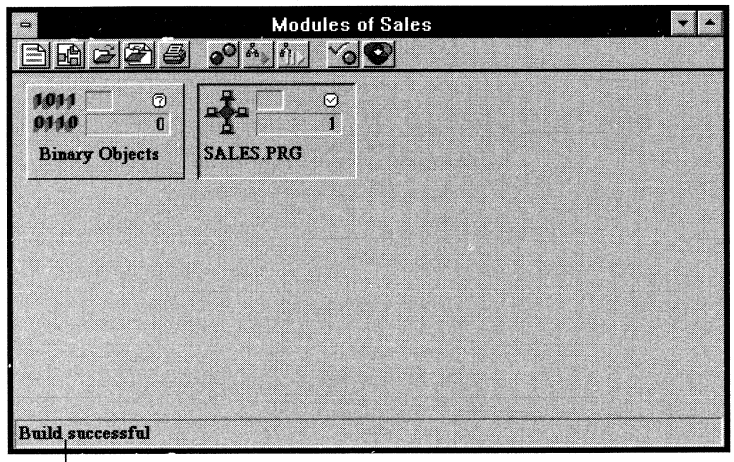
Next, close the Error Browser if it is still open, and then build the application again. The CA-Clipper Workbench Link Diagnostics dialog box appears, indicating that there is a link error due to the unresolved external "EF":




To correct this error:

1. Open the Source Code Editor.
2. Move the cursor to line 7.
3. Change EF() to **EOF()**, which is the correct function name for detecting end of file.
4. Choose Save.

Again, rebuild and run the application to see if it executes without error—this time, the status bar should indicate a successful build:




Successful compilation message

**Tip:** When trying to resolve compiler errors, you can use your online Help reference to quickly view the correct syntax and usage of the item in question.

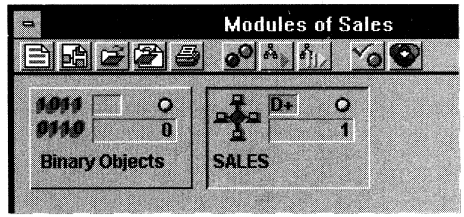
You have seen how the Workbench's debugging tools allow you to quickly pinpoint and resolve compiler and linker errors. Let us take a look at how runtime errors can easily be identified and resolved.

# Accessing the Debugger

To use any of the debugger features:

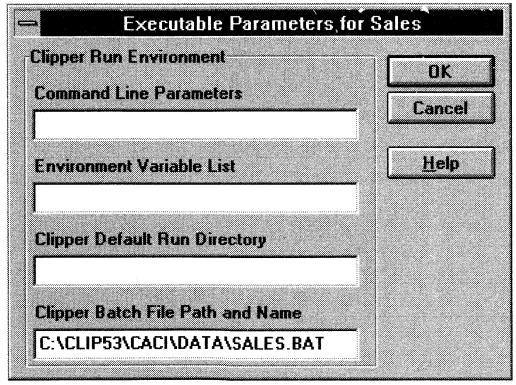
1. Build your application.
2. Run your application using the Debug toolbar button. Alternatively, choose the Run command from the Debug menu.

Debug toolbar button



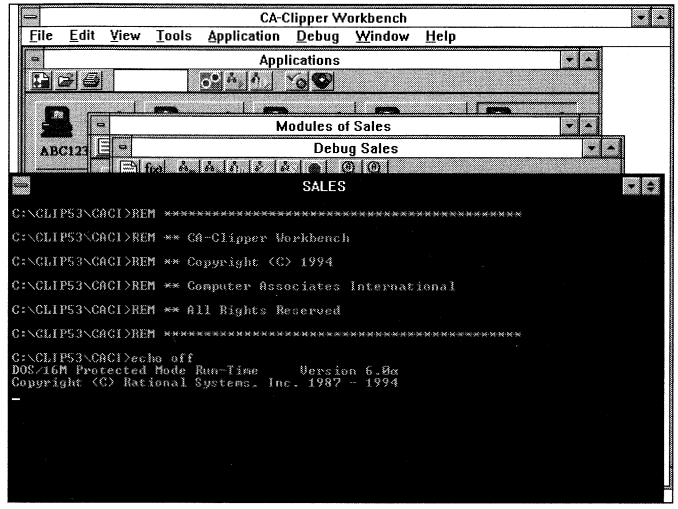

**Note:** To use Debug with an application, verify that the debugger library, CLD.LIB, is specified in the Additional Object Files edit control of the Application Linker Options dialog box. Additionally, the application's Debug Information option must be on. If it is not on, you can use the Application Compiler Options dialog box to turn it on, remembering to rebuild the application afterwards.

The Executable Parameters for Sales dialog box appears:



3. Choose OK.

   A DOS window is created showing your CA-Clipper Workbench application:




4. Press Alt+Tab or click on the Debug Sales window to reveal the Workbench's debugger.

**Tip:** You can arrange your windows so that you can view the DOS window and the debugger at the same time. For more information, see Customizing Your DOS Window later in this chapter.

5. Next, press the F5 key to attempt to run the program (instead of stepping through it). In the DOS window, you will get a typical runtime error message stating that a file—in this instance, SALES.NTX—cannot be found:


6. Choose Quit to exit the program and close the DOS window, pressing OK at the Program Terminated Normally prompt.
7. Close the Workbench debugger.

## Correcting Errors Using the Source Code Editor

There are no visual indicators for runtime errors in the Workbench nor are they recorded in the Error Browser. You should open the Entity Browser, locate the entity in question, and then launch the Source Code Editor to correct the problem.

To do this,

1. Choose the Tools Entity Browser menu command.
2. Find the entity in which the error is located—in your sample application, the error was in the only entity. Double-click on the entity to open it in the Source Code Editor.
3. Scroll to the documented line number.

   In your sample application, the error was in line 5. This line incorrectly refers to the SALES.NTX file—it should be SALESREP.NTX.
4. Change line 5 from

   ```
   SET INDEX TO Sales
   ```

   so that it reads:

   ```
   SET INDEX TO SalesRep
   ```
5. Close the Source Code Editor window, saving your work.

   The error is corrected.

Rebuild, execute, and then run the application by pressing F5 to see if it executes without error. This time you get a typical runtime error message stating that a variable cannot be found.


If you choose Quit and follow the steps above to load the faulty entity in the Source Code Editor, you will see that the following line of code is causing the error:

```
cOldSalesRep := Sales->Sale
```

The error message indicated that the Sale field could not be found, but unless you are familiar with the database file structure or documentation about the file is on hand, you may not know the correct field name. Using the debugger may be the quickest way to determine this type of information.

## Correcting Errors Using the Debugger

To debug your sample application:

1. Start the Workbench's debugger by clicking on the Debug toolbar button.

   The Executable Parameters dialog box

   The Workbench launches the debugger and also creates a DOS window on top of it where your CA-Clipper program will run. The CA-Clipper program, however, will be suspended before executing the first line of code.

2. Switch to the debugger by pressing Alt+Tab or clicking on the Debug Sales window.

   The Debug source code window appears.

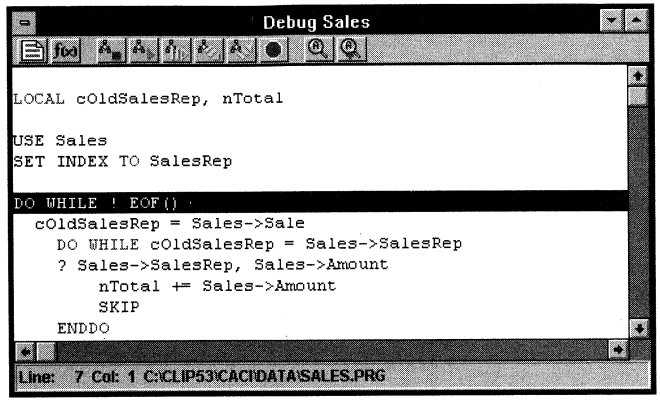
3. Move the cursor to line 7 in the application which reads:

   ```
   DO WHILE ! EOF()
   ```

   The reason for placing the cursor here is that line 9 is the line *before* the statement in which we know the field runtime error resides.

4. Choose the Debug Step To Cursor menu command.

The debugger runs the application up to line 7—*before* executing the line in which the runtime error was encountered—and then returns control to the Debug source code window with line 7 highlighted:



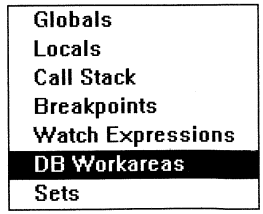

## Viewing Work Areas

Since the current error in your sample application has to do with a database field name, you can open the *Database Work Area window* to find out what is going on. This window allows you to view information about the databases open in the different work areas currently in use by an application.

To open the Database Work Area window:

1. Click on the debugger's View toolbar button.

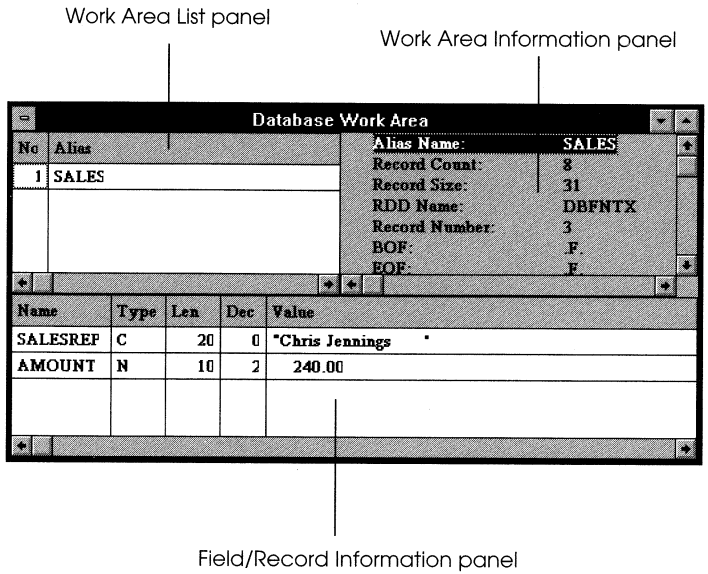
   A local pop-up menu appears:



2. Choose the DB Workareas command.

   Alternatively, choose the View DB Workareas command from the Debug menu.

   The Database Work Area window appears:




Note that at this point, SALES.DBF is in use by your sample application.

The Database Work Area window is divided into three separately scrollable panels: Work Area List, Work Area Information, and Field/Record Information.

Work Area List

The Work Area List panel displays a list of the work area number and alias for each database file currently open and in use by the application. You can view information about any other listed work area by clicking on it in this panel—the Work Area Information panel and the Field/Record Information panel are updated to reflect data for the newly selected work area.

Work Area Information

The Work Area Information panel displays a host of data about the selected work area, including beginning and end of file status, current record number, and filter.

Field/Record Information

The Field/Record Information panel displays the database file structure of the selected work area, including field names, types, lengths, and decimal settings. It also displays the contents of the current record.

Using the Database Work Area window, and in particular the Field/Record Information panel, you can see that there is no field named "Sale" in the SALES.DBF file—rather, it is called "SalesRep."

## Correcting the Error and More Debugging

Now that you have successfully isolated the cause of the original runtime error, you can correct it. To do so:

1. Switch back to and close the Debug source code window. (This action also automatically closes the Database Work Area window.)
2. Go to the Source Code Editor to change the field name in line 8 from Sales->Sale to **Sales->SalesRep**.
3. Close the Source Code Editor, saving your changes.

   The error has been corrected.

To continue testing your application, build and execute the application again (using the Execute toolbar button, not the debugger). You will see a typical runtime error message indicating that an operation is not supported. Specifically, the message indicates that there is an argument error involving the + operator in line 13:

```
nTotal += Sales->Amount
```

Choose Quit—you will use the debugger once again to find out more about the problem.

## Using Breakpoints

As your application becomes larger and more complex, it may be more difficult to remember how the flow of the application proceeds. For example, you may know that there is a bug in a certain module or even a specific entity, but stepping to it might take a long time.

Use a *breakpoint* to stop running the application at the line where you place the breakpoint, and to display the source of the entity containing that line in the Debug source code window.

Breakpoints are set from within the Debug source code window or by using the AltD() function in your application. (For more information about AltD(), see the *Reference Guide, Volume 1*.)

> **Tip:** Remember, you can also control which modules the debugger will step into using individual debug settings at that level. See Setting Debugging Options earlier in this chapter for more information.

In the sample application, you know that line 13 results in a runtime error, so you can instruct the debugger to run the program and stop before that line is executed. Once in the Debug source code window, you can set watch expressions, open one or more of the debugger's view windows, or even evaluate an expression to get a closer look at what is happening in the application.

## Setting, Viewing, and Clearing Breakpoints

As noted earlier, there is a problem at line 13 in your sample application, so a good place for a breakpoint would be line 9:

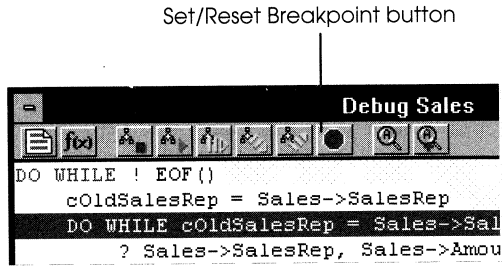
```
DO WHILE cOldSalesRep = Sales->SalesRep
```

To set a breakpoint at a particular line of code, the entity in which you want to set the breakpoint must be in the Debug source code window.

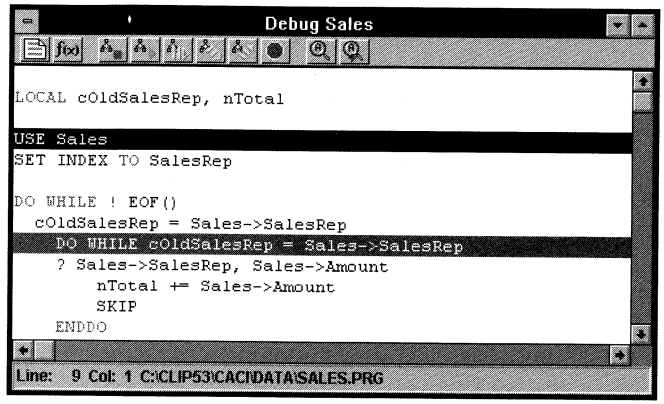
### Setting a Breakpoint

For example, to set a breakpoint at line 9:

1. If not already open, launch the debugger by choosing the Debug toolbar button.
2. In the Debug source code window, place the cursor on the desired line of code (in this case, line 9) by clicking on it.
3. Click the Set/Reset Breakpoint toolbar button (or use the Debug Breakpoint command:



The system color-codes the specified line, denoting that a breakpoint is set at that location:




**Tip:** You can set other breakpoints using this same procedure—just click in the desired line and then click Set/Reset Breakpoint.

### Setting a Breakpoint in an Entity Outside the Debug Source Code Window

You can also set a breakpoint in an entity which is *not* currently in the Debug source code window but is located elsewhere in the current application. To do this, you must open the entity using one of the browsers, bringing it into the debugger:

1. With the Debug source code window open, select the Entity Browser command from the Tools menu.

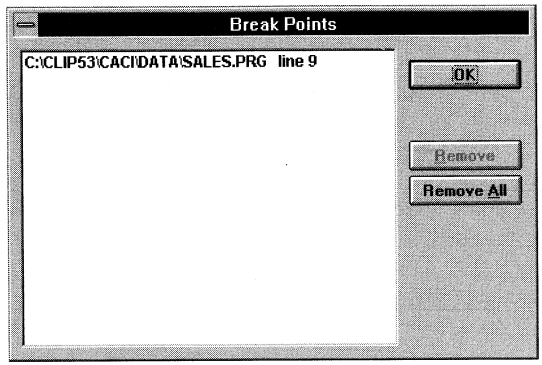
   The Entity Browser appears.

2. Double-click on the desired entity.
3. Return to the Debug source code window, where the specified entity has been opened.
4. Set a breakpoint in the newly opened entity.

### Viewing/Clearing Breakpoints

Note that the Set/Reset Breakpoint toolbar button used to set a breakpoint acts as a toggle switch—clicking it when the cursor is positioned in a breakpoint line clears that breakpoint.

You can also clear breakpoints using the View Breakpoints command on the Debug menu. Choosing this command displays the Break Points dialog box:




This dialog box allows you to:

- View all breakpoints currently set in an application
- Delete one or more breakpoints using the Remove or Remove All push buttons. (Remove deletes just the selected breakpoint, while Remove All clears all breakpoints.)

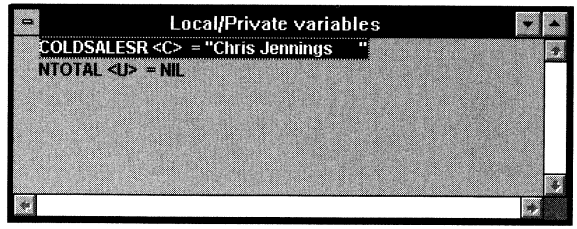
## Running with Breakpoints

Once the breakpoint is set, run the application from the debugger using the Execute toolbar button. It runs until the breakpoint is encountered and then returns control to the debugger.

## Viewing Local and Private Variables

To view local and private variables within an entity, choose the View Locals command from the Debug menu or select Locals from the View Window toolbar button.

The Local/Private Variable window appears.




From the values displayed in this window, you can see *nTotal* has a value of NIL at this point, indicating that it was probably not initialized.

## Modifying Local and Private Variables

By right-clicking on the current variable, you can access a local pop-up menu with the following commands for modifying variables:

- Modify

  This command allows you to change the value of the highlighted variable using the Modify Variable dialog box. (See Implementing a Temporary Fix for details.)

- Add Watch

  This command allows you to create a watch expression for the variable using the Add Watch Expression dialog box. (See Using Watch Expressions later in this chapter.)

**Tip:** This local pop-up menu is also available for variables displayed in the Global/Public Variable window. (For more information, see Viewing Global/Public Variables later in this chapter.)

## Implementing a Temporary Fix

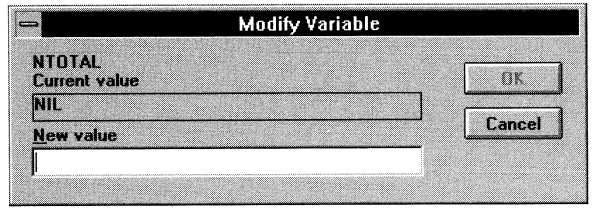
You can implement a temporary correction for our last error from within the debugger's View Local/Private Variable window. This temporary fix will enable your program to run beyond the line of code in question. To do this, you will initialize *nTotal* to a value of zero and continue executing the program:

1. Highlight *nTotal* in the View Local/Private Variable window by clicking on it.
2. Press the right mouse button.

   A local pop-up menu appears:


3. Choose the Modify command from the local pop-up menu.

   The Modify Variable dialog box appears:




4. Type **0** in the New Value edit control.
5. Choose OK to close the dialog box and modify the variable.

   **Note:** The fix that you just entered is not logically correct because it introduces an accumulation error in the total for each sales representative, but it will get you beyond the error and allow you to view your program results. Later, you can properly correct the error in the source code.

6. Return to the Debug source code window and click on the Execute toolbar button to run the remainder of the application.

   The DOS window now displays the results of the program. Note, however, that the total amount for each sales representative is getting bigger and bigger because the program does not reset *nTotal* to zero for each sales representative.

7. Click on the Debug source code window or, alternatively, press Alt+Tab to return to the Workbench. Then implement a permanent fix, described below.

## Correcting the Final Error

This is the final error in the sample application, and you can now correct it. Close the Debug source code window, go to the Source Code Editor, and enter the line **nTotal := 0** as a new line 9. The debugged program should look like this:

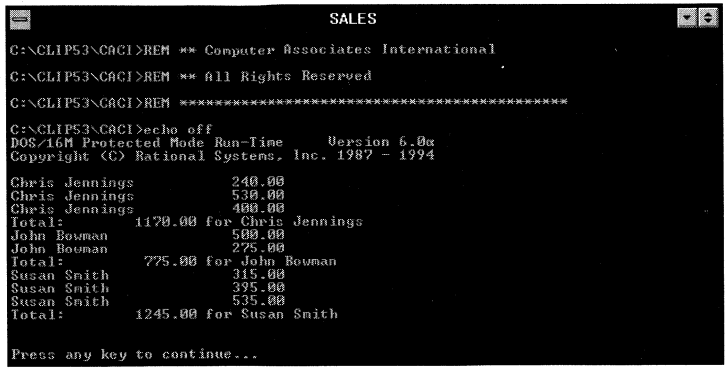
```
LOCAL cOldSalesRep, nTotal

USE Sales
SET INDEX TO SalesRep

DO WHILE ! EOF()
   cOldSalesRep = Sales->SalesRep
   nTotal := 0
   DO WHILE cOldSalesRep = Sales->SalesRep
      ? Sales->SalesRep, Sales->Amount
      nTotal += Sales->Amount
      SKIP
   ENDDO
   ? "Total: ", nTotal, "for", cOldSalesRep
ENDDO

CLOSE
?
?
WAIT
```

Now, rebuild and run SALES.PRG once more by pressing F5 when in the debugger. The result is:

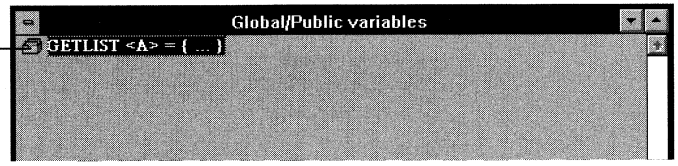



Even though there are no more errors, the remaining features of the debugger will be demonstrated using this program, so rebuild it and run it using the debugger as you have already done several times.

## Viewing Global and Public Variables

There is a separate window available for viewing globals and public variables. Restart the application using the debugger. Next, open the Global/Public Variables window by selecting the View Globals command from the Debug menu.

The Global/Public Variables window appears:




Now you can proceed to modify a variable or set a watch expression from this window, if you wish. The behavior of this window is identical to the View Local/Public Variables window discussed earlier in this chapter.

**Tip:** Click on the *File icon* for more detailed information about a global variable. This information is displayed in a collapsible/expandable tree-like structure.

## Using Watch Expressions

The debugger allows you to define as a *watch expression* any expression whose value you want to monitor during execution *except* one involving dimensioned arrays.

Each watch expression is evaluated according to the current scope of the program when it stops during debugging, and the expression's current value is updated.

**Note:** Watch expressions are automatically evaluated whenever execution stops. They are not, however, updated continually as the program runs.

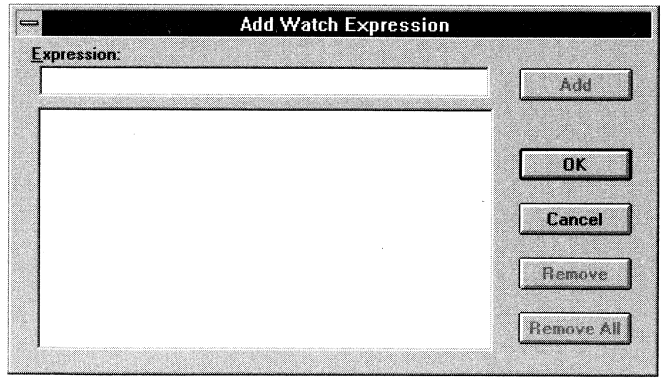
### Setting and Clearing Watch Expressions

Setting

To set a watch expression that you can subsequently view in the Watch Expression window:

1. Choose the Debug Watch command.

   The Add Watch Expression dialog box appears:



2. In the Expression edit control, enter the expression to be monitored (for example, **nTotal**).
3. Choose Add to add it to a list of watch expressions.
4. Choose OK to close the dialog box and save the list of watch expressions.

   Repeat steps 3 and 4 to create as many watch expressions as you like.

Clearing

To clear one or more watch expressions:

1. Highlight the expression you want to remove, and click on the Remove button.
2. To clear all watch expressions, click on the Remove All button.
3. Choose OK to close the dialog box and save the changes you have made to the list of watch expressions.

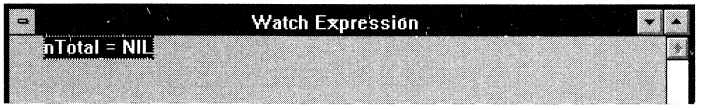
**Tip:** You can also create watch expressions from variables in the Local/Private Variables and the Global/Public Variables windows by selecting the Watch Variables command from the Debug menu.



## Viewing Watch Expressions

To view the current value of all watch expressions, use the debugger's View toolbar button and then select the Watch Expressions command from the local pop-up menu.

The Watch Expression window appears:




To watch the contents of this window change as the expressions are updated by the application, click on the Debug source code window and single step through the code.

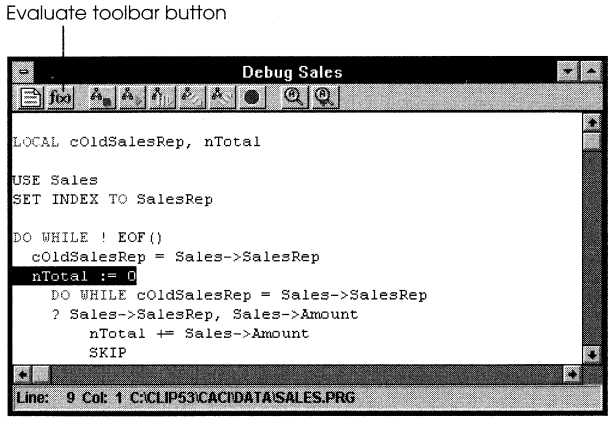
**Tip:** You may want to tile the Debug source code and Watch Expression windows so you can view them simultaneously.

## Evaluating Expressions

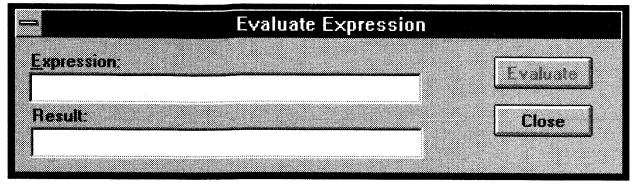
You can evaluate any expression (except one involving macros) from within the debugger using either the Expression command on the Debug menu or the Evaluate toolbar button. This is a convenient way to check a particular value without going to the trouble of setting it up as a watch expression. You can also create and change variables using the assignment operator.

To evaluate an expression using our sample debugging application:

1. Restart the debugger.
2. Step through the application to line 9.
3. Click the Evaluate toolbar button:

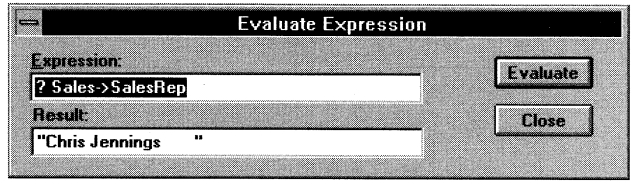


The Evaluate Expression dialog box appears:




4. Enter an expression in the Expression edit control (for example, **? Sales->SalesRep**).
5. Choose Evaluate.

The Workbench evaluates the expression, displaying the status of the evaluation process, and later the results of the calculation in the Result edit control:




***Important!*** *The expression you specify is evaluated in the context of the current application (and the libraries and DLLs in its search path). You will receive an error message if the expression cannot be evaluated.*

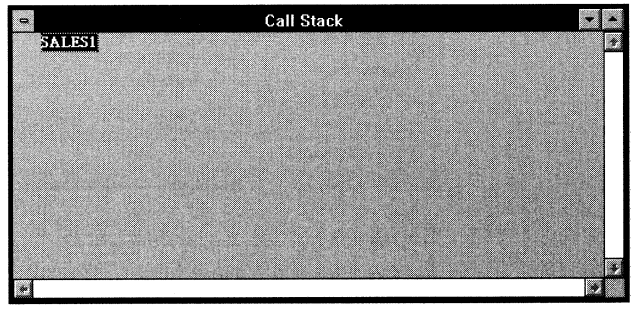
## Viewing the Call Stack

A *call stack* is a list of all activations that are currently pending on the stack. When a routine, such as a function or procedure, is called, it is added to the stack where it remains until it returns control to its caller. You can view the call stack using the Call Stack window.

To view the call stack for our sample Sales application:

1. Click on the debugger's View toolbar button.
2. Select the Call Stack command from the local pop-up menu.

   The Call Stack window appears, displaying the most recently called function first:



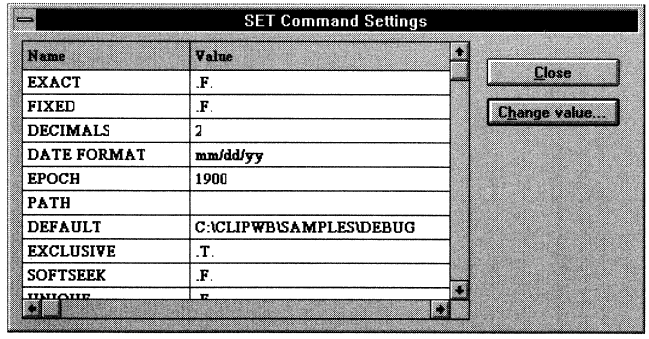
## Viewing Sets

You can also view and modify the Set/Environment settings for your application using the SET Command Settings dialog box. This dialog box displays the current settings of the various Set/Environment commands/functions, and allows you to temporarily change them during the current debugging session.

For example, if you have numerical output that is clearly not what you expected, you may want to increase your decimal setting as an additional test of your program's logic as you debug the balance of the application. To do this:

1. Click on the debugger's View toolbar button.
2. Select the Sets command from the local pop-up menu.

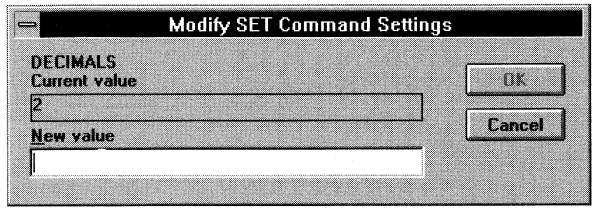
   The SET Command Settings dialog box appears:




3. Highlight the name of a command or function (for example, DECIMALS).

4. Click on the Change Value push button.

   The Modify SET Command Setting dialog box appears:




5. In the New Value edit control, enter a value (for example, **10**).
6. Click OK.

   You are returned to the SET Command Settings dialog box.
7. Click on the Close push button.
8. Step through the remainder of your application.

   If the subsequent numerical output for this debugging session is reasonable, then you know that you should correct the decimal setting in the application's source code.

# Appendix A
# Workbench File Types

## Overview

This appendix lists the different types of files generated or used by the CA-Clipper Workbench utility.

## Supported File Types

The supported file types follow in alphabetical order:

| File Type | Description |
|---|---|
| .APP | Internal repository file |
| .CDX | FoxBase-compatible, compressed index file |
| .CEF | External application file |
| .CH | Xbase include file |
| .CW | Internal repository file |
| .DBF | Xbase database file |
| .EXE | Executable program |
| .IND | Internal repository file |
| .MDX | dBASE IV index file |
| .NDX | dBASE III+ index file |
| .NTX | Index file |
| .PRG | Source code |

# Appendix B
# Using Linker Template Files

## Overview

Under most circumstances, defining options via the Linker Options dialog box and using the default linker are sufficient. However, for complete flexibility and control over the linking process, you can use a linker template file to define the exact commands that are executed during the link phase of an application build. For example, you might use a template file if you prefer a linker other than the default one, if the complexity of one or more linker commands exceeds the capacity of the Linker Options dialog box, or if you want to invoke a utility other than a linker, such as the Microsoft Librarian.

To use a template file, you check the Use Linker Template check box in the Linker Options dialog box. Then, using special template commands and symbols, you provide the information necessary to invoke the linker (or whatever utility you choose). The Template push button in the Linker Options dialog box opens an editor for the purpose of creating the template file.

Once you have defined a linker template file for an application and have designated (via the Use Linker Template check box) that the application is to use the template file, the linking process for the application is strictly defined by the contents of the template file, overriding any defaults that the system would otherwise use to link the application.

Each time you build, the template file is used, along with application and module information, to construct a batch file defining the DOS command(s) necessary to invoke the linker (or other utility) and a script file that serves as input to the linker. Then, the batch file is executed under COMMAND.COM to get the link process started and, normally, the command in the batch file that invokes the linker instructs it to take input from the generated script file.

**Note:** The WORKING entry in the AutoMake section of CACI.INI (located in your main Windows directory) defines the directory location for all files generated as a result of choosing to use a linker template file. This includes batch files, script files, error log files, and any other files generated by the system.

A command language is available to write commands to either the batch file or the script file, and to control the processing logic used to generate these files. Application and module data are available via special symbols that you use in the template file. Symbols are replaced by actual application or module data during the generation of the batch and script files. This appendix describes the commands and symbols that you can use in a linker template file.

**Note:** Although the symbols and commands documented in this appendix are shown in mixed case for clarity, interpretation of them is without regard to case.

# Symbols

Symbols can represent string, string list, or numeric data. Certain numeric symbols are restricted to values of zero (0) and one (1) and are, therefore, used to represent Boolean data.

## Linker Information Symbols

Linker information symbols allow you to pick up certain pieces of key information that you may need in order to perform the link operation. For example, there are symbols for the list of object files that make up an application and for the name of the map file. The information, for the most part, is defined using the Linker Options dialog box. (Refer to "Working in the Workbench" in this guide for more detailed information on any of the dialog box controls mentioned here.)

__ClipperEnv — A string symbol representing the value you enter into the Embed Environ. List in .EXE edit control of the Linker Options dialog box.

__ClipperName — A string symbol representing the name you enter into the Altered CLIPPER Env. Var. edit control of the Linker Options dialog box.

__ClipperOver — A numeric symbol representing the status of the Environment Override check box of the Linker Options dialog box. A value of 0 means it is not checked, and a value of 1 means that it is checked.

__ClipperTools — A numeric symbol representing the status of the Use Clipper Tools check box of the Linker Options dialog box. A value of 0 means it is not checked, and a value of 1 means that it is checked.

__IgnoreError — A numeric symbol representing the status of the Ignore Errors check box of the Linker Options dialog box. A value of 0 means it is not checked, and a value of 1 means that it is checked.

| | |
|---|---|
| __Lib | A string list symbol representing the library names you entered in the Libraries edit control of the Linker Options dialog box. |
| __LogFile | A string symbol representing the name of the error log file for the build process. The contents of this file is automatically displayed after a bad build, but if for some reason you need to view it independently, the error log file is named AUTOMAKE.EL. |
| __Map | A numeric symbol representing the status of the Create Link Map File check box of the Application Map Linker Options dialog box. A value of 0 means it is not checked, and a value of 1 means that it is checked. |
| __MapFile | A string symbol representing the name you enter into the Link Map File Name edit control of the Application Map Linker Options dialog box. |
| __MapSegs | A numeric symbol representing the status of the Segments with Assigned Addresses check box of the Application Map Linker Options dialog box. A value of 0 means it is not checked, and a value of 1 means that it is checked. |
| __MapSort | A string symbol representing the radio button you selected in the Sort Properties group box of the Application Map Linker Options dialog box ("N" for Publics Sorted by Name, and "A" for Publics sorted by Address). |
| __Mod | A string list symbol representing the modules you entered in the Module <obj> From <lib> edit control of the Linker Options dialog box. |
| __NoDefLibs | A numeric symbol representing the status of the No Default Libs check box of the Linker Options dialog box. A value of 0 means it is not checked, and a value of 1 means that it is checked. |

| | |
|---|---|
| __Obj | A string list symbol representing the list of all object files—except the main one—that are part of this application. This value is not taken from any particular control in the Linker Options dialog box. Instead, the system determines it based on the knowledge of all modules in the application (one per object file) and excluding the __ObjMain file. |
| __ObjExtra | A string list symbol representing the list of object files you enter into the Additional Object Files edit control of the Linker Options dialog box. |
| __ObjMain | A string symbol representing the name you enter into the Main Module for Clipper App combo box of the control in Linker Options dialog box. |
| __ObjNew | Because applications are compiled on an as needed basis in the Workbench, not every module needs to be compiled for every build—it depends on whether or not that module has changed since the last build. __ObjNew is a string list symbol representing the list of object files—except the main one—that were created as part of this build. |
| __OutFile | A string symbol representing the name you enter into the Executable File Name edit control of the Linker Options dialog box. |
| __Packages | A string list symbol representing the check boxes you selected in the Linker Package Options dialog box. |
| __ProcDepth | A numeric symbol representing the value you entered in the Maximum Procedure Depth edit control of the Linker Options dialog box. |
| __ScriptFile | A string symbol representing the name of the script file generated by this linker template file. |
| __StackSize | A numeric symbol representing the value you entered in the Stack edit control of the Linker Options dialog box. |

## Diagnostic Symbols

There are also some diagnostic symbols that you can set manually and that are set automatically based on certain syntactic errors in the template file:

__Error — A string symbol that is set manually by the #error command, or because of some error condition in the template file.

__Warning — A string symbol that is set manually by the #warning command, or because of some warning condition in the template file.

## Runtime Symbols

In addition to the predefined symbols mentioned so far, you can create string and string list symbols at runtime. See the #assign command under Template Commands later in this appendix for more information and examples.

**Note:** You can use an environment variable name in a linker template file anywhere that a string symbol is appropriate. For example, to create a runtime string symbol using the CLIPPER environment variable:

```
#assign ClipperValue = "$(CLIPPER)"
```

## Symbol Usage

You can use symbols in your linker template file in various ways.

### Substitution

To substitute the contents of a symbol directly into a batch or script command, use the dollar sign ($) followed by the symbol name in parentheses.

Strings

For string symbols, the $(<*Symbol*>) expression is replaced by the value of the symbol or the null string if it is undefined. Consider the following template file lines as an example:

```
IF EXIST $(__OutFile) DEL $(__OutFile)
EXOSPACE @$(__ScriptFile) >> $(__LogFile)
```

The corresponding generated commands would look something like this:

```
IF EXIST ACCOUNTS.EXE DEL ACCOUNTS.EXE
EXOSPACE @accounts.lnk >> automake.el
```

**Note:** When string or string list symbols containing file names are substituted, the full path of the file is written to the corresponding batch or script file. For readability, however, the examples in this appendix omit path names.

String Lists

For string list symbols, the treatment of the $(<*Symbol*>) expression varies depending on its location in the template file.

Inside a generated script section (defined by the #gen command), the $(<*Symbol*>) expression is replaced by a single element in the list. In a generated script section, a list is processed in its entirety, one element at a time. Consider the following line as an example:

```
#gen __Mod
    MODULE $(__Mod) FROM $(__Mod)
#endgen
```

The corresponding generated command would look something like this, in which two modules are taken from one library and a third is taken from a second library:

```
MODULE Module1 FROM Lib1
MODULE Module2 FROM Lib1
MODULE Module3 FROM Lib2
```

Outside a generated script section, it is replaced by a separated list. The default separator is a comma, but you can change it using the #sep command. Consider the following line as an example:

```
EXOSPACE PACKAGE $(__Packages)
```

The corresponding generated command would look something like this:

```
EXOSPACE PACKAGE DOS25, INT10, NET5C
```

Numerics

For numeric symbols, the $(<*Symbol*>) expression is replaced by the decimal ASCII value of the symbol. Consider the following line as an example:

```
STACK $(__StackSize)
```

The corresponding generated command would look something like this:

```
STACK 4096
```

## Decision Making

Symbols can also be used in a decision making process with the #if command. The following table summarizes when the various symbol types are considered true:

| Symbol Type | True if... |
|---|---|
| Numeric | Value is not zero |
| String | String is not null |
| String list | List is not empty |

Refer to the #if command in the Template Commands section later in this appendix for more information and examples.

## File Specification Components

Normally, when a string contains a file name, the file name is substituted using its full path name (for example, $(__OutFile) may produce something like C:\CLIP53\EXE\ACCOUNTS.EXE). You may find occasions, however, when you need to extract components of the file specification.

The following table summarizes expressions that you can use in a template file to extract various portions of a file specification:

| Expression | Returns |
|---|---|
| $B(<*StringSymbol*>) | Drive and directory |
| $D(<*StringSymbol*>) | Base file name |
| $F(<*StringSymbol*>) | Base file name and extension |
| $R(<*StringSymbol*>) | Drive, directory, and base file name |

The following example shows a $R expression used to create a map file name if none is specified:

```
#if __Map
#if __MapFile
    MAP = $(__MapFile)
#else
    MAP = $R(__OutFile).MAP S, A
#endif
#endif
```

## Using the Escape Symbol and Other Special Characters

Certain characters have special meaning within a template file. For example, $ is used to substitute symbols and the double-quote character (") is used in quoted strings. To process these as normal characters, the linker template allows you to *escape* them (that is, preface them with a special character called an *escape* symbol). The default escape symbol is ^, but you can change it using the #escape command.

For example, to assign a runtime symbol in which the string contains a double quote:

```
#assign QuotedSymbol = "This is a ^"quoted^" symbol"
```

There are also certain cases in which a new line (that is, a carriage return/linefeed pair) must be manually written to the batch or script file. In these cases, use the escape symbol followed by an "n". See the #more command under Template Commands in this appendix for an example using ^n.

# Template Commands

Template commands provide information, delimit template sections, and specify conditional processing. The first character of a template command must be the pound (#) character or the character specified by the #template command.

#assign <*statement*>

Creates a string or string list symbol at runtime. The <*statement*> defines the symbol name first, followed by an assignment operator (i.e., = for assignment and += for compound assignment), followed by a string or string list symbol or a quoted string.

The following example assigns a string symbol representing the main object file, then augments the newly created symbol using the string list symbol representing the other object files in the application. The result is the runtime symbol, *allObjs*, which will be a string list symbol:

```
#assign allObjs  = __ObjMain
#assign allObjs += __Obj
```

The next example defines *MySymbol* using a literal string, then replaces its current value by reassigning the same symbol name with a different value.

```
#assign MySymbol = "alpha"
#assign MySymbol = __LogFile
```

#batch
...
#endbatch

Delimits the *batch section* of the template file in which the contents of the generated batch file are defined. After the template file is processed and the batch and script files generated, the batch file is executed under COMMAND.COM. Therefore, to get the linking process going, each template file must have a batch section in which the linker (or other utility) is invoked. In most cases, the utility will be invoked using a script file generated by the same template file. For example, the following batch section defines a command line to invoke the Microsoft Librarian:

```
#batch
LIB $(__OutFile) @$(__ScriptFile); >> $(__LogFile)
#endbatch
```

#comment "<*string*>"

Normally, template commands and blank lines that are part of the linker template are not written to the target script file. However, if #comment appears, template commands and blank lines are written to the script file as comments, using the comment indicator specified by <*string*>. This feature is useful because it results in an annotated script file, which can be handy for debugging.

To create a script file for the CA-Clipper/Exospace linker, which uses the "#" symbol as a comment indicator, you would include the following #comment command in your script file. Note that in this case, in which the comment indicator is the same symbol that is used to introduce template commands, the commands are written to the script file using a single leading # symbol. For example:

```
#comment "#"
```

**Note:** To write a blank line to the target script or batch file, regardless of the #comment setting, use ^n on a line by itself in the appropriate place in the template file rather than leaving the line completely blank.

#error <*string*>

Writes a message to the log file and sets the __Error diagnostic symbol.

#escape <*c*>

By default, the ^ character works as the escape symbol in template files allowing the use of reserved characters. For example, to use the " character in a quoted string, you would use ^" to process the double quote as a normal character. To change the escape symbol, use the #escape command, as in:

```
#escape ~
```

***Important!*** *The characters \ and # are not allowed as escape symbols.*

#gen <*symbol*>
...
#endgen

Delimits a generated script section, which permits a string list to be expanded into one or more script commands. The #gen...#endgen section is processed if the string list, represented by <*symbol*>, is not empty. In the body of the generated script section, $(<*symbol*>) is expanded into a series of elements from the members of <*symbol*>. A sufficient number of script lines are generated to exhaust the list.

In this example, a FILE script command is generated based on all object files necessary to build this application:

```
#gen allObjs
#sep   ", "
FILE $(allObjs)
#endgen
```

**Note:** In this example, #sep has special significance because it occurs within a generated script section. See the #sep command for more information.

#if <*symbol*>
...
(#else)
...
#endif

Defines one or more lines that will be processed if the specified <*symbol*> is "true." Proper nesting of #if constructs is permitted. You can use this construct within the batch section to control output to the batch file, or elsewhere to control output to the script file.

Whether or not a symbol is true depends on the type of symbol involved. For a string, #if tests true (.T.) if the string is not null. For a string list, it tests true (.T.) if the list is not empty. For a number, it tests true (.T.) if the value is not zero. For example:

```
#if __Map
    #if __MapFile
        MAP=$(__MapFile)
    #else
        MAP=$R(__OutFile) .MAP S, A
    #endif
#else
#message  You have not requested a map.
#endif
```

To test for a "false" condition, simply leave the section between the #if and #else empty. For example:

```
#batch
#if __Error
#else
    IF EXIST $(__OutFile) DEL $(__OutFile)
    EXOSPACE @$(__ScriptFile) >> $(__LogFile)
#endif
#endbatch
```

#max <*n*>

Specifies the maximum length allowed for a generated script command. The default is 80 characters. If insertion of a single list item plus a #more string exceeds the maximum length, the line is not truncated, but written to the script file.

In this example, the maximum line length is expanded to 120 characters:

```
#gen allObjs
#max 120
...
#endgen
```

#message <*string*>

Writes a message to the log file without setting a diagnostic symbol.

#more "<*string*>"

Permitted only in a generated script section (see #gen), this command specifies a string to be added to the end of a script line when data is to be continued on a new line.

In this example, when the first FILE statement exceeds the maximum line length, the #more command terminates the statement with a new line character and, thus, writes the string " FILE" at the beginning of the next line:

```
#gen allObjs
#max 40
#sep   ", "
#more  "^n   FILE "
#write "   FILE "
$(allObjs)
#endgen
```

**Note:** This example does not demonstrate the easiest way to generate a series of FILE statements. It is overly complex for the purpose of demonstrating the #more command. You could accomplish the same thing by replacing the #more, #write, and $(allObjs) lines with a single line, FILE $(allObjs).

For a maximum line length of 40 characters, the generated output will look something like this:

```
FILE File1, File2, File3, File4, File5
FILE File6, File7
```

In this example, the new line was forced using ^n, but this is not always necessary with the #more command. If the <*string*> does not explicitly include ^n, #more assumes a new line character at the end of the string. This is demonstrated in the example below for the #sep command.

#sep "<*string*>"

Certain symbols produce a delimited list of values in which the default delimiter, or *separator*, is a comma, followed by a single space. The #sep command, if issued outside of a generated script section, defines a new separator symbol to use when a delimited list is produced via $(<*symbol*>).

If issued within a generated script section (as defined by the #gen command), #sep indicates that multiple list elements separated by the specified <*string*> can occur on a single script line. If the addition of another list element would cause a script line (with a continuation string) to exceed the maximum line length, a new script line is started. See #max and #more for information on specifying the maximum line length and the continuation string, respectively.

This example specifies "-+" as the separator:

```
#gen allObjs
#max 40
#sep     " -+"
#more    " &"
-+$(allObjs)
#endgen
```

For a maximum line length of 40 characters, the generated output will look something like this:

```
-+File1 -+File2 -+File3 -+File4 -+File5 &
-+File6 -+File7
```

#template <c>

By default, the # symbol is used to identify template commands. When the template file is being processed, the system looks for this symbol as the first character in a line to determine its course of action. You can change this default using the #template command, as in:

```
#template %
```

After issuing the #template command, all subsequent commands in the template file must begin with the newly designated symbol instead of the # symbol. For example:

```
%batch
...
%endbatch
```

The #template command is useful if the syntax of your output script file requires the # symbol to be the first character of a script command.

#warning <*string*>

Writes a message to the log file and sets the __Warning diagnostic symbol.

#write "<*string*>"

Writes a string to the output script file without any interpretation. The string is not automatically terminated with a new line. The escape sequence, ^n, inside the string is interpreted as a new line character.

# Index

## A



## B

## C

## D

## E

## F

## G



## H



## I



## L



## M

## N



## O



## P



## R



## S

## T

## U



## V

## W