Visual Objects

For
Windows 2000® and Windows XP®

IDE User Guide
Version 2.7

VISUAL
OBJECTS
2.7
Visual Objects 2.7
Client Server Edition
The simpler C++, the more intelligent XBase
For Windows 2000® / XP®
ca
GRAFX
visit us at www.cavo.com
All Product names referenced herin are trademarks of their respective companies
©Computer Associates 1992-2002 Computer Associates International, Inc.One Computer Associates Plaza, Islandia NY, 11788-7000

Contents

Chapter 3: Using the Repository Explorer

Chapter 1

Introduction

This guide explains how to use the various features of the CA-Visual Objects 2.7 integrated development environment (IDE). It is organized into the following chapters:

Chapter 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.

Chapter 2: Working in the Desktop**,** presents the IDE and its various components, explains how to customize and save the current desktop, and describes how to set default system options.

Chapter 3: Using the Repository Explorer**,** explains how to use the Repository Explorer to create, view, and modify your projects, applications, modules, and entities, as well as view their class hierarchy.

Chapter 4: Using the Window Editor**,** explains how to create various types of application windows and define their GUI and common controls.

Chapter 5: Using the Menu Editor**,** describes how to create custom menus and toolbars, as well as standard, predefined ones, and how to add them to your applications.

Chapter 6: Using the Source Code Editor**,** demonstrates how to enter and edit source code in CA-Visual Objects.

Chapter 7: Defining Data Servers and Field Specifications**,** describes how to create and maintain data servers and all ancillary information like index files, field lists, and so on for the data server objects in your applications, using the following editors: DB Server, SQL, and FieldSpec.

Chapter 8: Creating Data-Aware Windows**,** explains how to create data, data dialog, and sub-data windows that utilize your data servers and field specs.

Chapter 9: Using the Report Editor**,** provides instructions for creating sophisticated reports using CA-Visual Objects Report Editor.

Chapter 10: Using the Image Editor**,** describes how to create and modify images, such as icons, cursors, and ribbons, for your applications.

Chapter 11: Debugging Your Applications**,** demonstrates how to set various debugging options, and to test and debug your applications at any level using advanced debugging tools.

Chapter 12: Importing and Exporting Applications**,** explains how to import and export common types of files, including text-based source files.

Appendix A: File Types**,** lists the different types of files generated or used by CA-Visual Objects.

Appendix B: CA-Visual Objects Registry Entries**,** explains key entries in the system registry.

Appendix C: Using the Install Maker**,** describes how to use the CA-Visual Objects Install Maker to generate installation disks for a selected application.

Appendix D: Using the CA-Uninstall Utility**,** explains how to use the CA-Uninstall utility to remove CA-Visual Objects from your PC.

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 CA-Visual Objects.

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 and discussion of them.

Key Names

The names of keys, such as Enter, Ctrl, and Del, appear in the document as they do on your keyboard, where possible.

Note that when referring to the four arrow keys as a group, they are referred to as Direction keys; however, the name of each Direction key (for example, Up arrow or Left arrow) is used when referring to them individually.

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, method, and procedure names (like SetDoubleClickTime() and Abs())
- Class names (for example, TopAppWindow and DBServer)

- Variable names (for example, oTopAppWindow and nLoopCounter)

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 *IDE User Guide*.

- Part name in single quotes:

  See ‘Database Programming’ in the *Programmer’s Guide*.

- Chapter name in double quotes:

  See “Using the Source Code Editor” in the *IDE User Guide*.

- Section name as it appears in the document:

  Also see the Setting the Search Path section.

Getting Help

CA-Visual Objects 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 CA-Visual Objects language and IDE.
Context Help	Allows you to get context-sensitive help for an item or area currently displayed on your screen.
How to Use Help	Describes how to use the Windows online Help system.

In the IDE 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.

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 Desktop

This chapter introduces you to the CA-Visual Objects IDE and its various components, explains how to customize and save the current desktop, and describes how to set default system options.

Note: For an overview of all the innovative changes to the IDE in CA-Visual Objects, version 2.7, see either *Getting Started Guide*. Also, for assistance with specific topics, refer to *CA-Visual Objects 2.7 Help*—the online help system.

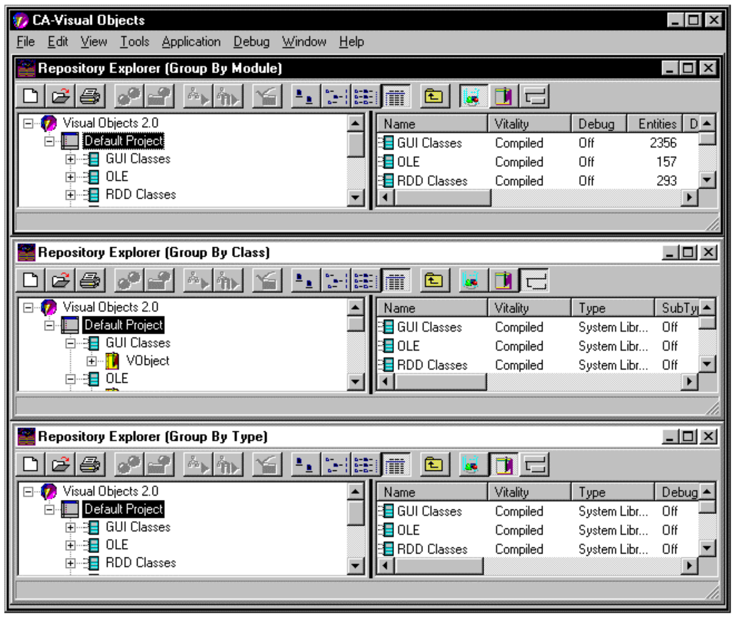
Desktop Basics

The IDE is a flexible, intuitive, and powerful environment for creating applications, libraries, and dynamic link libraries (DLLs). Almost all features of the IDE—the Repository Explorer, the visual editors, the Source Code Editor, the background compiler, the Debugger, and the Error Browser—are available at the touch of a button from almost any window. For example, you can:

- Open and work with multiple applications
- Double-click on any entity—function, class, menu, form, report, data server, field spec, etc.—to launch the editor associated with that entity
- Modify a single entity and click the Build button to rebuild the necessary parts of the application based on that change
- Start up two editor sessions and copy and paste information from one to the other
- Execute a compiled application or generate an executable file

Arranging and Manipulating Windows

The CA-Visual Objects desktop permits you to open and simultaneously work with multiple windows and editors. You can even have multiple copies of the Repository Explorer open, as shown below!



Name	Vitality	Debug	Entities	D
GUI Classes	Compiled	Off	2356	
OLE	Compiled	Off	157	
RDD Classes	Compiled	Off	293	

Name	Vitality	Type	SubTy
GUI Classes	Compiled	System Libr...	Off
OLE	Compiled	System Libr...	Off
RDD Classes	Compiled	System Libr...	Off

Name	Vitality	Type	Debug
GUI Classes	Compiled	System Libr...	Off
OLE	Compiled	System Libr...	Off
RDD Classes	Compiled	System Libr...	Off

(Three different views of the Default Project in CA-Visual Objects are shown here, with the project's subitems grouped by module, class, and type, respectively, in each of the right panes. See Browsing Projects, Applications, and Modules for more information about viewing items in the Repository Explorer's tree structure.)

You can switch between open windows by clicking them with the mouse or 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 standard Windows techniques.

The Toolbars

Almost every window and editor contains a customized toolbar that provides buttons as shortcuts for 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 Repository Explorer contains buttons for creating a new application, importing an existing application, and setting default application options. Other toolbar buttons allow you to group items in the Repository Explorer tree by module, type, or class. The buttons in the Menu Editor toolbar, on the other hand, allow you to 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 tooltip window with descriptive text pops up right next to it.

The Status Bars

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

For example, move the mouse over the entities displayed in the Repository Explorer to view the first line of each entity in the status bar; or highlight a menu command to display a description of what it does in the status bar.

Saving, Building, and Executing

Note: CA-Visual Objects now provides version control for managing applications with its new Source Code Control Interface. See the online help for detailed information about this new feature.

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 and then save any additional edits on a frequent basis.

To initially save a new entity in the repository, click the Save toolbar button in any editor.

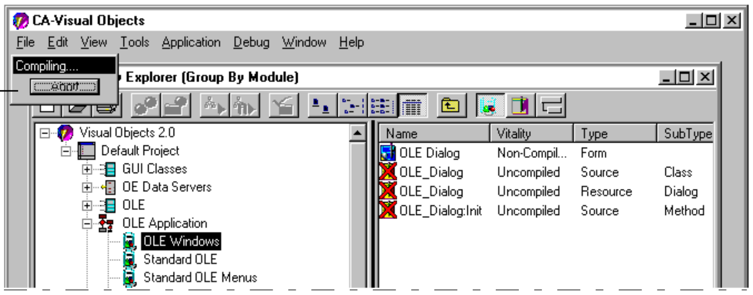
To save any subsequent changes, just click the Save toolbar button again.

Building an Application

Note: See the online help for detailed information about the new Rebuild All menu command.

You can build an application at any time by clicking the Build button on the toolbar. A compilation message box will appear, allowing you to abort the compilation process if you want. For example:

Compilationmessagebox



Name	Vitality	Type	SubType
OLE Dialog	Non-Compil...	Form	
OLE_Dialog	Uncompiled	Source	Class
OLE_Dialog	Uncompiled	Resource	Dialog
OLE_Dialog:Init	Uncompiled	Source	Method

Note: Because CA-Visual Objects 2.7 has a background compiler and linker, you can continue working in another module or entity while building the application. However, you *cannot* save your current work until the build is complete, as indicated by the disappearance of the modeless compilation message box and the “Build done.” status bar message.

Based on date and time stamps and the system-maintained dependency list, CA-Visual Objects 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, rebinding, and relinking.

Tip: You can update the date / time stamp for each entity and force the compiler to rebuild all of the entities in an application, thereby overriding the control of the system-maintained dependency list. To do so, use either the Touch All Entities command on the Edit menu or the Touch command on the local pop-up menu when you right-click on an entity. See the online help for details about Touch All Entities.

The manner in which an individual application is built can be controlled using application-specific compiler options or project-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.

You can also execute an application by choosing the Command Line command from the Tools menu. See the Command Line section later in this chapter for details.

The third option is to create an executable file that can be run independently of the IDE. This is discussed in detail next in Generating an EXE.

Generating an EXE

CA-Visual Objects allows you to generate quickly and easily a stand-alone executable file (.EXE) for an application. This .EXE can then be run independently of the IDE. Simply choose the Make EXE toolbar button or the Application Make EXE menu command when the desired application is selected.

When creating an .EXE file, CA-Visual Objects also creates a folder and icon for the Windows Start menu—you can use system-supplied defaults for these or customize them for each application. (See **Creating a New Application** in “Using the Repository Explorer” for details.)

If the selected application is a DLL, this toolbar button corresponds to the Application Make DLL command, generating a .DLL and an .AEF file with the same name. The .AEF file defines the public protocol, or interface, for the DLL.

The IDE Tools

CA-Visual Objects 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 windows, menus, source code, data servers, reports, and icons. Other tools include a compiler, debugger, UDC tester, and command-line utility.

In the IDE, all development tools are closely integrated with the repository. For example, double-clicking on an entity in a browser invokes the appropriate editor for that entity: the Window Editor or Menu Editor for form or binary menu entities, respectively; the Report Editor for report entities; and the Source Code Editor for code (functions, classes, methods, and so on).

Repository Explorer

The Repository Explorer allows you to view and manipulate the code that is currently stored in your repository in a convenient and organized way. In CA-Visual Objects, you can browse:

- Projects
- Applications, libraries, and DLLs
- Modules
- Entities
- Classes
- Errors

Tip: You can customize the Repository Explorer by specifying what particular items and subsets of information are displayed in its collapsible/expandable tree structure. See **Customizing the Repository Explorer** in the "Using the Repository Explorer" chapter for detailed information.

Browsing Projects, Applications, and Modules

Projects represent the highest level in the CA-Visual Objects hierarchy: *projects* consist of *applications*, *applications* consist of *modules*, which in turn consist of *entities*. The Repository Explorer follows this top-down hierarchy.

Group By Module

When you start CA-Visual Objects, the *Repository Explorer* is automatically loaded, displaying all of the various projects, applications, libraries, DLLs, and modules that currently exist.

Note: This default view is referred to as *module view* and is indicated by the depressed Group By Module toolbar button.

Double-clicking on a project displays the applications, libraries, or DLLs defined for it, while double-clicking on an application displays the modules defined for it. Similarly, double-clicking on a module displays the entities defined for that module.

Browsing Entities

Group By Type

You can browse all of the entities in the *current* application by clicking on the Group By Type toolbar button. This view of the Repository Explorer displays entities in a similar collapsible / expandable tree structure and allows you to set a name filter.

Browsing Classes

Group By Class

At any time you can also view a comprehensive list of the methods and properties associated with an application's classes by clicking on the Group By Class toolbar button.

Tip: To return to the Repository Explorer's default view, simply click the Group By Module toolbar button.

Browsing Errors

At any time during the development cycle, you can access the Error Browser to view a comprehensive list of all of the *compilation* errors and warnings within an application.

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

See **Chapter 11: Debugging Your Applications** for details about the Error Browser.

Editors

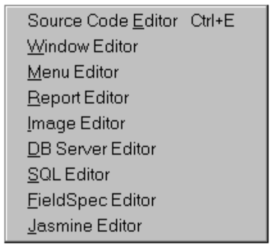
The various editors allow you to create windows, menus, source code, data servers, reports, and images easily, conveniently, and efficiently.

Accessing the Editors

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

Creating an Entity

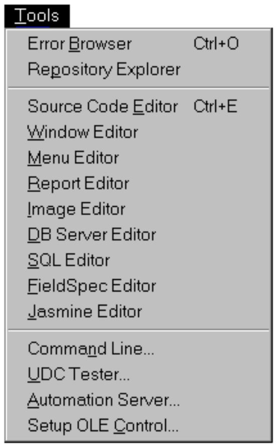
The most logical place to create a new entity is from within a module—simply click on the New Entity toolbar button. Then select the appropriate editor for the type of entity you want to create from the local pop-up menu that appears:



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 visual editor's toolbar.

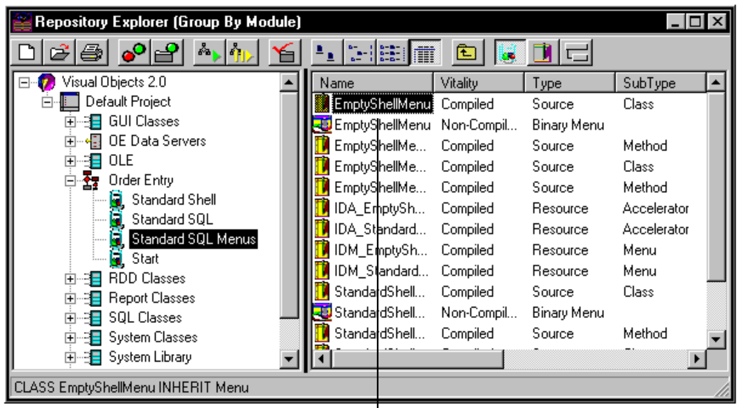
You also can create a new entity by accessing an editor from the Tools menu:



Tip: All new entities are created as part of the *current* module. If you want the entity to be part of a new module, you must create the module before creating the entity.

Opening an Entity

Opening an entity simply means to display the entity in its associated editor. To open an existing entity, simply double-click on it, or highlight it and press Enter:

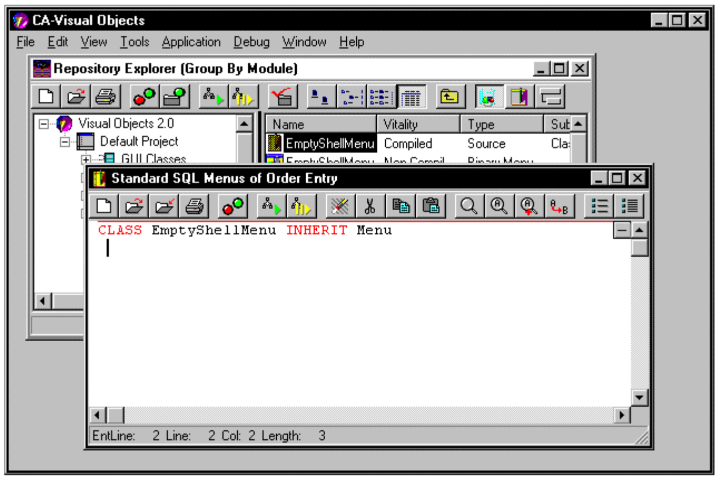


Name	Vitality	Type	SubType
EmptyShellMenu	Compiled	Source	Class
EmptyShellMenu	Non-Compil...	Binary Menu	
EmptyShellMe...	Compiled	Source	Method
EmptyShellMe...	Compiled	Source	Class
EmptyShellMe...	Compiled	Source	Method
IDA_EmptySh...	Compiled	Resource	Accelerator
IDA_Standard...	Compiled	Resource	Accelerator
IDM_EmptySh...	Compiled	Resource	Menu
IDM_Standard...	Compiled	Resource	Menu
StandardShell...	Compiled	Source	Class
StandardShell...	Non-Compil...	Binary Menu	
StandardShell...	Compiled	Source	Method

CLASS EmptyShellMenu INHERIT Menu

Double-clickhere...

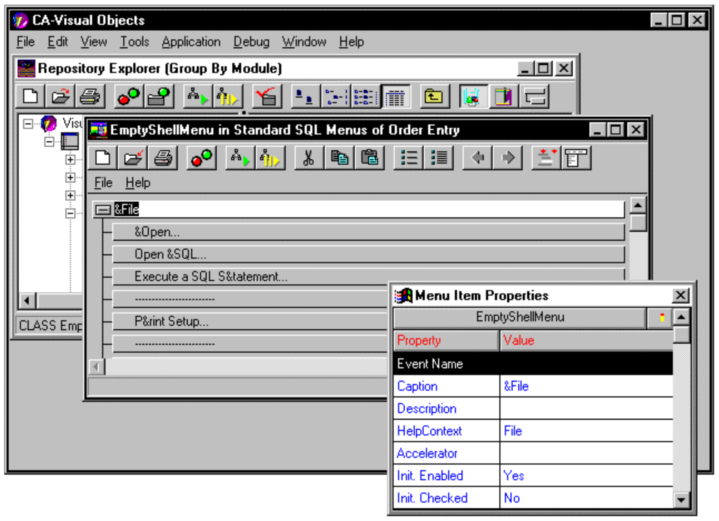
For example, double-clicking the EmptyShellMenu source entity invokes the Source Code Editor:


CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Visual Objects 2.0
Default Project
GUI Classes
Name Vitality Type Sub
EmptyShellMenu Compiled Source Cla
Standard SQL Menus of Order Entry

```
CLASS EmptyShellMenu INHERIT Menu
```

EntLine: 2 Line: 2 Col: 2 Length: 3

Double-clicking the EmptyShellMenu binary menu entity, on the other hand, invokes the Menu Editor:


CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
EmptyShellMenu in Standard SQL Menus of Order Entry
File Help
&File
&Open...
Open &SQL...
Execute a SQL S&tatement...
P&rint Setup...
CLASS Emp
Menu Item Properties
EmptyShellMenu

Property	Value
Event Name	
Caption	&File
Description	
HelpContext	File
Accelerator	
Init. Enabled	Yes
Init. Checked	No

You can work with multiple entities within the same module at the same time. For example, if you had two binary menu entities, two form entities, and two source entities open simultaneously, the IDE desktop would have two separate Menu Editors and two separate Window Editors open but only a single Source Code Editor, as both source code entities would appear in the *same* Source Code Editor window.

See **Creating Entities** in “Using the Repository Explorer” for more detailed information about entity types and creating new entities.

Creating and Editing Source Code

You can create source code entities at any time in CA-Visual Objects in several different ways. First, you can access the Source Code Editor directly and manually type in code.

Secondly and most importantly, when you define entities using the visual editors, CA-Visual Objects generates straightforward source code utilizing the supporting class libraries. For example, creating a report in CA-Report Writer, part of CA-Visual Objects Report Editor, will generate a subclass of the ReportQueue class. This code can then be edited from within the Source Code Editor.

Lastly, you can import text-based source files, while within the Source Code Editor.

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 **Chapter 6: Using the Source** Code Editor for details about editing your applications' source code.

The Visual Editors

Many of the editors are *visual*, that is, you can lay out various GUI controls—like push buttons, check boxes, and scroll bars—on a window, design a report, and create a custom menu using point-and-click, drag-and-drop techniques. Visual feedback is immediate when designing objects in the Menu Editor, Window Editor, and Report Editor.

See the following chapters for more information about these editors: **Chapter 5: Using the Menu** Editor, **Chapter 4: Using the Window** Editor, **Chapter 8: Creating Data-Aware Windows**, and **Chapter 9: Using the Report** Editor.

The Data Server Editors

CA-Visual Objects provides a set of data server editors: the DB Server and SQL Editors.

DB Server and SQL Editors

The DB Server Editor allows you to create *data servers* based on the traditional Xbase model of a .DBF file, whereas data servers created with the SQL Editor are based on the SQL paradigm. Data servers are high-level objects used to provide an object-oriented interface for a database whose structure is known at compile-time. Information about the database, such as its file name and sort order, is stored in the data server along with detailed field information stored in the form of FieldSpec objects. You can create automatic layouts for data servers in the Window Editor that you can easily modify.

Note: With both editors, you can import an existing database structure and generate a default set of field specifications that you can optionally modify. In addition, the DB Server Editor allows you to design a data server "from scratch" and generate a database file (and index files) from the data server definition.

The FieldSpec Editor

Although both data server editors have 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 creating a Salary field in a new table.

See **Chapter 7: Defining Data Servers** and Field Specifications for more information about these editors.

Creating and Editing Images

Using the Image Editor you can create custom icons, cursors, bitmaps, and ribbons for your applications using a drag-and-drop interface that allows you to work with several images at the same time.

See **Chapter 10: Using the Image** Editor for more information.

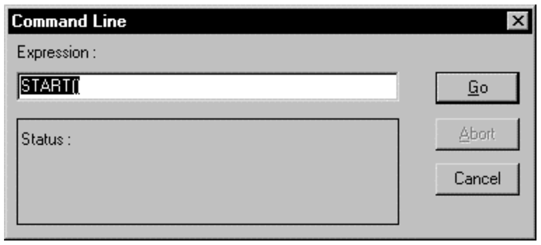
Command Line

CA-Visual Objects provides a command line utility that allows you to evaluate any valid expression at any time. Examples of valid expressions are 2 + 2, Start(), and QOut("Hi"), whereas the following is a statement, not an expression: ? "Hi" (For more detailed information about valid expressions, refer to the "Operators and Expressions" chapter of the *Programmer's Guide.*)

To access the command line utility, select the Command Line command from the Tools menu.

Note: You can only use the command line utility after the application, library, or DLL has been successfully compiled.

The Command Line dialog box appears—by default, it contains "Start()" in the Expression edit control. Choosing Go at this point will execute the application:



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.*

Refer to the **Chapter 11: Debugging Your Applications** chapter for more information about evaluating expressions.

UDC Tester

CA-Visual Objects provides a UDC tester that you can use to create and test user-defined commands (or *UDCs*). In general, a UDC provides a way to specify an English-language statement that is, in fact, one or more expressions, thereby improving the readability of source code.

See the online help system for details about the syntax rules and prerequisites for creating UDCs.

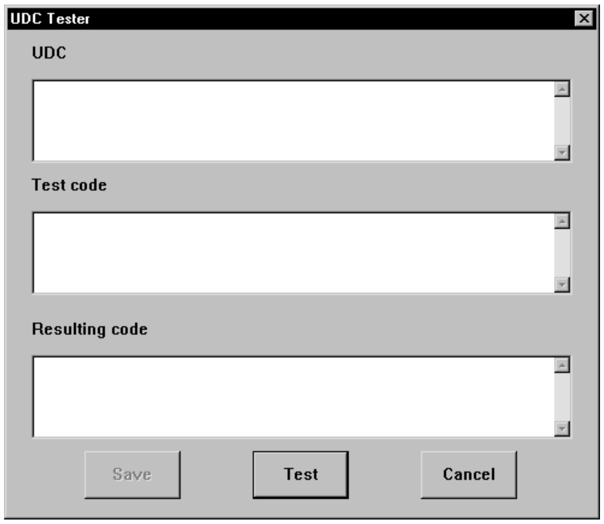
Note: All commands in CA-Visual Objects are UDCs and are supplied in the STD.UDC file. You can, however, create your own .UDC files and associate them with your applications, using the Properties dialog box. (See **Creating a New Application** in “Browsing Applications, Modules, Entities, and Classes” for details.)

Testing a UDC

To test a UDC:

1. Choose the UDC Tester command from the Tools menu.

   The UDC Tester dialog box appears:

2. In the UDC edit control, enter the UDC to be tested—for example, **SKIP <x> => DBSkip(<x>)**.

3. Enter some sample source code to be tested in the Test Code edit control—for example, **SKIP 1**.

4. Choose the Test push button.

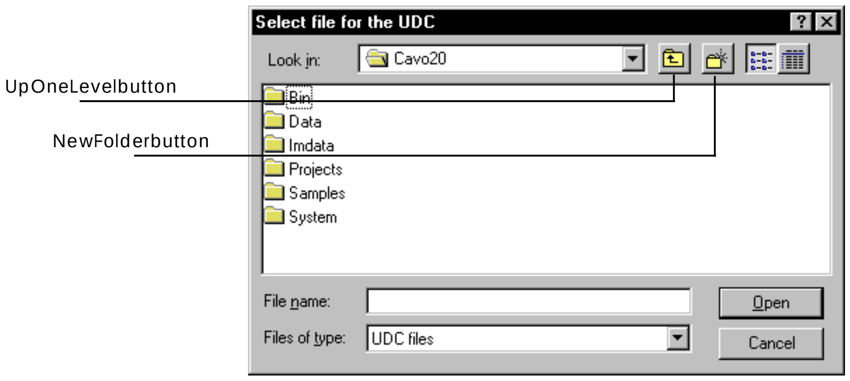
The Resulting Code edit control displays the result of the test. For example, it would display the following for the statements used as examples above:

```
DBSkip(1)
```

Tip: If you are testing a UDC that already exists, you may find it helpful to use the Clipboard's cut / paste feature to enter information into the UDC combo box. If you are unfamiliar with this feature, refer to your *Microsoft Windows* or *Windows NT User Guide.*

Creating and Saving a UDC

You can also save to a .UDC file using the UDC Tester dialog box. After testing the UDC, click the Save button to open the Select File for the UDC dialog box:



Double-click on a folder and select one of its .UDC files, or enter the .UDC file name in the File Name edit control, and the UDC will be written to the file.

Tip: Use the Up One Level toolbar button to move up a level within a directory, and use the New Folder button to create a new directory to store the UDC.

Once you have identified your .UDC file, subsequent saves append new UDCs to the same file without prompting until you choose Cancel to close the UDC Tester dialog box.

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 CA-Visual Objects Debugger, select the Debug toolbar button from within the Repository Explorer or the Trace Expression toolbar button in any of the editors. Naturally, you can also access the Debugger using the Debug Run menu command.

Note: For detailed information about the Debugger's , such as
AutoStart debugging and DLL debugging,
see the online help.

The Debugger allows you to:

- Set debugging options at the application, module, and entity levels
- 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, index, and other work area information in a separate window and modify database field values
- View and modify system settings

To utilize debugging in your application, the Enable Debug option must be checked in the Properties dialog box. You can also control debugging at the module and entity level by highlighting the module or entity and then pressing the right mouse button. This opens a local pop-up menu with debugging options that you can select.

See **Chapter 11: Debugging Your Applications** for complete information about the Debugger.

Reindexing the Repository

If you receive a message about index corruption while working in the IDE, you should completely rebuild and synchronize your index files by reindexing the repository. To do so, highlight the project and then select the Reindex Project command from the Repository Explorer's File menu.

Automation Server

Note: CA-Visual Objects 2.7 not only provides OLE client support for OLE automation servers, OLE objects, and ActiveX controls, but it now supports the creation of both OLE automation servers and ActiveX servers. For detailed information, refer to the online help.

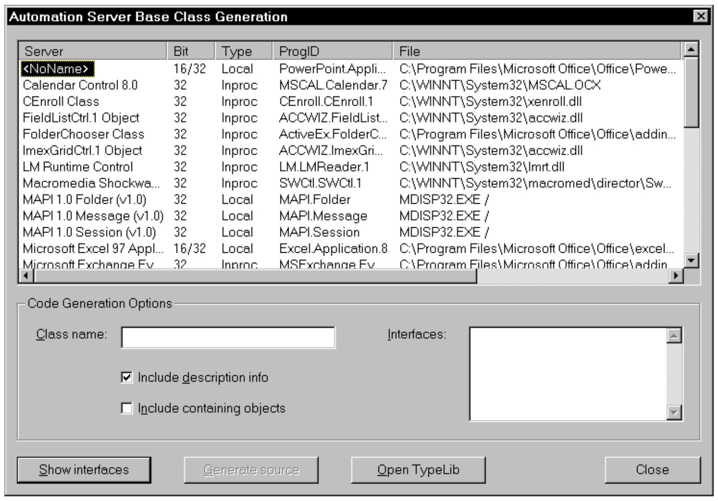
The Automation Server allows you to create CA-Visual Objects classes for object linking and embedding (OLE) automation servers provided by third-party applications.

Note: In order to access the Automation Server Base Class Generation window, the OLE library has to be included in your application's search path. See **Setting the Search Path** in the "Using the Repository Explorer" chapter.

To access the Automation Server:

1. Select the Automation Server command from the Tools menu.

   The Automation Server Base Class Generation window appears:



Server	Bit	Type	ProgID	File
<NoName>	16/32	Local	PowerPoint.Appli...	C:\Program Files\Microsoft Office\Office\Powe...
Calendar Control 8.0	32	Inproc	MSCAL.Calendar.7	C:\WINNT\System32\MSCAL.OCX
CEnroll Class	32	Inproc	CEnroll.CEnroll.1	C:\WINNT\System32\xenroll.dll
FieldListCtrl.1 Object	32	Inproc	ACCWIZ.FieldList...	C:\WINNT\System32\accwiz.dll
FolderChooser Class	32	Inproc	ActiveEx.FolderC...	C:\Program Files\Microsoft Office\Office\addin...
ImexGridCtrl.1 Object	32	Inproc	ACCWIZ.ImexGri...	C:\WINNT\System32\accwiz.dll
LM Runtime Control	32	Inproc	LM.LMReader.1	C:\WINNT\System32\lmrt.dll
Macromedia Shockwa...	32	Inproc	SWCtl.SWCtl.1	C:\WINNT\System32\macromed\director\Sw...
MAPI 1.0 Folder (v1.0)	32	Local	MAPI.Folder	MDISP32.EXE /
MAPI 1.0 Message (v1.0)	32	Local	MAPI.Message	MDISP32.EXE /
MAPI 1.0 Session (v1.0)	32	Local	MAPI.Session	MDISP32.EXE /
Microsoft Excel 97 Appl...	16/32	Local	Excel.Application.8	C:\Program Files\Microsoft Office\Office\excel...
Microsoft Exchange Ev	32	Inproc	MSExchange.Ev	C:\Program Files\Microsoft Office\Office\addin

[✓] Include description info

[ ] Include containing objects

   Note: In this version of CA-Visual Objects, this window has a new option, Include Containing Objects. For detailed information, see the online help.

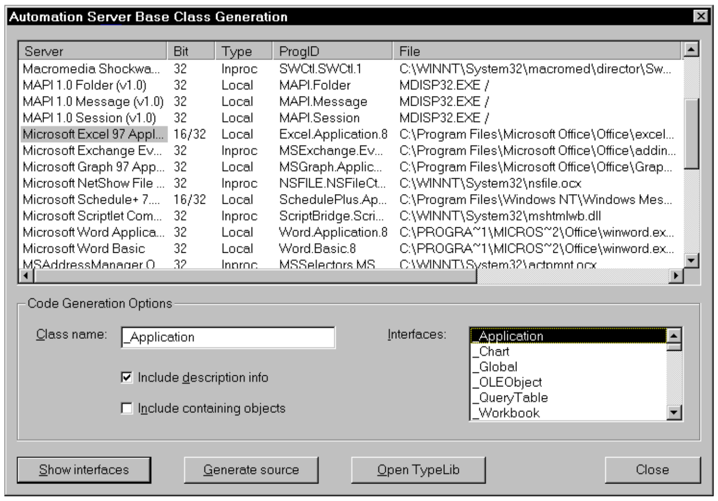
2. Select one or more automation servers defined to your PC from the list box (for example, Excel 97).

3. Click the Show Interfaces button.

   The Interfaces list box is filled with available interfaces.

   Note: Some servers may actually become visible when clicking on the Show Interfaces button.

4. Select an interface from the Interfaces list box:



Automation Server Base Class Generation

Server	Bit	Type	ProgID	File
Macromedia Shockwa...	32	Inproc	SWCtl.SWCtl.1	C:\WINNT\System32\macromed\director\Sw...
MAPI 1.0 Folder (v1.0)	32	Local	MAPI.Folder	MDISP32.EXE /
MAPI 1.0 Message (v1.0)	32	Local	MAPI.Message	MDISP32.EXE /
MAPI 1.0 Session (v1.0)	32	Local	MAPI.Session	MDISP32.EXE /
Microsoft Excel 97 Appl...	16/32	Local	Excel.Application.8	C:\Program Files\Microsoft Office\Office\excel...
Microsoft Exchange Ev...	32	Inproc	MSExchange.Ev...	C:\Program Files\Microsoft Office\Office\addin...
Microsoft Graph 97 App...	32	Local	MSGraph.Applic...	C:\Program Files\Microsoft Office\Office\Grap...
Microsoft NetShow File ...	32	Inproc	NSFILE.NSFileCt...	C:\WINNT\System32\nsfile.ocx
Microsoft Schedule+ 7....	16/32	Local	SchedulePlus.Ap...	C:\Program Files\Windows NT\Windows Mes...
Microsoft Scriptlet Com...	32	Inproc	ScriptBridge.Scri...	C:\WINNT\System32\mshtmlwb.dll
Microsoft Word Applica...	32	Local	Word.Application.8	C:\PROGRA~1\MICROS~2\Office\winword.ex...
Microsoft Word Basic	32	Local	Word.Basic.8	C:\PROGRA~1\MICROS~2\Office\winword.ex...
MSAddressManager.O...	32	Inproc	MSSelectors.MS...	C:\WINNT\System32\actpmnt.ocx

Code Generation Options

Class name: _Application

Interfaces: _Application, _Chart, _Global, _OLEObject, _QueryTable, _Workbook

☑ Include description info

☐ Include containing objects

Alternatively, click the Open TypeLib button to access a type library file (.TLB or .OLB) for code generation.

Either the interface name or the type library file name will appear appropriately in the Class Name edit control as the default class name.

5. Click Generate Source.

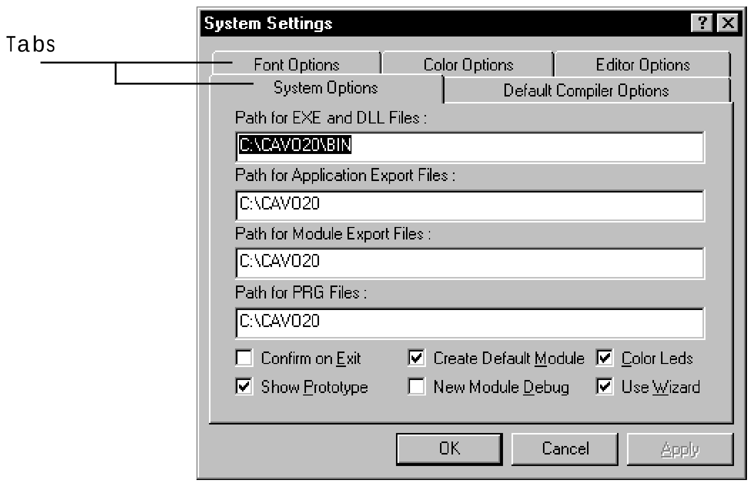
Below are more complete descriptions of the available options for this window:

(Automation Servers)	Lists the available third-party applications acting as (or accessible as) an OLE automation server (for example, Microsoft Excel 97, PowerPoint, Exchange, etc.). Additional information includes: 16/32-bit characterization, type, program ID, file name, and class ID (CLSID).
Class Name	The name of the CA-Visual Objects-generated class. By default, CA-Visual Objects uses the interface name as the class name.
Include Description Info	If selected, the automation server is queried for information about each entity's server type. These descriptions will appear in the Repository Explorer's list view pane, as well as in the generated source code.
Interfaces	Lists the specific interfaces through which an automation server is available. Each server has at least one main dispatch interface.
Open Type Lib	If selected, allows you to choose an external type library file (*.TLB, *.OLB) from the Open Type Library dialog box for code generation.

Setting System-Wide Options

Note: See the online help for new system, application, and Source Code Editor options in CA-Visual Objects 2.7, as well as detailed information about the Explorer tab page.

You can set a number of system-wide options for CA-Visual Objects, including compiler options and settings for the desktop, by selecting the File Setup command. The System Settings dialog box appears:



Path for EXE and DLL Files : C:\CAVO20\BIN

Path for Application Export Files : C:\CAVO20

Path for Module Export Files : C:\CAVO20

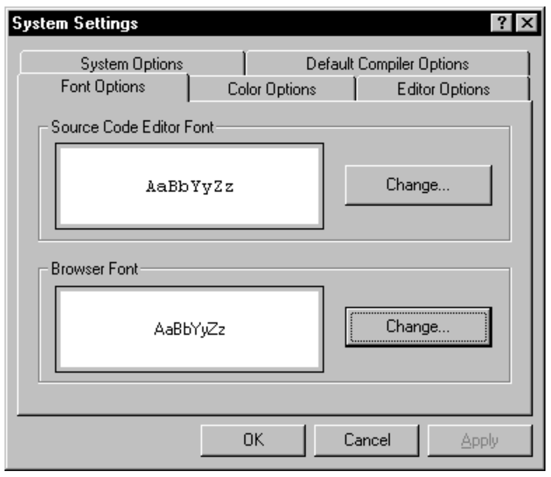
Path for PRG Files : C:\CAVO20

- [ ] Confirm on Exit
- [x] Create Default Module
- [x] Color Leds
- [x] Show Prototype
- [ ] New Module Debug
- [x] Use Wizard

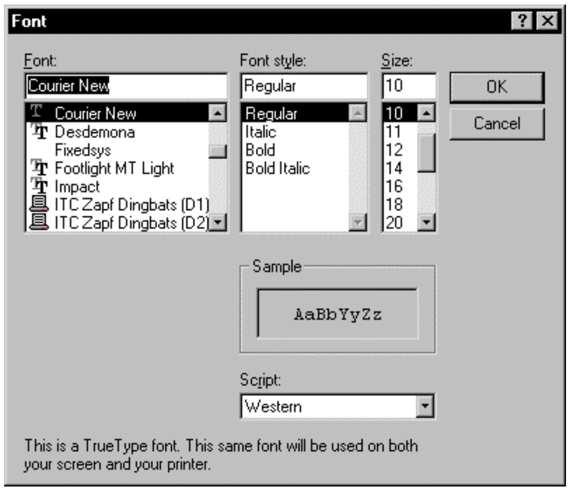
Note: This dialog box now contains the following tabs: System, Compiler Defaults, Fonts, Colors, Editor, and Explorer.

Selecting Fonts

CA-Visual Objects allows you to customize fonts on two levels: for the various browsers and for the Source Code Editor, as indicated by the Font Options tab:



Choosing the Change button for either Source Code Editor Font or Browser Font displays a standard Font dialog box:



Font: Courier New
Font style: Regular
Size: 10
Fonts: Courier New, Desdemona, Fixedsys, Footlight MT Light, Impact, ITC Zapf Dingbats (D1), ITC Zapf Dingbats (D2)
Styles: Regular, Italic, Bold, Bold Italic
Sizes: 10, 11, 12, 14, 16, 18, 20
Script: Western

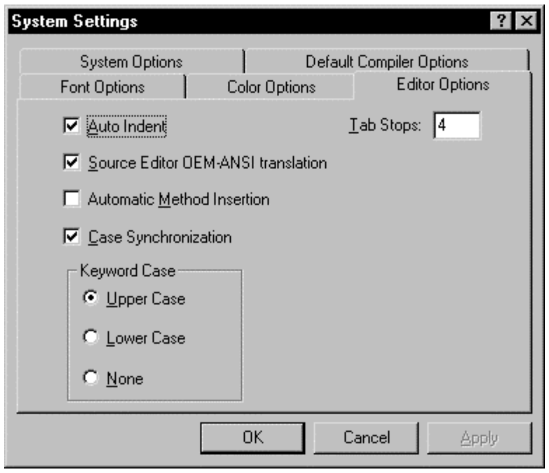
This is a TrueType font. This same font will be used on both your screen and your printer.

Browser Font The font you select for the Browser Font option is used for the text in the Repository Explorer, the standard text in the Error Browser, and the status bar text in all windows.

Source Code Editor Font The Source Code Editor Font option allows you to specify the font to be used for the text displayed in Source Code Editor windows.

Setting Editor Options

To set editing options—such as case synchronization and tab stops—for the Source Code Editor, click on the Editor Options tab in the System Settings dialog box:



[✓] Auto Indent
Tab Stops: 4
[✓] Source Editor OEM-ANSI translation
[ ] Automatic Method Insertion
[✓] Case Synchronization

Keyword Case: [● Upper Case] Lower Case None

Note: In this version of CA-Visual Objects, the Editor tab page provides three new options: Use Spaces for Tabs, Show Parameter Tips, and Activate IDE on Break. Additionally, the Automatic Method Insertion option has a new default value. For more detailed information, see the online help.

This dialog box provides the following options:

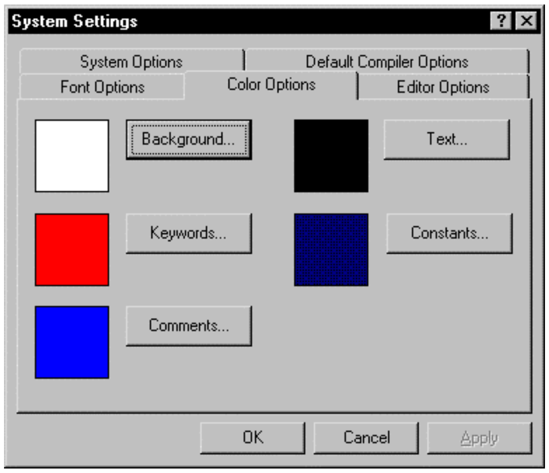
Auto Indent	If True, automatically indents text in the Source Code Editor according to the structure of your code.
Tab Stops	Defines the width of a tab (in characters).
Automatic Method Insertion	If True, enables CA-Visual Objects automatic method insertion feature, which brings up a local pop-up menu from which you can choose one of the specified object's methods. (See Chapter 6: Using the Source Code Editor for detailed information about this feature.)
	Note: In this version of CA-Visual Objects, the time-saving Automatic Method Insertion feature is enabled by default.
Case Synchronization	If selected, all references to other entities and keywords in the Source Code Editor are automatically case-synchronized according to their definitions in the repository. For example, if you type **arraynew()** in order to define a new array, CA-Visual Objects automatically changes the entry to its correct format, ArrayNew().
Keyword Case	The Keyword Case radio button group allows you to select a case setting for your keywords in the Source Code Editor. Valid choices are Upper Case, Lower Case, and None. The default is Upper Case.

Selecting Source Code Editor Colors

CA-Visual Objects 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, recognizing syntactic elements and color-coding them according to their category. (It also appropriately color-codes pasted or imported text.)

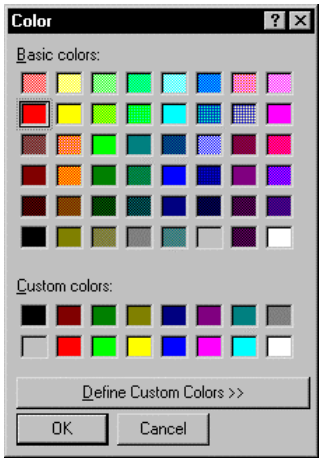
To customize these colors, click on the Color Options tab in the System Settings dialog box:



Then select one of the following push buttons:

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, classes, and so on
Constants	The color for all constants

In each case, a standard Color dialog box appears:

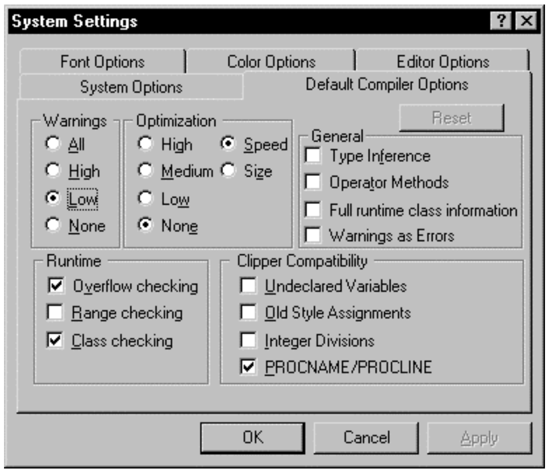


Setting Compiler Options

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

Default Compiler Settings

To set project-wide, default compiler options, click on the Default Compiler Options tab to access them:



System Settings

Font Options | Color Options | Editor Options | System Options | Default Compiler Options

Warnings: ( ) All ( ) High [● Low] ( ) None

Optimization: ( ) High ( ) Medium ( ) Low [● None]; [● Speed] ( ) Size

General:
- [ ] Type Inference
- [ ] Operator Methods
- [ ] Full runtime class information
- [ ] Warnings as Errors

Runtime:
- [✓] Overflow checking
- [ ] Range checking
- [✓] Class checking

Clipper Compatibility:
- [ ] Undeclared Variables
- [ ] Old Style Assignments
- [ ] Integer Divisions
- [✓] PROCNAME/PROCLINE

Typically, you will want to specify the most commonly used settings as permanent system defaults and then, if necessary, override some of the project-wide settings on an application-by-application basis. (Descriptions of the various options follow in the Application-Specific Compiler Options section.)

Note: Default compiler settings are in effect for *all* applications within a project, unless you override them by using the application-level compiler options. You can also override the default compiler settings at the module and entity levels. Refer to **Setting Module Properties** and **Setting Entity Properties** in "Using the Repository Explorer" for more information.

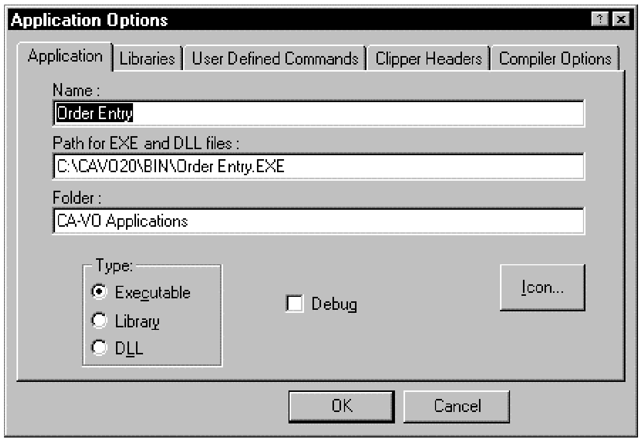
Application-Specific Compiler Options

To set compiler options for a specific application, overriding one or more system defaults:

1. Click the Application Properties toolbar button.

   Alternatively, right-click on the application and then select the Properties command from the local pop-up menu that appears, or select the Properties command from the Application menu.

The Application Options dialog box appears:


Application Options
Application | Libraries | User Defined Commands | Clipper Headers | Compiler Options
Icon...
OK
Cancel

Name : Order Entry

Path for EXE and DLL files : C:\CAVO20\BIN\Order Entry.EXE

Folder : CA-VO Applications

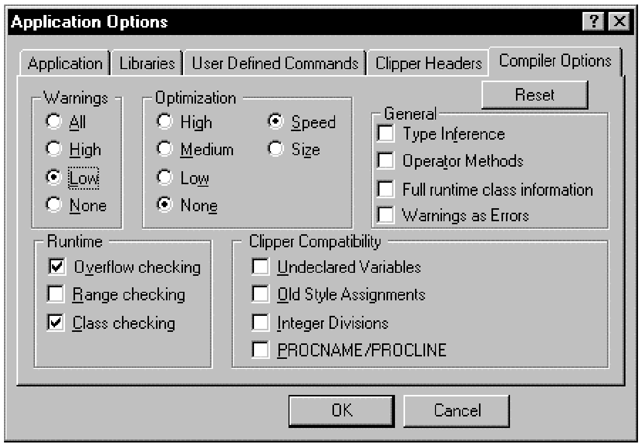
Type: [✓ Executable] Library DLL

Debug: ☐

Note: The Application Options dialog box has been updated in CA-Visual Objects 2.7 and now contains the following tabs: Application, Libraries, UDCs, Clipper Headers, Compiler, and OLE Server. Moreover, its tab pages provide many new options and updated default settings. For detailed information, see the online help.

Note: In this version of CA_Visual Objects, the appropriate Properties dialog box—Project Properties, Application Options, Properties (Module), or Properties (Entity)—can be accessed directly by using the Alt+Return key combination. This is an alternative method to right-clicking with the mouse and then selecting the Properties menu command from the local pop-up menu that appears.

2. Click on the Compiler Options tab to access the application-specific compiler options:


Application Options
Application | Libraries | User Defined Commands | Clipper Headers | Compiler Options
Reset
OK
Cancel

Warnings: All High [✓ Low] None

Optimization: High Medium Low [✓ None]; [✓ Speed] Size

General: ☐ Type Inference; ☐ Operator Methods; ☐ Full runtime class information; ☐ Warnings as Errors

Runtime: ☑ Overflow checking; ☐ Range checking; ☑ Class checking

Clipper Compatibility: ☐ Undeclared Variables; ☐ Old Style Assignments; ☐ Integer Divisions; ☐ PROCNAME/PROCLINE

3. Reset any of the compiler options described below.

4. Click OK.

Note: The Compiler Options tab of the Application Options dialog box is identical to the Compiler Options tab of the System Settings dialog box except for fact that the application-specific version has an active Reset button. Use this button at any time to reset the compiler options for the selected application to the project-wide, default compiler options.

Compiler Warnings

This group of options allows you to select the level of warnings that you want the compiler to generate.

The All, High, Low, and None options range from every type of warning message being generated to no warning messages at all.

Optimization

This group of options allows you to control the amount and type of optimization that the compiler performs when generating code.

The left column of radio buttons (High, Medium, Low, and None) lets you identify the degree of optimization that should take place. (The compiler tries to generate efficient code that is fairly compact.)

The Speed / Size radio buttons allow you to decide whether the compiler should try to bias code generation towards performance or towards code size. Your choice will necessarily be influenced by the use of undeclared variables in your code.

Runtime Checking

This list of check boxes allows you to decide whether certain conditions will generate compiler and / or runtime errors.

Overflow	Generates checks for numeric overflow conditions, if selected.
Range	Generates checks for attempts to access array elements outside the current size of the array, if selected. This option applies to dimensioned arrays only—not dynamic arrays (see “Arrays” in the *Programmer’s Guide* for more information on these two types of arrays).
Class	Generates checks to ensure that objects assigned to object variables are of the right class, if selected.

Note that whether error messages are displayed at compile time or runtime can depend on the code. For example, if you have selected Range Checking, the following code generates a compiler error:

```
LOCAL DIM x[1]
x[2]:= 5
```

However, this code results in a runtime error since it is only at runtime that the compiler checks to see that the value of *i* is within the bounds of the array:

```
LOCAL DIM x[1]
i := 2
x[i] := 5
```

CA-Clipper Compatibility

This group of check boxes allows you to specify what level of CA-Clipper compatibility the compiler should allow.

Undeclared Variables

Allows the use of CA-Clipper-style variables without declaring them first. If this box is not checked, any reference to an undeclared variable is flagged as a compiler error.

Old Style Assignments

Allows you to have the equal sign (=) as an assignment operator. If this box is not checked, any use of “=” is considered the equality comparison operator.

For a list of valid assignment operators, see the “Operators and Expressions” chapter of the *Programmer’s Guide.*

Integer Divisions

Permits the division of two integers to yield a floating point result. If this box is not checked, the division of two integers will always be an integer, and the remainder will be discarded.

PROCNAME/PROCLINE

Enables runtime support of ProcName() and ProcLine() function calls. If the box is not checked, these function calls will be compiled but will not run.

Note: Selecting the Debug option in the Properties dialog box causes the application to behave as if the PROCNAME / PROCLINE option is selected, even if it is not.

Important! *You must set this option if you wish to use the error messages displayed by the runtime system to lead you to bugs in your code.*

General Options

This group of check boxes controls additional compiler options.

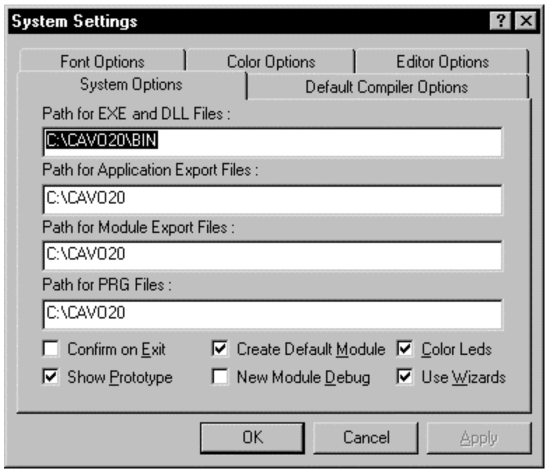
Type Inference	If checked, enables the compiler to infer the data type of undeclared variables by looking at their usage; the compiler attempts to determine the data type of a variable and generate more efficient code for it, if possible.
Operator Methods	If checked, the compiler will convert certain operations to method invocations. For details, refer to the "Objects, Classes, and Methods" chapter in the *Programmer's Guide.*
Full Runtime Class Information	If checked, includes symbolic runtime information in the executable for the PROTECT and HIDDEN instance variables, allowing functional access to these instance variables using the IVarGet() and IVarPut() functions.
Warning as Errors	If checked, treats warnings as errors—that is, an entity is not considered to be successfully compiled if a warning has occurred.

Setting System Options

Note: As mentioned earlier, the System Settings dialog box has been updated in CA-Visual Objects 2.7, providing new system, application, and Source Code Editor options. The System tab page, for example, now offers a new system option, Debug AutoStart. Another option, Use Wizard, has a new value, and still another, Show Prototype, has been moved to the new Explorer tab page. For more detailed information, see the online help.

CA-Visual Objects 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.

To set or override the default system options, select the System Options tab in the System Settings dialog box:



Path for EXE and DLL Files : C:\CAVO20\BIN

Path for Application Export Files : C:\CAVO20

Path for Module Export Files : C:\CAVO20

Path for PRG Files : C:\CAVO20

☐ Confirm on Exit ☑ Create Default Module ☑ Color Leds

☑ Show Prototype ☐ New Module Debug ☑ Use Wizards

Default Path Options

Path for EXE and DLL Files	Contains the default directory in which to place generated .EXE and .DLL files.
Path for Application Export Files	Contains the default directory in which to place generated .AEF files.
Path for Module Export Files	Contains the default directory in which to place generated .MEF files.
Path for PRG Files	Contains the default directory in which to look for .PRG files when importing and exporting within the Source Code Editor. This setting also determines where the compiler searches for icon (.ICO) files declared with the RESOURCE ICON statement.

Miscellaneous System Options

Confirm on Exit — By default, the Confirm on Exit feature is *not* enabled. Select this option if you want a verification dialog box to display when you exit CA-Visual Objects.

Show Prototype — By default, CA-Visual Objects displays prototypes for entities (like classes, methods, and functions) in the status bar. You can suppress this display by deselecting this option.

Create Default Module — By default, CA-Visual Objects creates and loads an empty default module automatically every time you create a new application. You can override this option by deselecting it.

New Module Debug — By default, the system does *not* 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, CA-Visual Objects displays LED-style indicators for the compilation status of entities, modules, and applications in the Error Browser. You can override this option by deselecting it; compilation status will then be indicated appropriately by a character. The following table describes the status for each indicator in the Error Browser:

Indicator	**Status**
Red LED or “X”	Entity contains compilation errors.
Yellow LED or “?”	Warning messages were generated during compilation.

Use Wizard — If selected, this option takes advantage of CA-Visual Objects wizard technology for creating applications.

Note: One of the more noticeable changes in CA-Visual Objects 2.7 is that the Application Wizard has been replaced by the new Application Gallery as the preferred method for creating applications. Therefore, the Use Wizard option is now disabled by default. For detailed information about the Application Gallery feature and the many new predefined application frameworks that it offers, see the online help.

Saving the Current Desktop

The Save Desktop command on the File menu allows you to save the current configuration of all *open browsers* and *editors*. The next time you start CA-Visual Objects, all browsers in the current configuration will be opened and arranged as they were when you selected the Save Desktop command.

Chapter

3

Using the Repository Explorer

This chapter explains how to use CA-Visual Objects Repository Explorer. You will learn how to:

- Navigate within the Repository Explorer
- Customize the Repository Explorer
- Create, view, and modify projects, applications, modules, and entities

The Repository Explorer

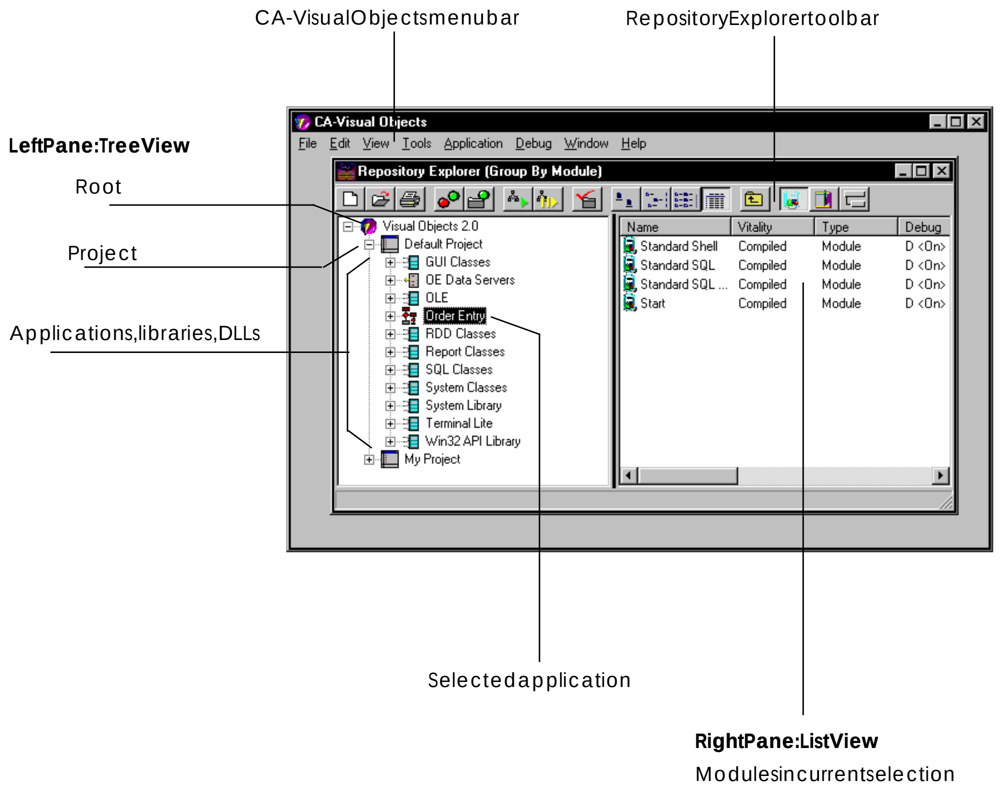
Top-Down Hierarchy

In CA-Visual Objects, *projects* consist of *applications*, *applications* consist of *modules*, and *modules* consist of *entities*. As explained earlier, the Repository Explorer tree structure follows this same top-down hierarchy. When you start CA-Visual Objects, the Repository Explorer is automatically loaded.

Clicking on a project in the Repository Explorer's tree view pane displays the applications defined for it, while clicking on an application displays the modules defined for that application. Similarly, clicking on a module displays the entities defined for that module in the Repository Explorer's list view pane.

Lastly, you can control the overall display by using the Group By Module, Group By Type, and Group By Class toolbar buttons. Note that you can customize the Repository Explorer's list view pane by using the Large Icons, Small Icons, List, and Details toolbar buttons. You can also use the View Options menu command to limit the display by name and type. (See Customizing the Repository Explorer for details.)

When you start CA-Visual Objects, the Repository Explorer is automatically loaded. It displays each of the projects, applications, libraries, DLLs, and modules currently stored in the CA-Visual Objects repository. For example:



Name	Vitality	Type	Debug
Standard Shell	Compiled	Module	D <On>
Standard SQL	Compiled	Module	D <On>
Standard SQL ...	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

Note: This default view is referred to as *module view* and is indicated by the depressed Group By Module toolbar button, shown here.

Working in the Repository Explorer

When you are in the Repository Explorer, you can:

- Create, copy, rename, delete, print, import and export, build, and debug applications, libraries, DLLs, modules, and entities
- Access the various editors, the Debugger, and the Error Browser

Repository Explorer Graphics

In the Repository Explorer, icons are used to pictorially represent projects, applications, modules, and entities in both the tree view pane and the list view pane. Each graphic indicates an item's level in the tree structure, as well as its type:

Graphic	Represents
🗔	Project
🔀	Application
🗄	CA-Visual Objects Library
🗄	User-Defined Library
🗄	DLL
🫙	Module - Internal (a module stored in the CA-Visual Objects repository)
🔺	Module - External (a module not stored in the repository, but rather in an external source file)
📝	Entity - Form
🗔	Entity - Binary Menu
📒	Entity - Source and Resource
▥	Entity - DB Server
▥	Entity - SQL Server
📁	Entity - Field Spec
🖨	Entity - Report
▦	Entity - Cursor, Icon, Bitmap, Ribbon

Note: You can assign a customized icon to an application using the Application Properties dialog box's Icon option, if you did not do so when using the New Application wizard initially to create the application. See Choosing an Icon later in this chapter for more information.

Compilation Status

If an application, module, or entity is new to the tree structure, its uncompiled status is denoted by a red X, as well as the notation "Uncompiled" in the Vitality column of the list view pane of the Repository Explorer. An item that is already compiled is denoted only by the notation "Compiled" in the Vitality column, and a binary menu entity is always denoted by the "Non-Compilable" notation.

In the Error Browser, compilation status is indicated using LED indicators, by default. *Red* indicates that there are compilation errors, or that the application has been modified and needs to be recompiled. *Yellow* indicates that warning messages were generated during compilation.

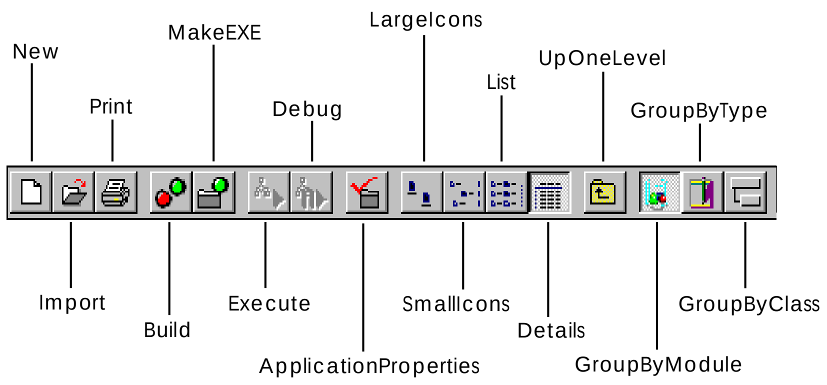
Note: The icons for errors and warnings have been updated slightly in this version of CA-Visual Objects. An error is now indicated by a red circle with an "E" inside it, and a warning and its severity level by a yellow or white circle with a number.

Source Control Icons

Note: In CA-Visual Objects 2.7, if the Show Source Control Icons in List View option is selected and if you have a Microsoft Common Source Control Interface compliant source control system installed, "lock" icons appear before each entity under source control in the Repository Explorer's list view. For detailed information about source code control, the source control icons, and the Show Source Control Icons in List View option, see the online help.

The Toolbar

The Repository Explorer toolbar contains the following buttons:



The New, Import, Print, Application Properties, Large Icons, Small Icons, List, Details, Up One Level, Group By Module, Group By Type, and Group By Class buttons are described later in this chapter. See **Chapter 2: Working in the** Desktop for information about Build, Make EXE, and Execute. See **Chapter 11: Debugging Your Applications** for information about Debug.

Tip: For a quick description of any toolbar button, simply point to it—a tooltip window with descriptive text pops up right next to the button.

Note: Some of the Repository Explorer's commands are also available on local pop-up menus. Others, such as Rename, List, and Edit All Source in Module, are available *only* via the local pop-up menus.

Navigating Basics

Navigation within the Repository Explorer is flexible and easy!

Up One Level

You can use the Up One Level toolbar button at any time to move up a level in the Repository Explorer's hierarchy and display that level's items. (Note that this button is also available in many dialog boxes as an aid in locating files, directories, and drives.)

Scrolling

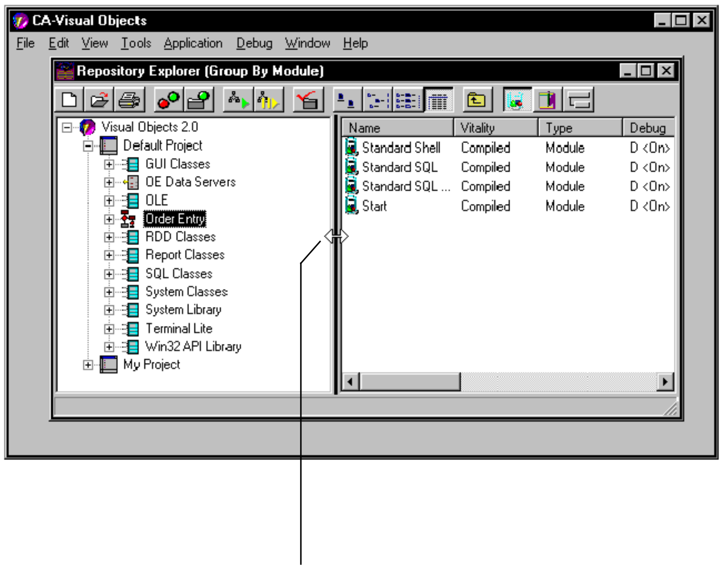
You can use the scroll bars and / or the Direction keys to scroll through the lists of items displayed in both the left (Tree View) and right (List View) panes of the Repository Explorer.

Moving from Pane to Pane

You can move from pane to pane by clicking the desired side with the mouse.

Resizing the Panes

The width of the left and right panes in the Repository Explorer can be adjusted at any time by placing the mouse pointer over the split bar:



Name	Vitality	Type	Debug
Standard Shell	Compiled	Module	D <On>
Standard SQL	Compiled	Module	D <On>
Standard SQL ...	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

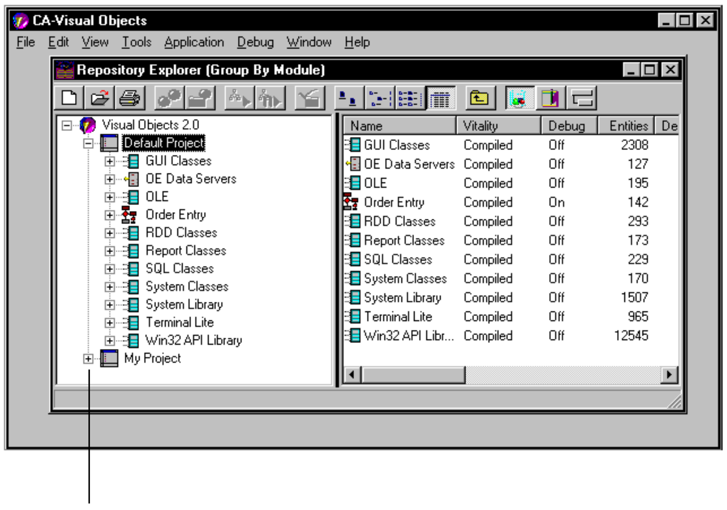
Position the pointer over split bar
to activate the double arrow pointer

This activates the split bar so that a double arrow pointer appears. You can also choose the Split command from the Window menu to activate the split bar. When the double arrow pointer is displayed, hold down the left mouse button and drag the mouse to move the split bar to the desired location, then release the mouse button.

Tip: Resize the Repository Window itself by placing the mouse pointer over the lower right-hand corner of the window, activating a diagonal double-arrow pointer. Hold down the left mouse button and drag the mouse to enlarge or reduce the window simultaneously in two directions.

Browsing Projects, Applications, and Modules

When the Repository Explorer tree initially appears, all branches in the tree are *collapsed* (or *condensed*) except for the Visual Objects root and the Default Project. If there are any other projects, their subitems are hidden from view. For example, My Project below is collapsed:



Name	Vitality	Debug	Entities	De
GUI Classes	Compiled	Off	2308	
OE Data Servers	Compiled	Off	127	
OLE	Compiled	Off	195	
Order Entry	Compiled	On	142	
RDD Classes	Compiled	Off	293	
Report Classes	Compiled	Off	173	
SQL Classes	Compiled	Off	229	
System Classes	Compiled	Off	170	
System Library	Compiled	Off	1507	
Terminal Lite	Compiled	Off	965	
Win32 API Libr...	Compiled	Off	12545	

+indicatesacollapsedbranch

Tip: If the Repository Explorer has been closed for any reason, choose the Repository Explorer command from the Tools menu to reopen it. Use this menu command, also, to open multiple copies of the Repository Explorer, if desired.

Expanding the Initial Tree

The Repository Explorer allows you to show the hidden levels in its tree by expanding it several different ways:

- By a single level
- For an entire branch
- For all branches in the tree

If a particular item has subitems (that is, it is expandable), a
+ button appears to the left of the item name. If an item has *no* subitems, no button is displayed.

Note: See Customizing the Repository Explorer for detailed information about restricting the module view, including setting name filters for applications and modules and limiting the types of applications and modules to be displayed.

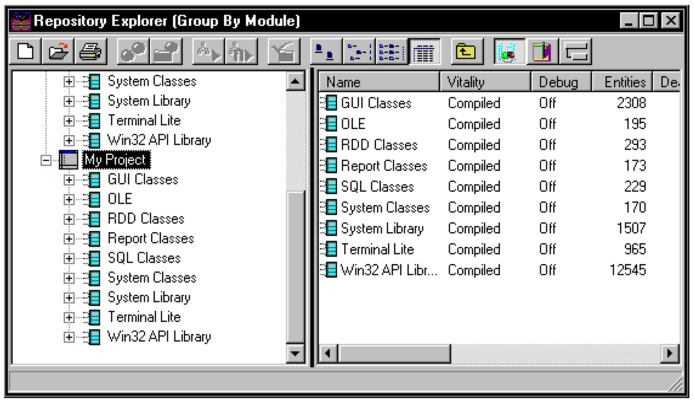
By a Single Level

Expanding an item by a single level allows you to display the *next* level of subitems that belong to that item, without displaying *all* of its subitems. For example, you may want to browse only the applications, libraries, and DLLs that belong to a project, but not their specific modules.

To expand an item by a single level:

1. Choose the branch to be expanded (for example, My Project).
2. Click the + button to the left of My Project, or choose the Expand One Level command from the View menu.

   The tree shows all of the branches at the next level of the selected item:



```
    System Classes
    System Library
    Terminal Lite
    Win32 API Library
  My Project
    GUI Classes
    OLE
    RDD Classes
    Report Classes
    SQL Classes
    System Classes
    System Library
    Terminal Lite
    Win32 API Library
```

Name	Vitality	Debug	Entities	De
GUI Classes	Compiled	Off	2308	
OLE	Compiled	Off	195	
RDD Classes	Compiled	Off	293	
Report Classes	Compiled	Off	173	
SQL Classes	Compiled	Off	229	
System Classes	Compiled	Off	170	
System Library	Compiled	Off	1507	
Terminal Lite	Compiled	Off	965	
Win32 API Libr...	Compiled	Off	12545	

In this instance, all of the CA-Visual Objects libraries that are automatically included when you create a new project are displayed.

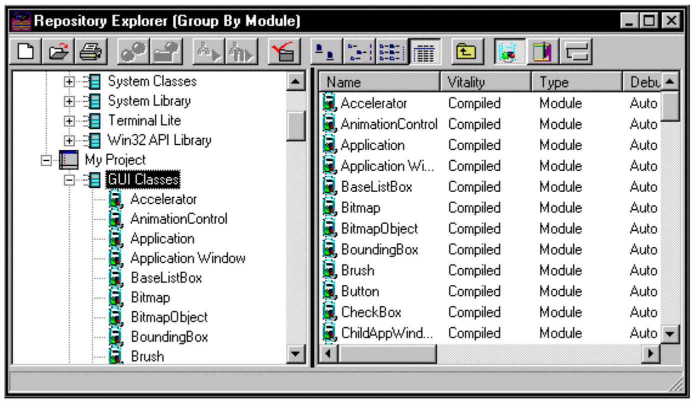
An Entire Branch

Expanding the entire branch of an item allows you to immediately display *all* subitems of a particular item, rather than the intermediate, level-by-level approach offered by the Expand One Level command described above.

To completely expand an item:

1. Select the branch to be expanded (for example, GUI Classes in My Project).
2. Choose the Expand Branch command from the View menu.

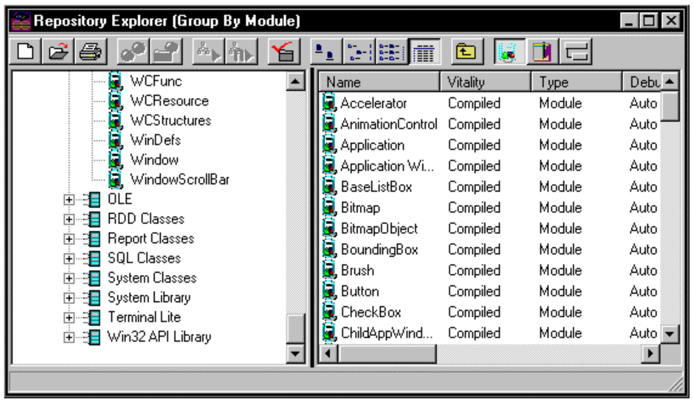
The tree is expanded to show all levels of subitems in the selected item (in this case all of the modules in the GUI Classes library):



```
System Classes
System Library
Terminal Lite
Win32 API Library
My Project
  GUI Classes
    Accelerator
    AnimationControl
    Application
    Application Window
    BaseListBox
    Bitmap
    BitmapObject
    BoundingBox
    Brush
```

Name	Vitality	Type	Debu
Accelerator	Compiled	Module	Auto
AnimationControl	Compiled	Module	Auto
Application	Compiled	Module	Auto
Application Wi...	Compiled	Module	Auto
BaseListBox	Compiled	Module	Auto
Bitmap	Compiled	Module	Auto
BitmapObject	Compiled	Module	Auto
BoundingBox	Compiled	Module	Auto
Brush	Compiled	Module	Auto
Button	Compiled	Module	Auto
CheckBox	Compiled	Module	Auto
ChildAppWind...	Compiled	Module	Auto

3. Use the scroll bar to see that the other libraries in My Project are still collapsed:



```
    WCFunc
    WCResource
    WCStructures
    WinDefs
    Window
    WindowScrollBar
  OLE
  RDD Classes
  Report Classes
  SQL Classes
  System Classes
  System Library
  Terminal Lite
  Win32 API Library
```

Name	Vitality	Type	Debu
Accelerator	Compiled	Module	Auto
AnimationControl	Compiled	Module	Auto
Application	Compiled	Module	Auto
Application Wi...	Compiled	Module	Auto
BaseListBox	Compiled	Module	Auto
Bitmap	Compiled	Module	Auto
BitmapObject	Compiled	Module	Auto
BoundingBox	Compiled	Module	Auto
Brush	Compiled	Module	Auto
Button	Compiled	Module	Auto
CheckBox	Compiled	Module	Auto
ChildAppWind...	Compiled	Module	Auto

All Branches in the Tree

The previous two sections describe different ways to expand the Repository Explorer tree for the *currently selected* branch.

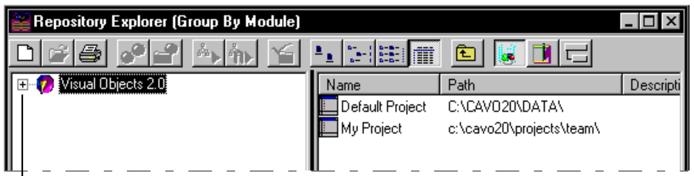
You can also expand *every* branch in the tree to display all available subitems for every item. To do this, select the Expand All command from the View menu.

Note: If the tree is already fully expanded, the Expand All menu command has no effect. Also note that at the project level only, Expand Branch and Expand All have the same effect.

Collapsing the Tree

Just as you can expand the Repository Explorer tree by a single level, for an entire branch, or for all branches in the tree, you can collapse it. *Collapsing* means to hide the additional levels that are below a selected item in the tree.

To condense all or a portion of the tree, use the Collapse One Level, Collapse Branch, and Collapse All commands on the View menu as you would their Expand command counterparts (as described in the previous section). For example, the Repository Explorer shown below is collapsed at the Visual Objects root level, after choosing the Collapse All command:



Name	Path	Descripti
Default Project	C:\CAVO20\DATA\	
My Project	c:\cavo20\projects\team\	

Tree collapsed at root level

Note: You can click the – button to the left of an item as a shortcut to the Collapse One Level command (similar in function to the + button for Expand One Level).

Browsing Entities

Entities form the lowest level in the application hierarchy. An entity is a component that has a distinct name and can be edited. Applications can share entities in libraries.

The basic types of entities available in CA-Visual Objects are:

Form	Binary Menu
DB Server	SQL Server
Field Spec	Report
Source	Resource
Cursor, Icon, Bitmap, Ribbon	

Resource and source entities can be further categorized by *subtype*, as follows:

Source Subtypes	Resource Subtypes
Class	Accelerator
Access	Dialog
Assign	Menu
Method	Icon
Function	Bitmap
Global	Ribbon
Structure (struct)	Version Info
Procedure	
Union	

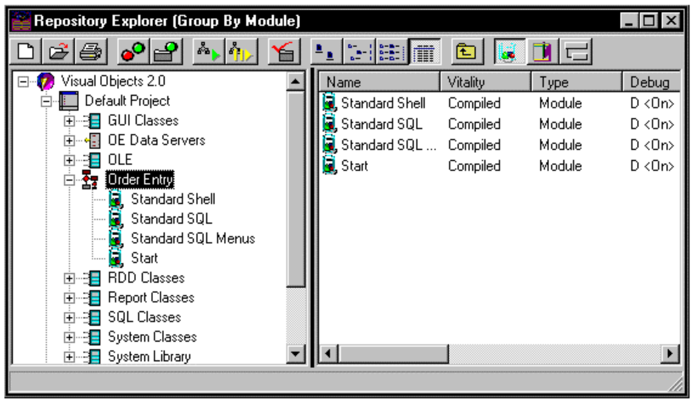
In CA-Visual Objects you can view either all of the entities defined for a specific module, or all of the entities defined for a particular application.

Viewing Entities at the Module Level

To view only the entities defined for a specific module in an application—for example, the Standard SQL Menus module in Order Entry—do the following:

1. Double-click on Order Entry or click the + icon to its left.

   All of the modules defined to Order Entry are displayed in the Repository Explorer tree:



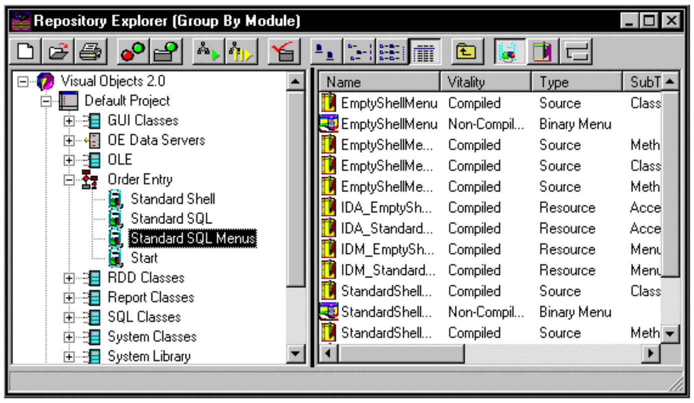
Repository Explorer (Group By Module)

```
Visual Objects 2.0
  Default Project
    GUI Classes
    OE Data Servers
    OLE
    Order Entry
      Standard Shell
      Standard SQL
      Standard SQL Menus
      Start
    RDD Classes
    Report Classes
    SQL Classes
    System Classes
    System Library
```

Name	Vitality	Type	Debug
Standard Shell	Compiled	Module	D <On>
Standard SQL	Compiled	Module	D <On>
Standard SQL ...	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

2. Click on Standard SQL Menus in the tree.

All of the entities belonging to the Standard SQL Menus module are displayed in the right, or List View, pane of the Repository Explorer:



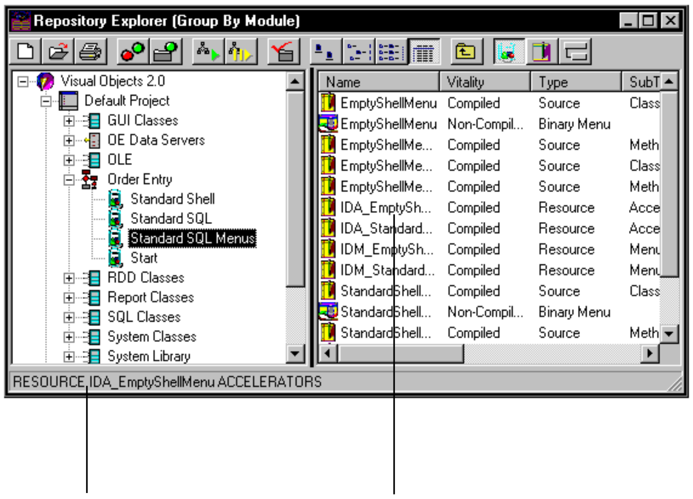
- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
    - OLE
    - Order Entry
      - Standard Shell
      - Standard SQL
      - Standard SQL Menus
      - Start
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library

Name	Vitality	Type	SubT
EmptyShellMenu	Compiled	Source	Class
EmptyShellMenu	Non-Compil...	Binary Menu	
EmptyShellMe...	Compiled	Source	Meth
EmptyShellMe...	Compiled	Source	Class
EmptyShellMe...	Compiled	Source	Meth
IDA_EmptySh...	Compiled	Resource	Acce
IDA_Standard...	Compiled	Resource	Acce
IDM_EmptySh...	Compiled	Resource	Menu
IDM_Standard...	Compiled	Resource	Menu
StandardShell...	Compiled	Source	Class
StandardShell...	Non-Compil...	Binary Menu	
StandardShell...	Compiled	Source	Meth

Notice that the entities are sorted by entity name in alphabetical order. You can, however, sort the entities by other criteria—such as vitality or entity type—simply by clicking on the appropriate column header in the list view pane.

3. Now move the cursor over the list of entities.

Notice that the Repository Explorer's status bar displays the syntactical prototype for each item as the mouse pointer passes over it. For example:



- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
    - OLE
    - Order Entry
      - Standard Shell
      - Standard SQL
      - Standard SQL Menus
      - Start
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library

Name	Vitality	Type	SubT
EmptyShellMenu	Compiled	Source	Class
EmptyShellMenu	Non-Compil...	Binary Menu	
EmptyShellMe...	Compiled	Source	Meth
EmptyShellMe...	Compiled	Source	Class
EmptyShellMe...	Compiled	Source	Meth
IDA_EmptySh...	Compiled	Resource	Acce
IDA_Standard...	Compiled	Resource	Acce
IDM_EmptySh...	Compiled	Resource	Menu
IDM_Standard...	Compiled	Resource	Menu
StandardShell...	Compiled	Source	Class
StandardShell...	Non-Compil...	Binary Menu	
StandardShell...	Compiled	Source	Meth

```
RESOURCE IDA_EmptyShellMenu ACCELERATORS
```

Prototypefor...Resourceentity(acceleratorsubtype)

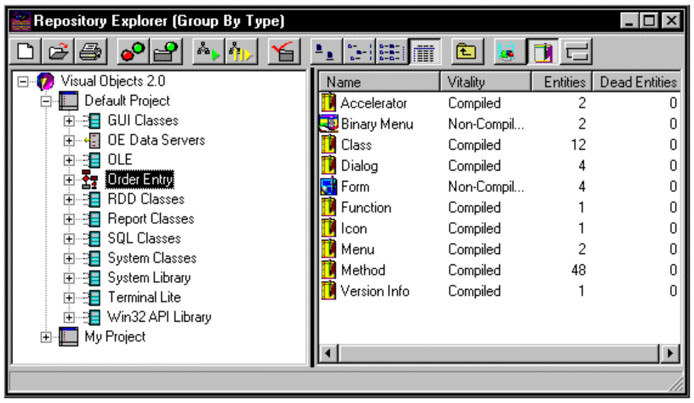
Note: This feature is available only if the Show Prototype option has been selected on the Explorer Tab of the System Settings dialog box. Furthermore, prototypes are not applicable to binary entities.

Viewing Entities at the Application Level

On the other hand, to browse *all* of the entities defined for an application (for example, Order Entry):

1. Select Order Entry again in the Repository Explorer tree.
2. Click the Group By Type toolbar button.

   The Repository Explorer switches from module view to *entity view*:



Repository Explorer (Group By Type)

```
Visual Objects 2.0
  Default Project
    GUI Classes
    OE Data Servers
    OLE
    Order Entry
    RDD Classes
    Report Classes
    SQL Classes
    System Classes
    System Library
    Terminal Lite
    Win32 API Library
  My Project
```

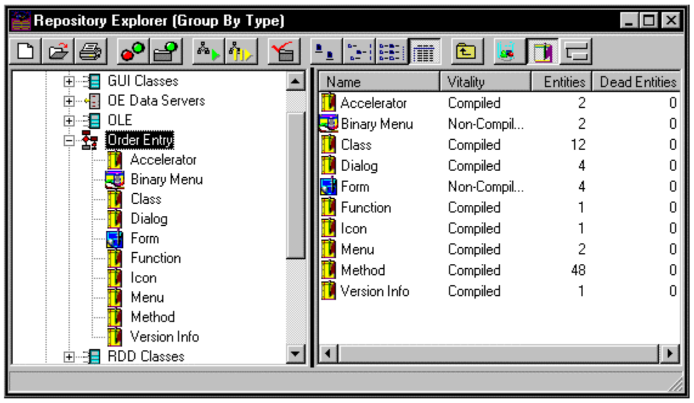
Name	Vitality	Entities	Dead Entities
Accelerator	Compiled	2	0
Binary Menu	Non-Compil...	2	0
Class	Compiled	12	0
Dialog	Compiled	4	0
Form	Non-Compil...	4	0
Function	Compiled	1	0
Icon	Compiled	1	0
Menu	Compiled	2	0
Method	Compiled	48	0
Version Info	Compiled	1	0

Notice that the types of entities defined for Order Entry are arranged alphabetically in the list view pane, and the number of each type is indicated in the Entities column.

Collapsing and Expanding the Display

Expanding

To expand the Repository Explorer tree so that an application's entities are displayed, choose the Expand One Level command from the View menu. For example:

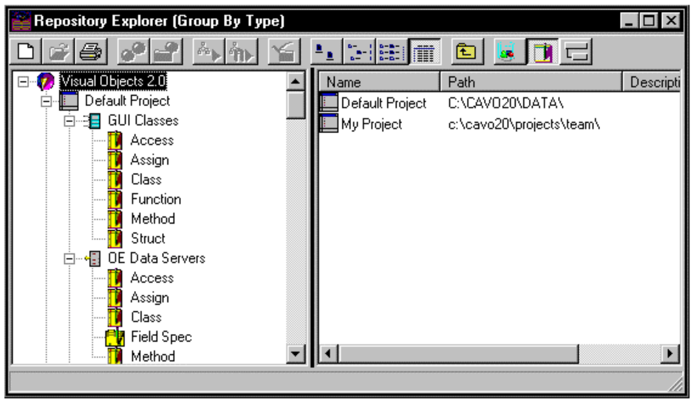


Repository Explorer (Group By Type)

```
GUI Classes
OE Data Servers
OLE
Order Entry
  Accelerator
  Binary Menu
  Class
  Dialog
  Form
  Function
  Icon
  Menu
  Method
  Version Info
RDD Classes
```

Name	Vitality	Entities	Dead Entities
Accelerator	Compiled	2	0
Binary Menu	Non-Compil...	2	0
Class	Compiled	12	0
Dialog	Compiled	4	0
Form	Non-Compil...	4	0
Function	Compiled	1	0
Icon	Compiled	1	0
Menu	Compiled	2	0
Method	Compiled	48	0
Version Info	Compiled	1	0

If you choose Expand All instead, the result is:



Visual Objects 2.0
Default Project
GUI Classes
Access
Assign
Class
Function
Method
Struct
OE Data Servers
Access
Assign
Class
Field Spec
Method

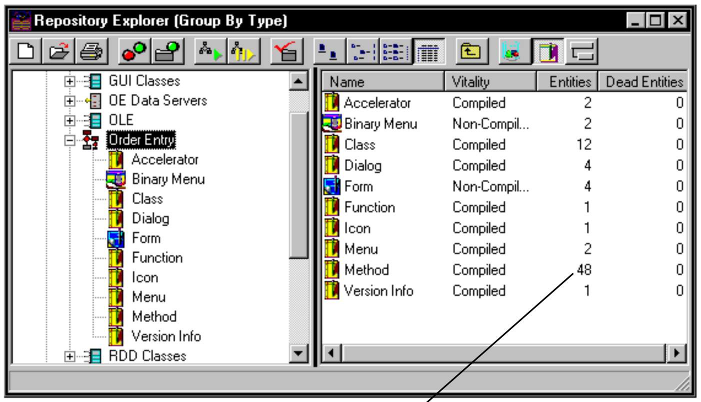
Name	Path	Descripti
Default Project	C:\CAVO20\DATA\	
My Project	c:\cavo20\projects\team\	

Collapsing

Just as you can expand the Repository Explorer tree by a single level, for an entire branch, or for all branches in the tree, you can collapse it. Use the Collapse One Level, Collapse Branch, and Collapse All commands on the View menu as you would their Expand command counterparts.

Viewing Additional Entities

When we used the Expand One Level menu command earlier, the list view pane indicated that there were a total of 48 method entities in Order Entry:

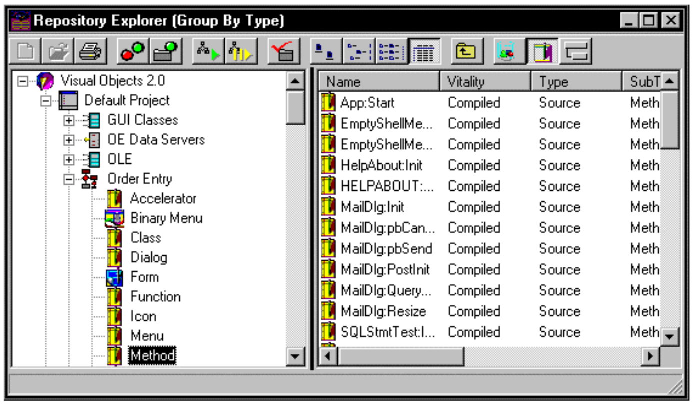


GUI Classes
OE Data Servers
OLE
Order Entry
Accelerator
Binary Menu
Class
Dialog
Form
Function
Icon
Menu
Method
Version Info
RDD Classes

Name	Vitality	Entities	Dead Entities
Accelerator	Compiled	2	0
Binary Menu	Non-Compil...	2	0
Class	Compiled	12	0
Dialog	Compiled	4	0
Form	Non-Compil...	4	0
Function	Compiled	1	0
Icon	Compiled	1	0
Menu	Compiled	2	0
Method	Compiled	48	0
Version Info	Compiled	1	0

NumberofmethodsinOrderEntry

To view these methods, simply click on Method in the tree view pane. The list view pane in the Repository Explorer changes accordingly:



Repository Explorer (Group By Type)

- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
    - OLE
    - Order Entry
      - Accelerator
      - Binary Menu
      - Class
      - Dialog
      - Form
      - Function
      - Icon
      - Menu
      - Method

Name	Vitality	Type	SubT
App:Start	Compiled	Source	Meth
EmptyShellMe...	Compiled	Source	Meth
EmptyShellMe...	Compiled	Source	Meth
HelpAbout:Init	Compiled	Source	Meth
HELPABOUT:...	Compiled	Source	Meth
MailDlg:Init	Compiled	Source	Meth
MailDlg:pbCan...	Compiled	Source	Meth
MailDlg:pbSend	Compiled	Source	Meth
MailDlg:PostInit	Compiled	Source	Meth
MailDlg:Query...	Compiled	Source	Meth
MailDlg:Resize	Compiled	Source	Meth
SQLStmtTest:I...	Compiled	Source	Meth

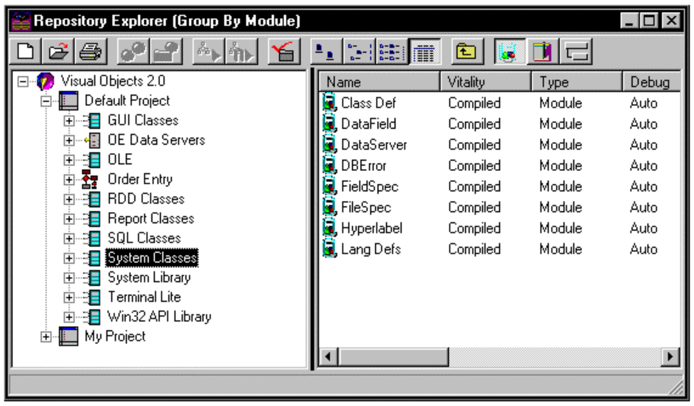
See Customizing the Repository Explorer for detailed information about restricting the entity view, including setting a name filter for entities and limiting the entity types and subtypes to be displayed.

Browsing Classes

Note: There are five new libraries in CA-Visual Objects 2.7: Console Classes, Internet, Internet Server API, and OLE Server. For more detailed information, see the online help.

CA-Visual Objects also allows you to take a comprehensive look at all of the classes defined for or used by an application. For example, to browse the classes in the System Classes library you need to switch from module view to class view. To do so:

1. Highlight System Classes in the Repository Explorer tree:



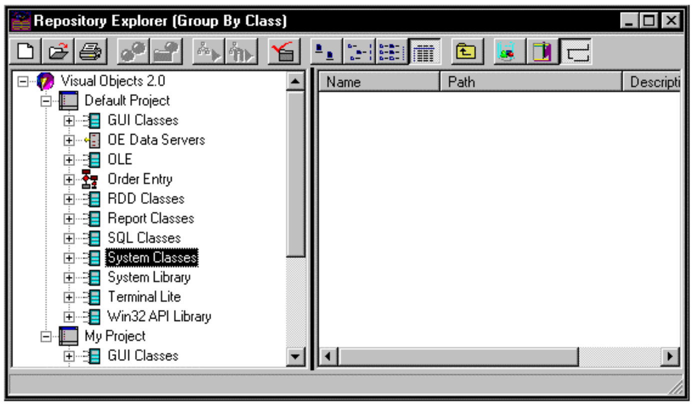
Repository Explorer (Group By Module)

- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
    - OLE
    - Order Entry
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library
    - Terminal Lite
    - Win32 API Library
  - My Project

Name	Vitality	Type	Debug
Class Def	Compiled	Module	Auto
DataField	Compiled	Module	Auto
DataServer	Compiled	Module	Auto
DBError	Compiled	Module	Auto
FieldSpec	Compiled	Module	Auto
FileSpec	Compiled	Module	Auto
Hyperlabel	Compiled	Module	Auto
Lang Defs	Compiled	Module	Auto

2. Click the Group By Class toolbar button.

   The Repository Explorer switches from module view to class view:



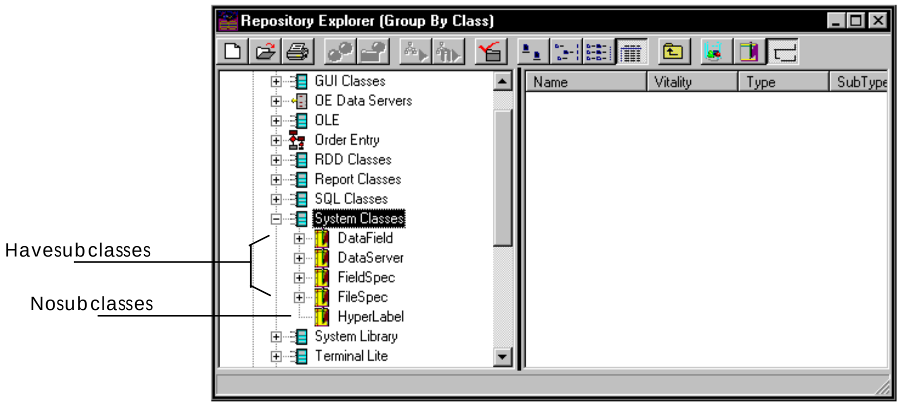
   Notice that all branches in the tree are collapsed and that the names of the columns in the list view pane have changed.

Expanding the Display

To view the classes and subclasses belonging to an application, use the Expand One Level, Expand Branch, and Expand All menu commands which you are already familiar with.

By a Single Level

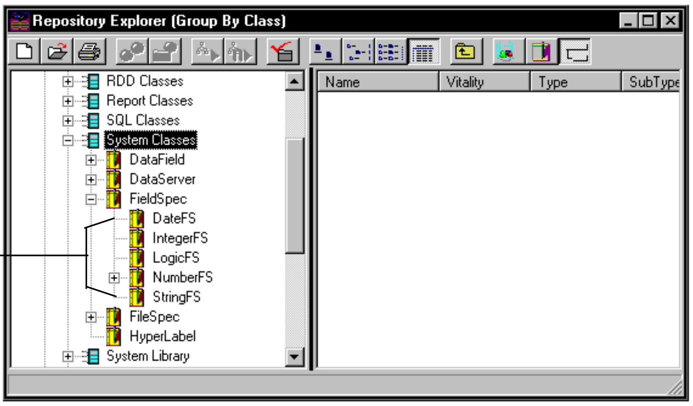
For example, to display the classes that belong to the System Classes library, click the + button to the left of System Classes, or choose the Expand One Level command from the View menu. The Repository Explorer tree now displays the topmost level
of classes:



Notice that if a particular class has subclasses (that is, it is expandable), a + button appears to the left of the class name.
(If a class has *no* subclasses, no button is displayed.)

To expand a class—for example, FieldSpec—simply click on its + button or choose Expand One Level again. The Repository Explorer tree changes accordingly:

Subclasses of FieldSpec class

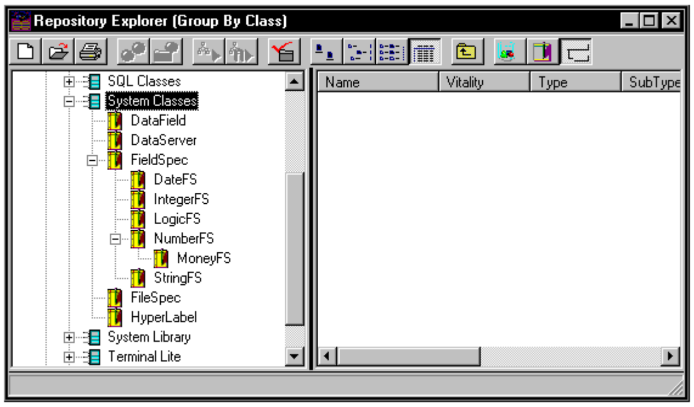


An Entire Branch

To view *all* of the subclasses of a particular class, use the Expand Branch command instead of expanding each branch one level at a time using Expand One Level, as we did above. For example:

1. Select the System Classes module again.
2. Choose the Expand Branch command from the View menu.

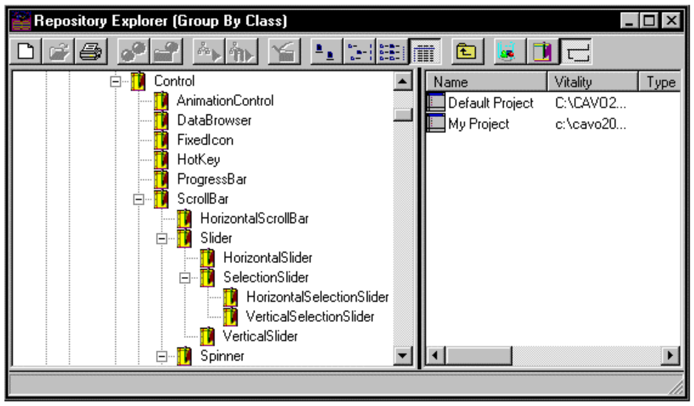
   The tree is expanded to show all levels of classes and subclasses in the System Classes library:



Note: If the selected class has no subclasses or is already completely expanded (indicated by a – button), the Expand Branch command has no effect.

All Branches in the Tree

You can also expand *every* branch in the tree to display all available subclasses for every class. To do this, select the Expand All command from the View menu. For example:



Name	Vitality	Type
Default Project	C:\CAVO2...	
My Project	c:\cavo20...	

Note: If the tree is already fully expanded, the Expand All menu command has no effect.

Collapsing the Tree

Just as you can expand the Repository Explorer tree by a single level, for an entire branch, or for all branches in the tree in class view, you can collapse it. Collapsing means to hide the additional class levels that are below a selected class in the tree.

To condense all or a portion of the tree, use the Collapse One Level, Collapse Branch, and Collapse All commands on the View menu as you would their Expand command counterparts.

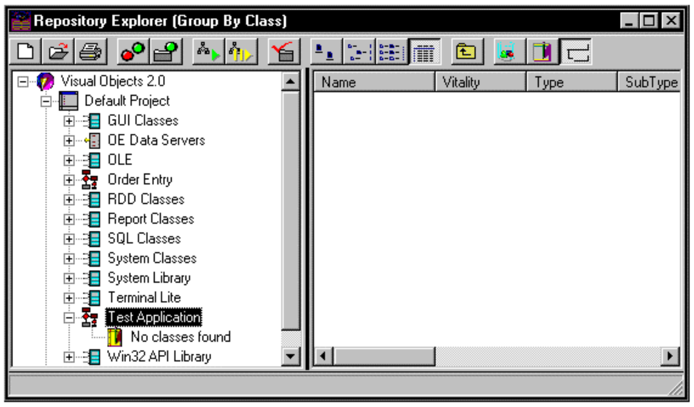
Note: You can click the – button to the left of a class name as a shortcut to the Collapse One Level command (similar in function to the + button for Expand One Level).

Showing Additional Classes

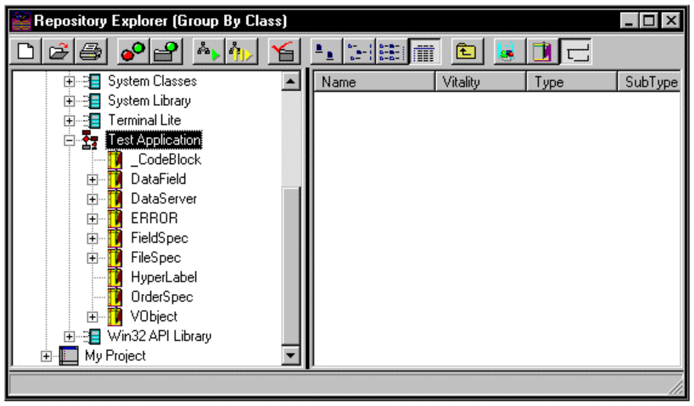
By default, the Repository Explorer initially displays only the classes that are defined to the modules in the current application. To view additional classes with which the current application may be associated, select the Include Libraries command from the View menu.

Tip: An application's search path—including libraries—is set when the application is created. You can, however, use the Properties command on the Application menu or the local pop-up menu to add libraries to or remove libraries from the current search path.

For example, suppose you create an MDI application named Test Application that includes the default libraries—GUI Classes, RDD Classes, and System Classes—in its search path and contains only the standard application framework. The Repository Explorer tree would initially look as follows in class view when first displayed for Test Application:



If you then choose the Include Libraries menu command, classes belonging to the default libraries are merged and arranged in proper hierarchical order in a single tree:



To hide the additional classes, deselect the View Include Libraries menu command.

Viewing Inherited Properties and Classes

By default, the list view pane of the Repository Explorer displays only those properties and methods that the currently selected class *owns*. That is, the properties and methods it may have inherited from its superclasses are not shown.

You can optionally display all properties and methods that are *inherited* by the currently selected class by selecting the Include Inherited command from the View menu.

To hide the inherited properties and methods, deselect the View Include Inherited menu command.

Customizing the Repository Explorer

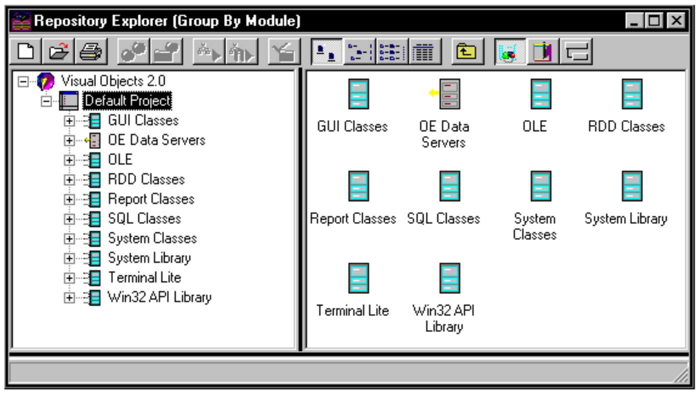
By default, the Repository Explorer initially displays everything—applications, libraries, and DLLs. You can, however, customize the Repository Explorer's initial display in several ways: specifying the size of the icons used, displaying data in list or detailed format, and restricting the display to a specified application type(s) and / or name(s). This is in addition to selecting module, type, or class view by clicking the Group By Module, Group By Type, or Group By Class toolbar button, respectively.

Icons

You can specify the size of the icons used to represent projects, applications, modules, and entities in the list view pane of the Repository Explorer.

Large Icons

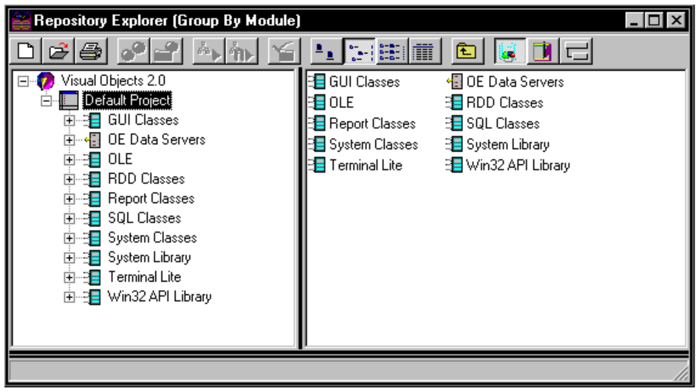
For large icons, simply click the Large Icons toolbar button. For example:



Tip: To customize the display, choose the Arrange Icons command from the View menu to arrange the icons by name, vitality, creation date, and so on.

Small Icons

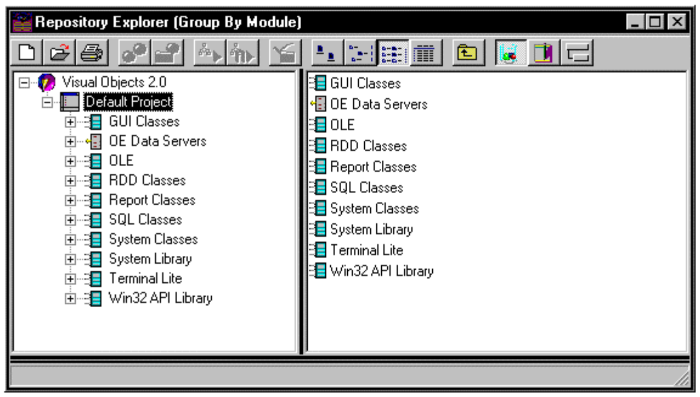
For small icons, simply click the Small Icons toolbar button. For example:



List and Details Toolbar Buttons

List Button

To view the data in the list view pane in list format, click the List toolbar button. For example:



Repository Explorer (Group By Module)

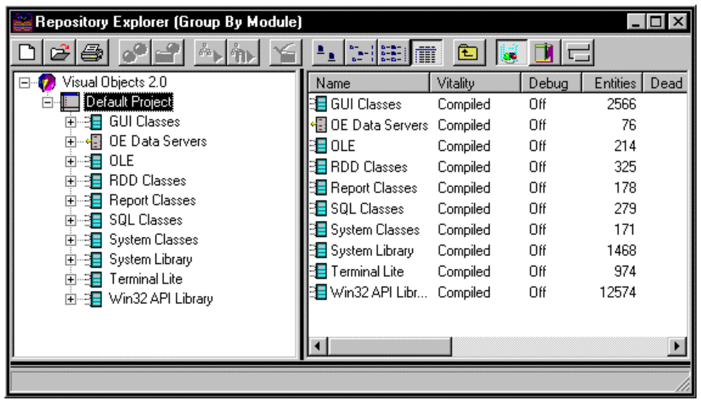
- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
    - OLE
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library
    - Terminal Lite
    - Win32 API Library

GUI Classes
OE Data Servers
OLE
RDD Classes
Report Classes
SQL Classes
System Classes
System Library
Terminal Lite
Win32 API Library

Notice that the only data in the list view pane are application and library names and their corresponding icons.

Details Button

To view the data in the list view pane in detailed format, click the Details toolbar button. For example:



Repository Explorer (Group By Module)

- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
    - OLE
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library
    - Terminal Lite
    - Win32 API Library

Name	Vitality	Debug	Entities	Dead
GUI Classes	Compiled	Off	2566	
OE Data Servers	Compiled	Off	76	
OLE	Compiled	Off	214	
RDD Classes	Compiled	Off	325	
Report Classes	Compiled	Off	178	
SQL Classes	Compiled	Off	279	
System Classes	Compiled	Off	171	
System Library	Compiled	Off	1468	
Terminal Lite	Compiled	Off	974	
Win32 API Libr...	Compiled	Off	12574	

Notice that there is now much data in the list view pane formatted in columns. You can control the types of data displayed by hiding columns selectively. See Setting View Options below.

Setting View Options

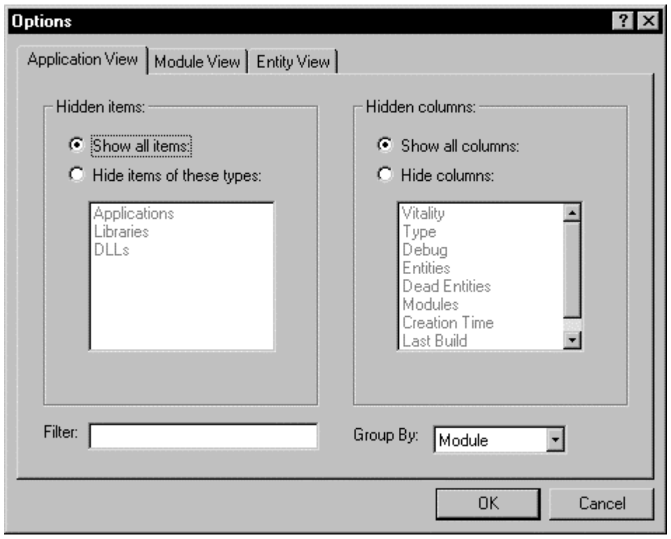
You can also customize the Repository Explorer by limiting the display of applications, modules, and entities by type or name, as well as to specify the particular subsets of data to be displayed in its list view pane.

Limiting by Type

For example, to limit the CA-Visual Objects display to applications only:

1. Select the Options command from the View menu.

   The Options dialog box appears:

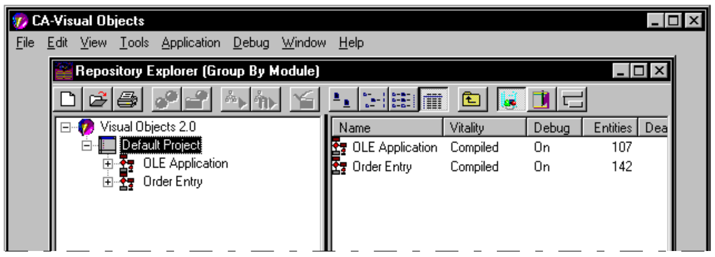
   Notice that there are three tabs: Application View, Module View, and Entity View.

2. Select the Application View tab.
3. Choose Hide Items of These Types from the Hidden Items radio button group.
4. Highlight Libraries, DLLs, and System Libraries.

   Note: System Libraries is new with CA-Visual Objects 2.7.

5. Click OK.

The Repository Explorer now displays only applications:


CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Visual Objects 2.0
Default Project
OLE Application
Order Entry

Name	Vitality	Debug	Entities	Dea
OLE Application	Compiled	On	107	
Order Entry	Compiled	On	142	

Tip: Since you need to access libraries and DLLs less frequently than your applications, limiting the display to just applications results in an uncluttered Repository Explorer.

Similarly, you can limit the types of modules or entities displayed by clicking on the appropriate tab in the Options dialog box and setting options for Hidden Items. (See the online help system for complete descriptions of the available options for this dialog box.)

Limiting by Name

Using the Name Filter

You can also limit the display to applications with a certain name. For example, if you want to view only applications and libraries whose names begin with the acronym “OLE”:

1. Select the Options command from the View menu again.

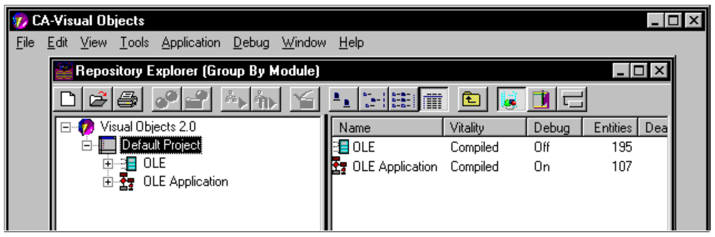
   The Options dialog box appears.

2. Select the Application View tab.
3. Choose Hide Items of These Types from the Hidden Items radio button group.
4. Highlight DLLs.
5. Enter **OLE** in the name filter—that is, the Filter edit control—in the Options dialog box.

   Note: No wild cards are allowed.

6. Press Enter.

The Repository Explorer changes accordingly, displaying the following OLE applications and libraries:


CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Visual Objects 2.0
Default Project
OLE
OLE Application

Name	Vitality	Debug	Entities	Dea
OLE	Compiled	Off	195	
OLE Application	Compiled	On	107	

Restoring the Display

To restore the Repository Explorer window to its original display:

1. Select the Options command from the View menu once more. The Options dialog box appears.
2. Select the Application View tab.
3. Highlight the contents in the filter.
4. Press the Del key to delete the filter.
5. Click OK.

Note: You can also limit modules and entities by name using the name filter in the appropriate tab in the Options dialog box.

Limiting Details

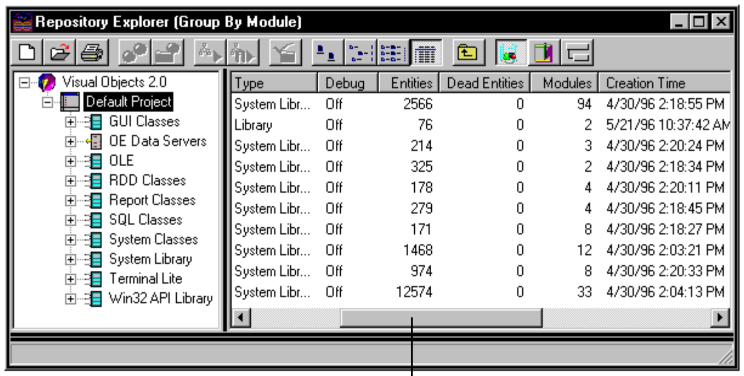
Additionally, you can customize the Repository Explorer by specifying the following types of data to be displayed in the list view pane:

Data Type	Description
Name	Application name.
Vitality	Compilation status.
Type	Application type - application, library, or DLL.
Debug	Debugging status.
Entities	Number of entities in application.
Dead Entities	Number of dead entities.
Modules	Number of modules in application.
Creation Time	Date/time application was created.
Last Build	Date/time application last compiled.
Description	Brief description of application.

Note: Similar data can be specified for modules and entities, as well.

Scrolling the List View Pane

For example, if you enlarge the list view pane shown below and scroll through it, you will see that all of the possible types of data about each application in the Default Project are displayed:



Type	Debug	Entities	Dead Entities	Modules	Creation Time
System Libr...	Off	2566	0	94	4/30/96 2:18:55 PM
Library	Off	76	0	2	5/21/96 10:37:42 AM
System Libr...	Off	214	0	3	4/30/96 2:20:24 PM
System Libr...	Off	325	0	2	4/30/96 2:18:34 PM
System Libr...	Off	178	0	4	4/30/96 2:20:11 PM
System Libr...	Off	279	0	4	4/30/96 2:18:45 PM
System Libr...	Off	171	0	8	4/30/96 2:18:27 PM
System Libr...	Off	1468	0	12	4/30/96 2:03:21 PM
System Libr...	Off	974	0	8	4/30/96 2:20:33 PM
System Libr...	Off	12574	0	33	4/30/96 2:04:13 PM

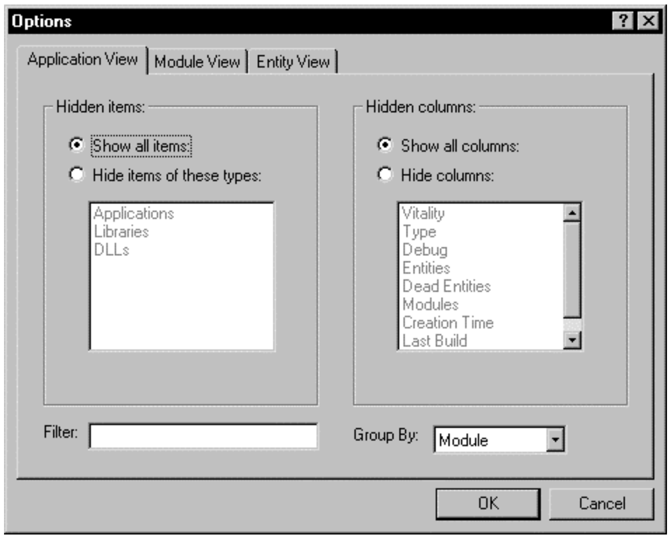
Usescrollbartoviewallcolumns

Limiting the Display

Suppose, however, you want to limit the display to just Name, Vitality, Type, Debug, Entities, and Modules. To do this:

1. Select the View Options menu command.

   The Options dialog box appears:



Hidden items: [✓ Show all items] / Hide items of these types:
Applications
Libraries
DLLs

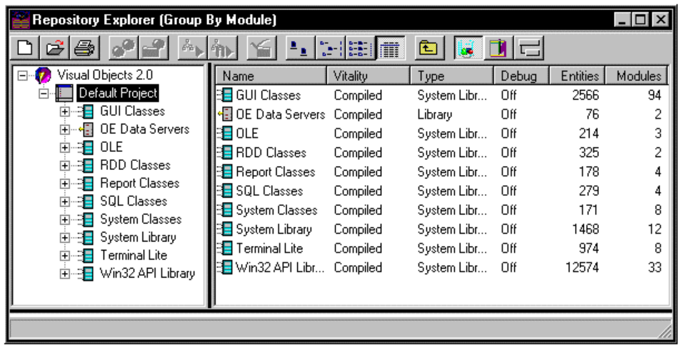
Hidden columns: [✓ Show all columns] / Hide columns:
Vitality
Type
Debug
Entities
Dead Entities
Modules
Creation Time
Last Build

Filter:

Group By: Module

2. Select the Application View tab, if it is not already selected.
3. Choose Hide Columns from the Hidden Columns radio button group.
4. Highlight the following items: Dead Entities, Creation Time, Last Build, and Description.
5. Click OK.

The Repository Explorer changes accordingly:



Repository Explorer (Group By Module)

- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
    - OLE
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library
    - Terminal Lite
    - Win32 API Library

Name	Vitality	Type	Debug	Entities	Modules
GUI Classes	Compiled	System Libr...	Off	2566	94
OE Data Servers	Compiled	Library	Off	76	2
OLE	Compiled	System Libr...	Off	214	3
RDD Classes	Compiled	System Libr...	Off	325	2
Report Classes	Compiled	System Libr...	Off	178	4
SQL Classes	Compiled	System Libr...	Off	279	4
System Classes	Compiled	System Libr...	Off	171	8
System Library	Compiled	System Libr...	Off	1468	12
Terminal Lite	Compiled	System Libr...	Off	974	8
Win32 API Libr...	Compiled	System Libr...	Off	12574	33

Notice that now there is no horizontal scroll bar.

Note: You can also limit the type of data displayed for modules and entities by clicking on the appropriate tab in the Options dialog box and setting options for Hidden Columns. (See the online help system for complete descriptions of the available options for the Options dialog box.)

New Customization Options

Note: As mentioned in Setting System Options in the previous chapter, the System Settings dialog box has been updated in CA-Visual Objects 2.7. Its new Explorer tab page, for example, now offers the following options for the Repository Explorer: Show Prototype, Select Entire Row, Grid Lines, Track Selection, AutoSize Name Column, and Show Source Control Icons in List View. For detailed information about these options, see the online help.

Managing Projects

Note: CA-Visual Objects 2.7 also provides version control for managing your applications with its new Source Code Control Interface. For complete information, see the online help.

CA-Visual Objects allows you to group together all of a repository's applications, libraries, and DLLs as a single project. Because CA-Visual Objects is a repository-based system, you can even create multiple projects that access separate repositories.

All projects currently available to you are managed through a *project catalog*. A project can only belong to one catalog at a time. When you create a new project, it is added automatically to your catalog. If you wish, however, to share your work with other members of a development team, you can remove the project from your catalog. Note that this action does not *delete* the project's directory; it only removes the project from your Repository Explorer's window. This allows another developer to add the existing project to his or her own catalog.

Note: To view a project's properties, access the Project Properties dialog box directly by using the Alt+Return key combination, or by right-clicking and selecting the Properties command from a local pop-up menu.

For additional information about projects, see **Exchanging Projects** in the "Importing and Exporting Files" chapter later in this guide.

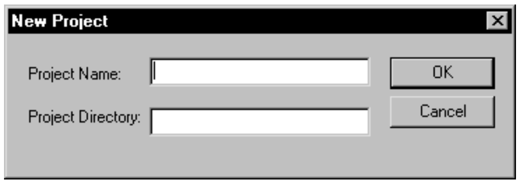
Creating a Project

To create a project:

1. From the root level of the Repository Explorer, click the New toolbar button.

   Alternatively, select the New Project command from the File menu.

   The New Project dialog box appears:

   In CA-Visual Objects 2.7, there is an additional button on this screen that allows you to browse your existing folders.

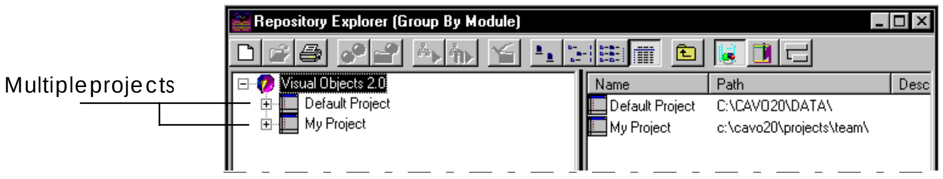
2. Enter a name in the Project Name edit control (for example, **My Project**).

   Project names can be up to 30 characters long including spaces and special characters.

3. In the Project Directory edit control, enter the path for the repository with which the project is to be associated (for example, **C:\CAVO27\TEAM\PROJECTS**).

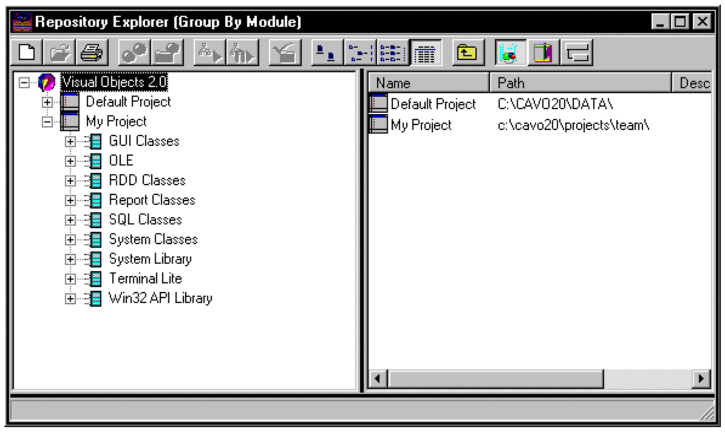
4. Click OK.

   The new project is added to the Repository Explorer tree structure:



Name	Path	Desc
Default Project	C:\CAVO20\DATA\	
My Project	c:\cavo20\projects\team\	

If you now click on the + icon to the left of My Project, you can see that the new project has access to the CA-Visual Objects predefined system libraries:



Name	Path	Desc
Default Project	C:\CAVO20\DATA\	
My Project	c:\cavo20\projects\team\	

These predefined libraries, which reside usually in the CAVO27\SYSTEM subdirectory, are shared by *all* projects. They cannot be modified, as they are completely read-only.

5. Proceed to add applications, user-defined libraries, and DLLs to the new project.

   Note that all user-defined components of a project will reside in the specified project directory.

Deleting Projects

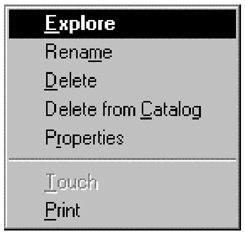
In CA-Visual Objects you can either remove a project from a catalog or delete it completely from the repository.

Deleting from a Catalog

To remove a project from your catalog and the Repository Explorer *without* deleting it from the repository itself:

1. Right-click on the project in the Repository Explorer (for example, My Project).

   The local pop-up menu appears:



2. Choose the Delete from Catalog command.

   Note: The Delete from Catalog menu command is available only via the local pop-up menu.

The project is deleted from your catalog and removed from the Repository Explorer.

Deleting from the Repository

To delete a project in its entirety from the repository:

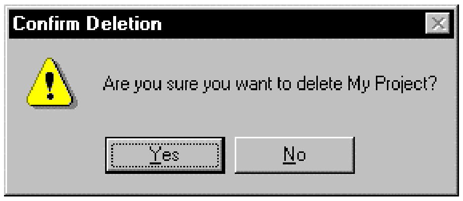
1. Right-click on the project in the Repository Explorer (for example, My Project).

   The local pop-up menu appears.

   (Alternatively, select the Delete command from the Edit menu.)

2. Choose Delete from the local pop-up menu.

   The standard Confirm Deletion dialog box reappears:



Are you sure you want to delete My Project?

3. Click Yes.

   The project is deleted from the repository, and its directory is also deleted.

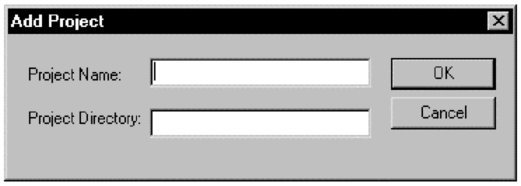
Note: See your online help system for information about the other local pop-up menu commands.

Adding a Project

To add someone else's project to your own catalog:

1. From the root level of the Repository Explorer, select Add Project from the File menu.

   The Add Project dialog box appears:



Project Name:
Project Directory:

   In CA-Visual Objects 2.7, there is an additional button on this screen that allows you to browse your existing folders.

2. Enter a name in the Project Name edit control.

3. In the Project Directory edit control, enter the path for the repository with which the existing project is associated.

4. Click OK.

The specified project is added to the Repository Explorer tree structure.

Renaming a Project

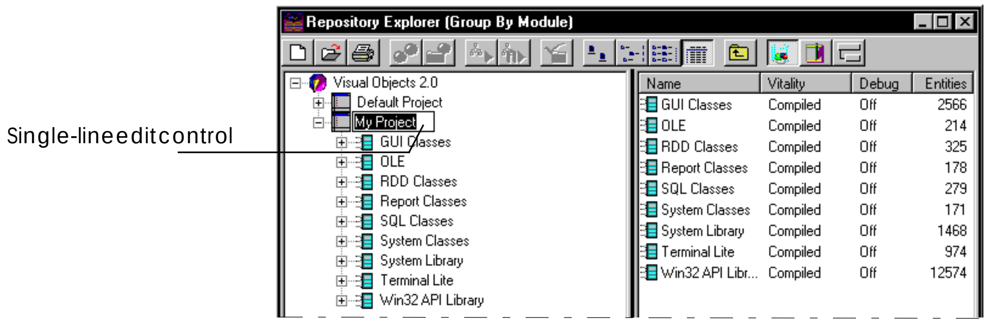
To rename a project:

1. Right-click on the project in the Repository Explorer (for example, My Project).

   The local pop-up menu appears.

2. Choose Rename from the local pop-up menu.

   A single-line edit control now surrounds the specified application, indicating that you are now in edit mode:



- Visual Objects 2.0
  - Default Project
  - My Project
    - GUI Classes
    - OLE
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library
    - Terminal Lite
    - Win32 API Library

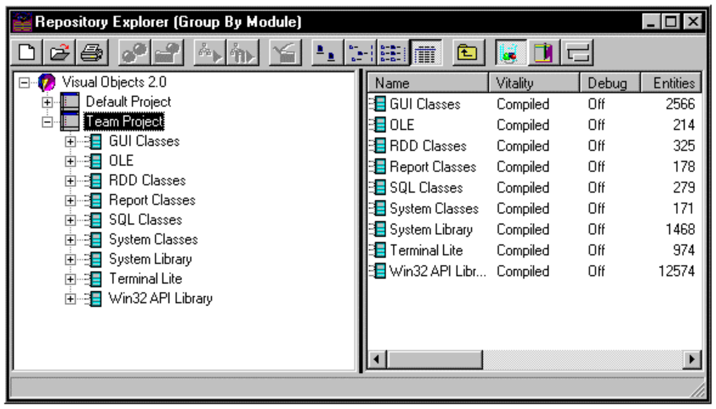
Name	Vitality	Debug	Entities
GUI Classes	Compiled	Off	2566
OLE	Compiled	Off	214
RDD Classes	Compiled	Off	325
Report Classes	Compiled	Off	178
SQL Classes	Compiled	Off	279
System Classes	Compiled	Off	171
System Library	Compiled	Off	1468
Terminal Lite	Compiled	Off	974
Win32 API Libr...	Compiled	Off	12574

Note: The Rename command is available only via a local pop-up menu. However, you can also just single-click on a project to open the single-line edit control.

3. Enter the new name in the single-line edit control (for example, **Team Project**).

4. Click outside the single-line edit control to close the control and save your changes.

   The new project name appears in the Repository Explorer tree:



- Visual Objects 2.0
  - Default Project
  - Team Project
    - GUI Classes
    - OLE
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library
    - Terminal Lite
    - Win32 API Library

Name	Vitality	Debug	Entities
GUI Classes	Compiled	Off	2566
OLE	Compiled	Off	214
RDD Classes	Compiled	Off	325
Report Classes	Compiled	Off	178
SQL Classes	Compiled	Off	279
System Classes	Compiled	Off	171
System Library	Compiled	Off	1468
Terminal Lite	Compiled	Off	974
Win32 API Libr...	Compiled	Off	12574

Creating a New Application

Note: As mentioned earlier in this guide, one of the more noticeable changes in CA-Visual Objects 2.7 is that the Application Wizard has been replaced by the new Application Gallery as the preferred method for creating applications. Therefore, the Use Wizard option is now disabled by default. For detailed information about the Application Gallery (also referred to as the New Application dialog box) and the many new predefined application frameworks that it offers, see the online help.

For this section, you will need to enable the Application Wizard. To do so, select File Setup and check the Use Wizard Checkbox on the System tab, then click OK.

If you followed the tutorial in the *Getting Started* guide, you should have already created the Order Entry application, which is used to demonstrate CA-Visual Objects features. Here we will recap the application creation process, going into more detailed explanations of the intuitive, easy-to-use wizard and its various options.

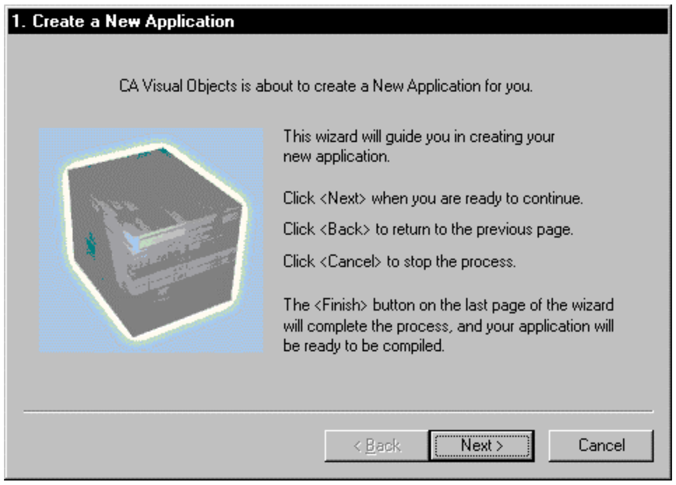
Using the Wizard

To access the Application Wizard in order to define a new application and its properties:

1. From the project level of the Repository Explorer, click the New toolbar button.

   Alternatively, select the New Application command from the File menu.

   The Create a New Application page displays:

   1. Create a New Application

   CA Visual Objects is about to create a New Application for you.

   This wizard will guide you in creating your new application.

   Click <Next> when you are ready to continue.

   Click <Back> to return to the previous page.

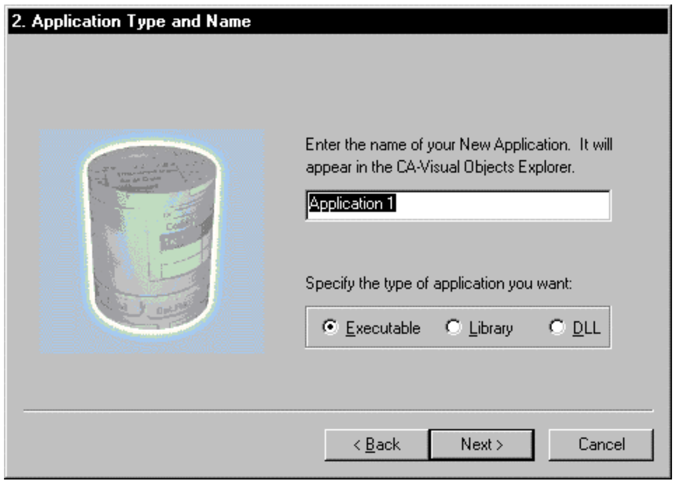
   Click <Cancel> to stop the process.

   The <Finish> button on the last page of the wizard will complete the process, and your application will be ready to be compiled.

   This window explains the Back, Next, Cancel, and Finish push buttons.

2. Click Next.

The Application Type and Name page appears:


2. Application Type and Name

Enter the name of your New Application. It will appear in the CA-Visual Objects Explorer.

Application 1

Specify the type of application you want:

[✓ Executable] Library DLL

3. Enter **Order Entry**, if you have not already created the Order Entry application.

   The default name is Application *n*.

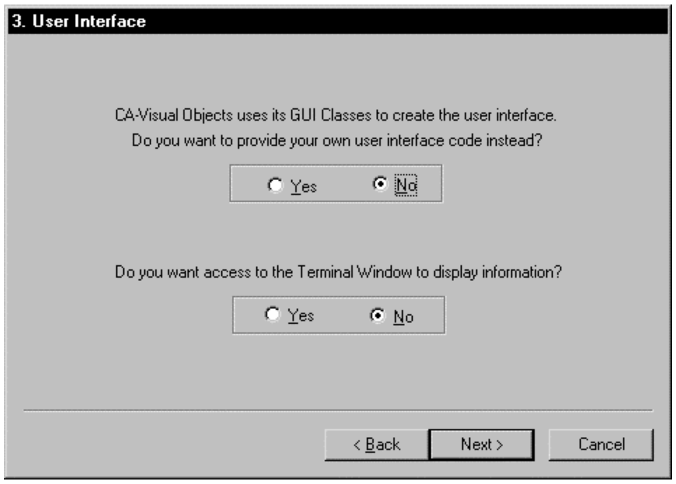
The Application Type and Name page provides the following options:

Application Name	Specify the name of the new application. Application names can be up to 30 characters long, including spaces and special characters (the default is Application <*n*>, where <*n*> is an integer), and they appear in the Repository Explorer's tree structure.
Application Type	Specify whether to create an executable, library, or DLL. This section discusses creating applications—for details about libraries and DLLs, see Creating Libraries and DLLs later in this chapter.

Note: Whereas the Application Wizard limits you to building a standard SDI or MDI application, library, or DLL, the new Application Gallery allows you to build many new types of basic and standard applications, including terminal and console applications, ActiveX controls, Internet applications,
, a n d OLE server applications, as well as ISAPI and Active Serve r
Page (ASP) component DLLs. Examples of these new applications are available from the Samples tab page of the Application Gallery. For detailed information, see the online help.

4. Click Next.

The User Interface page appears:



3. User Interface

CA-Visual Objects uses its GUI Classes to create the user interface.
Do you want to provide your own user interface code instead?

Yes [✓ No]

Do you want access to the Terminal Window to display information?

Yes [✓ No]

The User Interface page provides the following options:

User Interface

CA-Visual Objects offers two types of user interface programming: CA-Visual Objects GUI interface or your own user interface code.

Selecting the default No radio button gives your application access to the CA-Visual Objects GUI Classes library, which contains over a hundred classes that allow you to create the objects required for a full-featured GUI. These include windows, menus, push buttons, scroll bars, list boxes, and so on.

The GUI Classes option also gives you access to CA-Visual Objects self-configuring Standard Application, which generates a basic application, including windows, menus, startup code, and default event and error handling.

For most applications, you will want to include this library, so the No radio button is selected by default. If you plan to use code generated by the IDE's visual editors, you *must* select the No radio button here.

Note: If the Yes radio button is selected, the GUI Classes library is *not* included in the path for the specified application.

Terminal Window

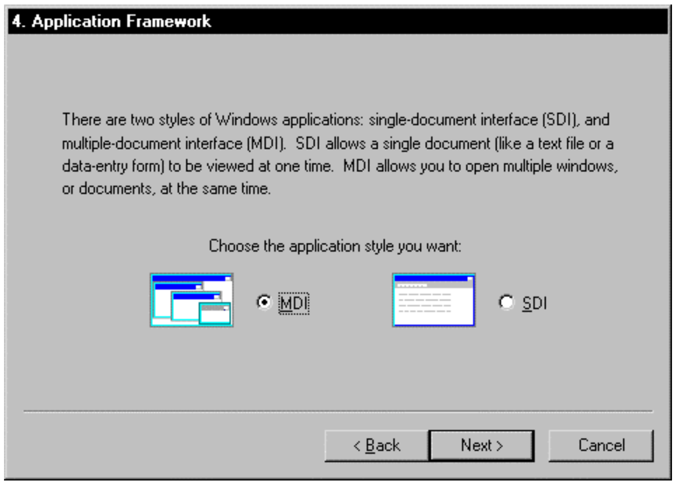
The Terminal Window option, on the other hand, is not initially chosen. This is because the terminal emulation practices used in most Xbase applications (such as @...SAY...GET and most terminal window functions and commands) have no place in event-driven, GUI applications and, therefore, they are no longer supported in CA-Visual Objects.

However, there may be situations where you may want to view or test simple text-based applications. To do so, select the Yes radio button. (Note that this option requires that the Terminal Lite library be included in your application's search path.)

Note: Remember, these are just *initial* settings to help you get your application started. You can change them at any point during the lifetime of the application.

5. Click Next.

   The Application Framework page appears:



4. Application Framework

There are two styles of Windows applications: single-document interface (SDI), and multiple-document interface (MDI). SDI allows a single document (like a text file or a data-entry form) to be viewed at one time. MDI allows you to open multiple windows, or documents, at the same time.

Choose the application style you want:

[✓ MDI] SDI

Note: As mentioned earlier, the new Application Gallery allows you to build many new types of basic and standard SDI and MDI applications, as well as terminal and console applications, ActiveX controls, Internet applications, OLE server applications, ISAPI DLLs, and Active Server Page (ASP) component DLLs.

The Application Framework page provides the following options:

Application Style

Typically, the main window in a GUI application either supports multiple-document interface (MDI) or single-document interface (SDI) applications. (Note that MDI applications are more common.)

To help you get started, the Application Wizard asks you to choose one of these application style options:

- MDI

  An MDI application is structured around the presentation of multiple documents *simultaneously* in many windows. It typically uses a shell window as the main, or “owner,” window. The documents that are opened in the shell window are typically child application windows or data windows.

  Selecting the MDI radio button creates a *multiple windows* application in which the main window is a subclass of the ShellWindow class and the child windows are data windows.

- SDI

  An SDI application, on the other hand, is structured around displaying one document at a time, and it typically uses a top application window as the main window. In this case, the child window is also a child application window or data window.

  Selecting the SDI radio button creates a *single window* application in which the main window is a subclass of the TopAppWindow class and the child window is a data window.

Once you have selected an application style, CA-Visual Objects provides support for your application by automatically creating:

- The appropriate windows and menus based upon its specified application style

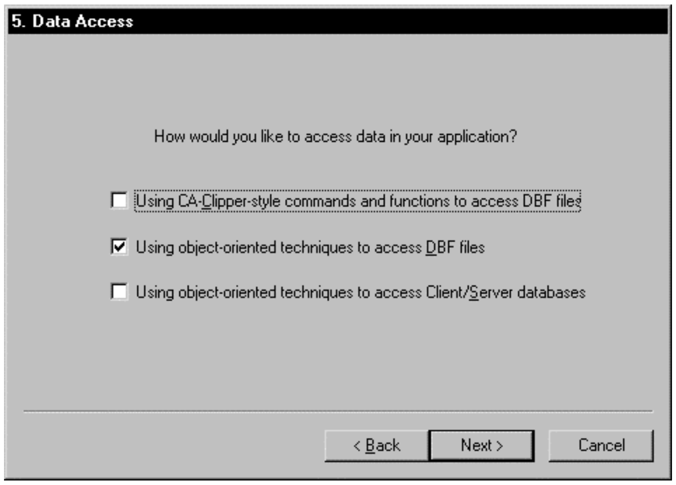
  Note that in both MDI and SDI applications, you automatically receive standard menus (like File and Help),
  a status bar, and a toolbar.

- Startup code for the application

  This code is inserted in the startup module that is created for either type of application. It includes a Start() method that instantiates and displays the application's main window.

6. Click Next.

   The Data Access page appears:



5. Data Access

How would you like to access data in your application?

- [ ] Using CA-Clipper-style commands and functions to access DBF files
- [✓] Using object-oriented techniques to access DBF files
- [ ] Using object-oriented techniques to access Client/Server databases

Note: Using the Application Gallery in CA-Visual Objects 2.7, you can choose either a standard SDI or MDI application framework with support for any one of the following databases: Xbase, SQL, OLE.

The Data Access page provides the following options:

Database

CA-Visual Objects provides three choices for database access. Similar to the user interface choices described above, the choices made here affect what CA-Visual Objects libraries are automatically included in the application's search path and, consequently, what type of database access is available to the application.

If checked, the "Using CA-Clipper-style commands and functions to access DBF files" option provides support for traditional Xbase database operations, like SKIP and EOF(), and includes the RDD Classes library in your application.

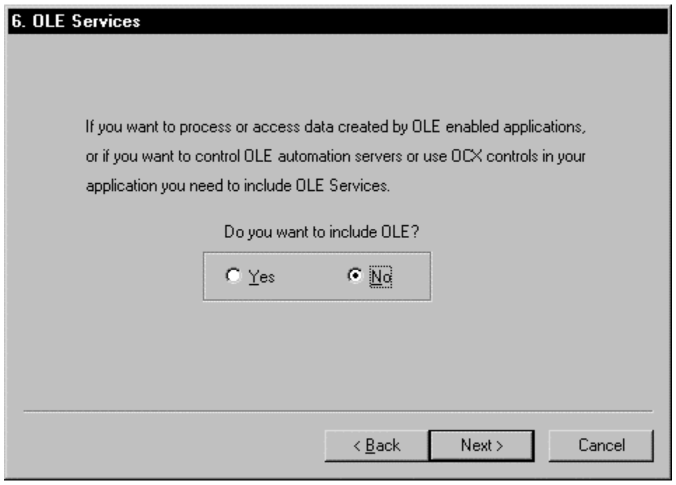
If checked, the "Using object-oriented techniques to access DBF files" option includes the RDD Classes library in your application's search path, giving your application access to an object-oriented interface for Xbase files. This is the default setting.

Similarly, the "Using object-oriented techniques to access Client/Server databases" option, if checked, includes the SQL Classes library in your application's search path, giving your application access to an object-oriented interface for SQL tables.

An application can use any mixture of these options: choose all, just one, or even none of them if you do not plan to create a database application.

7. Click Next.

   The OLE Services page appears:



6. OLE Services

If you want to process or access data created by OLE enabled applications, or if you want to control OLE automation servers or use OCX controls in your application you need to include OLE Services.

Do you want to include OLE?

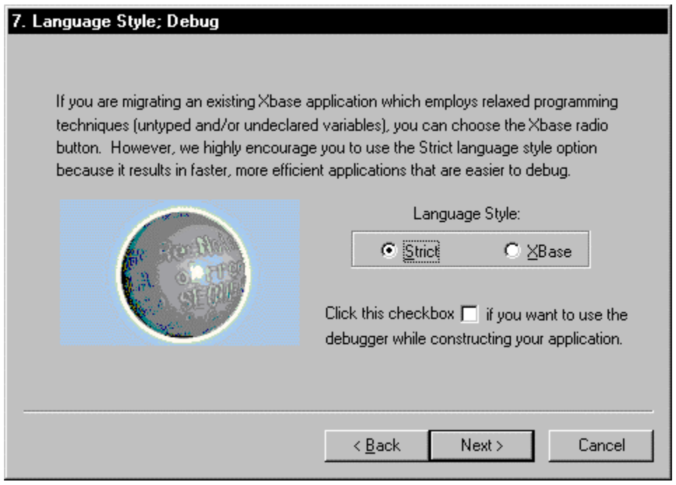
Yes [✓ No]

   Click the Yes radio button if you want to incorporate Object Linking and Embedding (OLE) technology in your application. This option provides you with the capability of linking and embedding OLE objects, as well as embedding OLE custom (OCX) controls in any window. It also gives you access to the Automation Server feature, which creates CA-Visual Objects classes from the objects in third-party applications.

Note: See Linking and Embedding OLE Objects in the “Using the Window Editor” chapter for more information about using OLE objects and OCX controls. Also, refer to the *Programmer’s Guide*.

8. Click Next.

   The Language Style / Debug page appears:



7. Language Style; Debug

If you are migrating an existing Xbase application which employs relaxed programming techniques (untyped and/or undeclared variables), you can choose the Xbase radio button. However, we highly encourage you to use the Strict language style option because it results in faster, more efficient applications that are easier to debug.

Language Style: [✓ Strict] XBase

Click this checkbox ☐ if you want to use the debugger while constructing your application.

The Language Style / Debug page provides the following options.

Language Style

CA-Visual Objects also provides a choice of language styles:

- Strict

  Strict is selected by default—choosing this option promotes the use of some of the more rigid programming techniques supported by CA-Visual Objects. Typically, using these techniques makes code easier to debug, as well as more robust and efficient at runtime. For example, with Strict enabled, using an undeclared variable in a program will raise a compiler error.

- XBase

  However, you may not be familiar with, or are not ready to move to, a more rigid programming style. For example, perhaps you want to prototype an application quickly, or you want to migrate an existing Xbase application. In such cases, it would be best to select the XBase option so that you do not get compiler errors for valid (but not always efficient) programming practices.

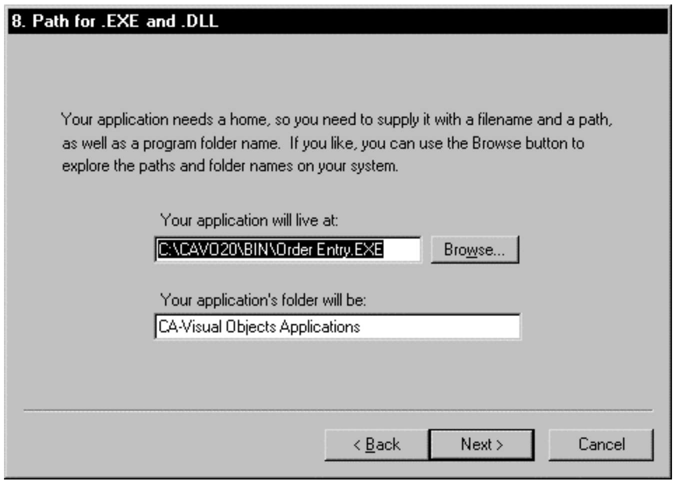
In either case, it is important to note that *neither* option locks you into a particular programming style. Your choice simply influences how the compiler options are *initially* set for the new application. If you change your mind, you can change the application’s compiler options. (See **Setting Compiler Options** in “Working in the Desktop” for complete details.)

Debug

Select the Debug check box so that the application can be debugged when it is compiled.

9. Click Next.

   The Path for .EXE and .DLL page appears:



8. Path for .EXE and .DLL

Your application needs a home, so you need to supply it with a filename and a path, as well as a program folder name. If you like, you can use the Browse button to explore the paths and folder names on your system.

Your application will live at: C:\CAVO20\BIN\Order Entry.EXE

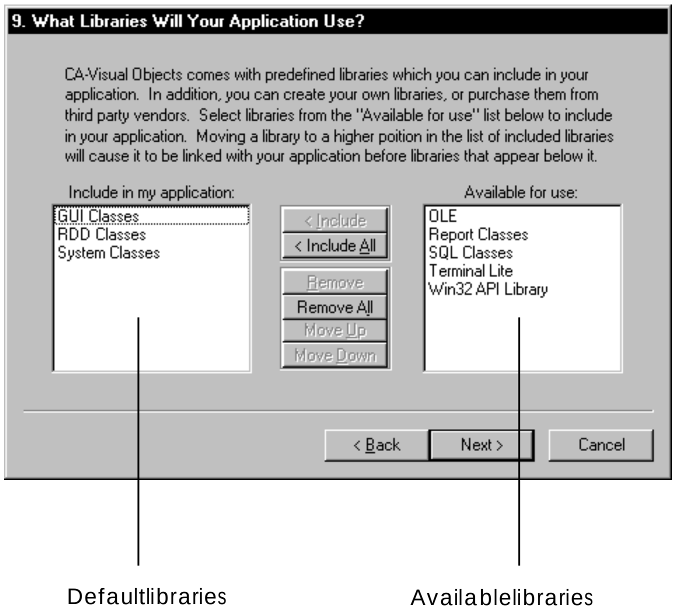
Your application's folder will be: CA-Visual Objects Applications

Note that the application name has defaulted from the wizard's Application Type and Name page to the first edit control on this page. This page offers you a chance to change the application's name and specify a folder for your application.

Note: The path for EXE and DLL files might not be set to C:\CAVO27\BIN (i.e., if you installed your system to another drive or directory.

10. Click Next.

    The Libraries page appears:



9. What Libraries Will Your Application Use?

CA-Visual Objects comes with predefined libraries which you can include in your application. In addition, you can create your own libraries, or purchase them from third party vendors. Select libraries from the "Available for use" list below to include in your application. Moving a library to a higher poition in the list of included libraries will cause it to be linked with your application before libraries that appear below it.

Include in my application: GUI Classes, RDD Classes, System Classes

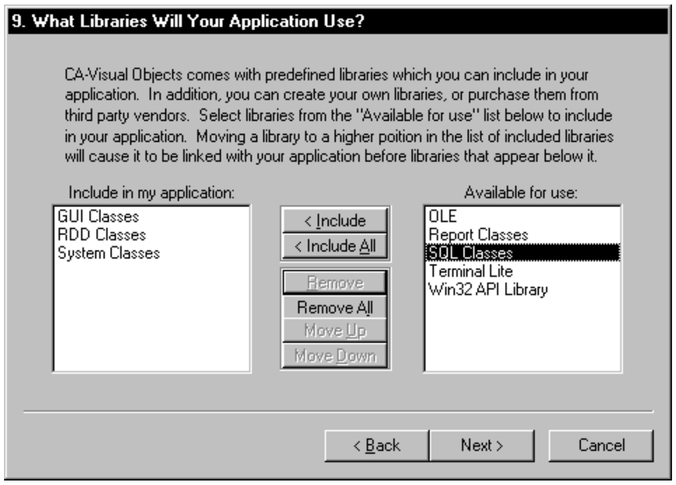
Available for use: OLE, Report Classes, SQL Classes, Terminal Lite, Win32 API Library

Note: In this version of CA-Visual Objects, the Available for Use list box now includes the following new class libraries: Console Classes, Internet, Internet Server API, and OLE Server.

This page provides the opportunity to confirm the libraries that should belong to the application's search path. For example, if you accepted the CA-Visual Objects GUI interface earlier, the following libraries appear in the Include in My Application list box by default: System Classes, GUI Classes, and RDD Classes.

As Order Entry will require the SQL Classes library, you need to add it here if you did not select the "Using object-oriented techniques to access Client/Server databases" option earlier on the Data Access page. To do so:

11. Highlight SQL Classes in the Available for Use list box:


9. What Libraries Will Your Application Use?

CA-Visual Objects comes with predefined libraries which you can include in your application. In addition, you can create your own libraries, or purchase them from third party vendors. Select libraries from the "Available for use" list below to include in your application. Moving a library to a higher poition in the list of included libraries will cause it to be linked with your application before libraries that appear below it.

Include in my application:
GUI Classes
RDD Classes
System Classes

Available for use:
OLE
Report Classes
SQL Classes
Terminal Lite
Win32 API Library

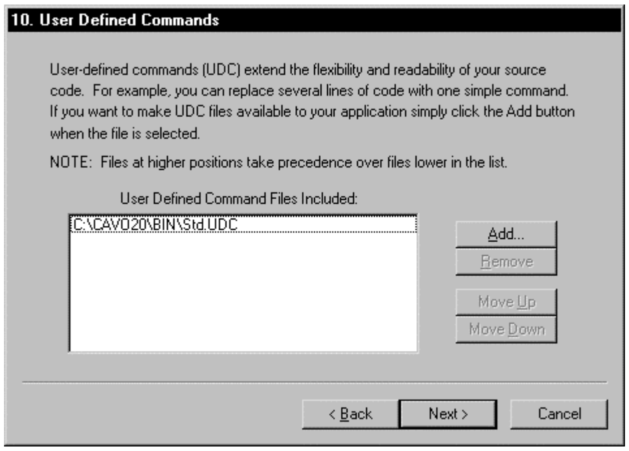
12. Click the Include button.

    SQL Classes is added to the search path.

You are free, of course, to include and/or remove other libraries and/or DLLs in the search path. See Setting the Search Path later in this chapter for complete details on how to do this.

13. Click Next.

The User Defined Commands page appears:


10. User Defined Commands

User-defined commands (UDC) extend the flexibility and readability of your source code. For example, you can replace several lines of code with one simple command. If you want to make UDC files available to your application simply click the Add button when the file is selected.

NOTE: Files at higher positions take precedence over files lower in the list.

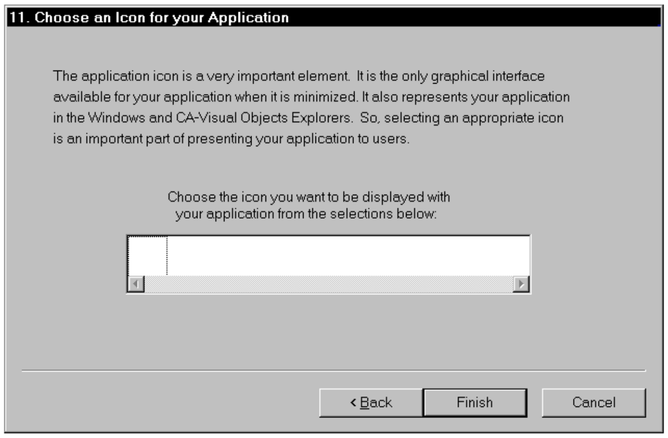
User Defined Command Files Included:

C:\CAVO20\BIN\Std.UDC

Associate one or more user-defined command (.UDC) files with the application. See Associating .UDC Files later in this chapter for details on how to do this.

14. Click Next.

The Application Icon page appears:


11. Choose an Icon for your Application

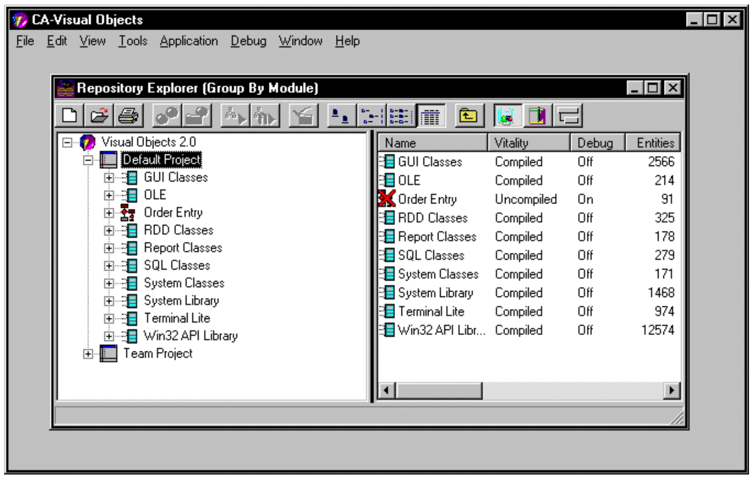
The application icon is a very important element. It is the only graphical interface available for your application when it is minimized. It also represents your application in the Windows and CA-Visual Objects Explorers. So, selecting an appropriate icon is an important part of presenting your application to users.

Choose the icon you want to be displayed with your application from the selections below:

Specify the icon to be used for the application in the folder for the Windows Start menu. See Choosing an Icon later in this chapter for details on how to do this.

15. Click Finish.

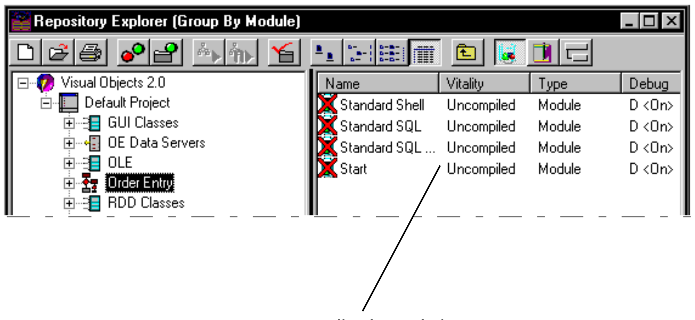
Your new application, Order Entry, is added to the Repository Explorer:



Name	Vitality	Debug	Entities
GUI Classes	Compiled	Off	2566
OLE	Compiled	Off	214
Order Entry	Uncompiled	On	91
RDD Classes	Compiled	Off	325
Report Classes	Compiled	Off	178
SQL Classes	Compiled	Off	279
System Classes	Compiled	Off	171
System Library	Compiled	Off	1468
Terminal Lite	Compiled	Off	974
Win32 API Libr...	Compiled	Off	12574

Note that, in the list view pane, Order Entry has a red X over its application icon indicating that it is uncompiled.

Now double-click on Order Entry. The Repository Explorer displays four predefined modules in the list view pane:

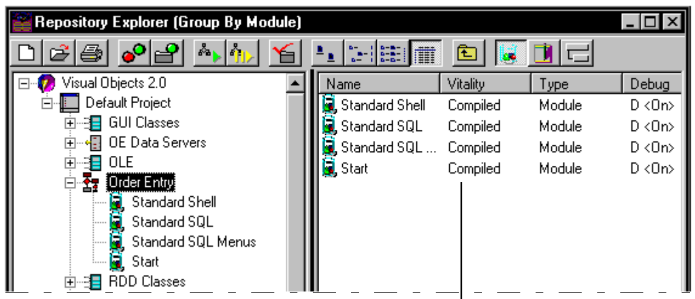


Name	Vitality	Type	Debug
Standard Shell	Uncompiled	Module	D <On>
Standard SQL	Uncompiled	Module	D <On>
Standard SQL ...	Uncompiled	Module	D <On>
Start	Uncompiled	Module	D <On>

Uncompiledmodules

Note that each module icon has both a red X and the "Uncompiled" notation in the Vitality column of the list view pane, indicating that each is uncompiled. Also note that debugging is turned on for each module, since the debugging option was selected earlier.

Now click the Build toolbar button to compile the entire application, as explained earlier in the Saving, Building, and Executing section of "Working in the Desktop." The build is successful, and the Repository Explorer reflects this change in vitality:



Repository Explorer (Group By Module)

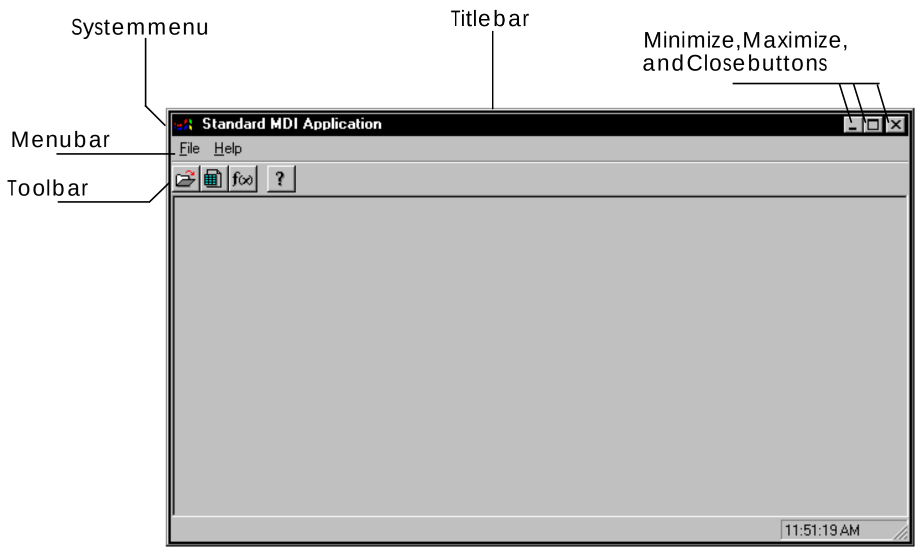
- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
    - OLE
    - Order Entry
      - Standard Shell
      - Standard SQL
      - Standard SQL Menus
      - Start
    - RDD Classes

Name	Vitality	Type	Debug
Standard Shell	Compiled	Module	D <On>
Standard SQL	Compiled	Module	D <On>
Standard SQL ...	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

✗ and "Uncompiled" replaced by "Compiled"

Before modifying any of its properties or adding any window controls to it, click the Execute button to run the application.

A new, functional standard MDI application window appears—the CA-Visual Objects Standard Application, complete with menus, startup code, and default event and error handling:



You have created your first CA-Visual Objects application in just seconds using the Application Wizard!

Modifying Your Application's Properties

When you use the Application Wizard to create an application, you set its *initial* properties. You are free to modify any of these properties, such as changing the standard MDI application's caption, or add any of the following options to your application:

- Include and / or remove additional libraries and / or DLLs in the search path
- Associate one or more user-defined command (.UDC) files with the application.
- Specify an application icon

Modifying the Application's Caption

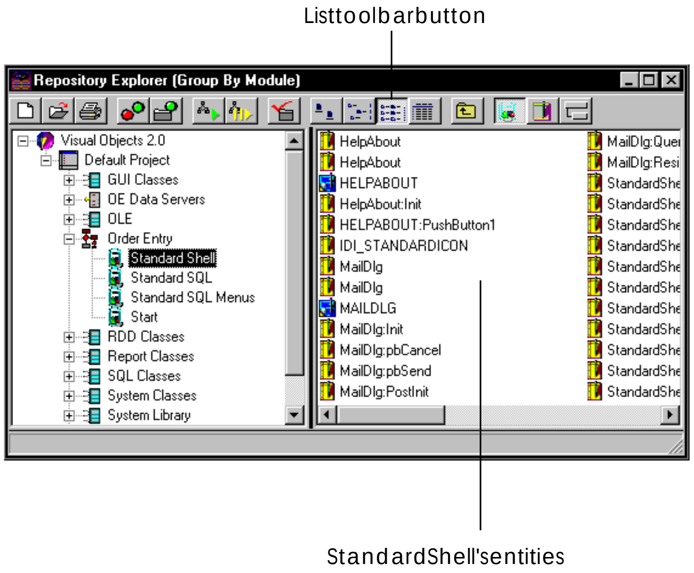
Obviously, you will want to change the caption (or title) for your application from the default caption, "Standard MDI Application," to "Order Entry." To do this:

1. Click the Close button in the upper-right corner to exit the application, if you have not already done so.

   Alternatively, choose the Exit command from the File menu to exit the application.

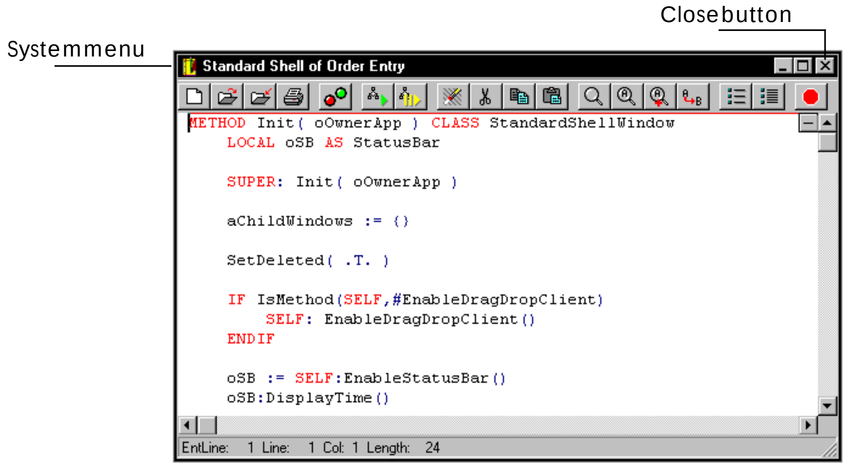
2. Highlight Order Entry's Standard Shell module in the Repository Explorer's tree structure, and then click the List toolbar button.

   All of Standard Shell's entities are displayed in the Repository Explorer's list view pane:



3. Scroll through the list of entities until you find the StandardShellWindow:Init method, and then double-click on it.

The shell window's Init() method is loaded in the Source Code Editor:


Close button
System menu
Standard Shell of Order Entry
EntLine: 1 Line: 1 Col: 1 Length: 24

```
METHOD Init( oOwnerApp ) CLASS StandardShellWindow
    LOCAL oSB AS StatusBar

    SUPER: Init( oOwnerApp )

    aChildWindows := {}

    SetDeleted( .T. )

    IF IsMethod(SELF,#EnableDragDropClient)
        SELF: EnableDragDropClient()
    ENDIF

    oSB := SELF:EnableStatusBar()
    oSB:DisplayTime()
```

4. Move the cursor to the line of code reading:

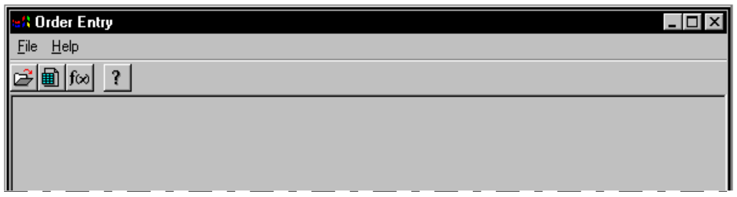
   `SELF:Caption := "Standard MDI Application"`

5. Change it to:

   `SELF:Caption := "Order Entry"`

6. Close the Source Code Editor by either double-clicking on its system menu or clicking on its Close button.

If you now rebuild and run Order Entry, you can see its updated caption:



Setting the Search Path

Sharing Code

Very often, you will want to store commonly used code in a library or a DLL so you can share that code among different applications without having to duplicate and maintain the common code in each application. You will also want to exploit the following CA-Visual Objects libraries: GUI Classes, OLE, RDD Classes, System Classes, Report Classes, SQL Classes, Terminal Lite, and Win32 API.

By default, all applications created in CA-Visual Objects are automatically associated with the System Library and the Win32 API Library. To associate any other library or DLL with an application (such as the GUI Classes library or your own or third-party libraries), you must explicitly add them to the application's *search path.*

Note: For more detailed information about libraries and DLLs, refer to the "Operating Environment" chapter of the *Programmer's Guide.*

Adding Libraries/DLLs

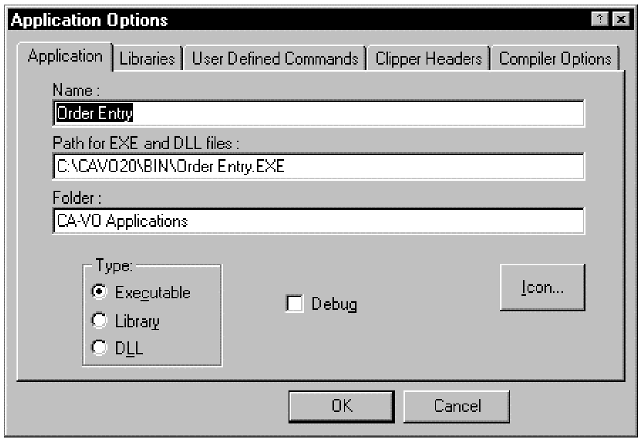
Adding such libraries and DLLs to an application's search path means that CA-Visual Objects can automatically resolve all references to them during compilation. For example, if the compiler encounters a reference to a function or a define that is not contained within any of the application's modules, it starts searching through the libraries and DLLs defined for the application's search path in the order the libraries and DLLs are listed in the Included list box.

Libraries and DLLs can be added to an application's search path using the Properties dialog box, displayed either when you first create the application, or afterwards, using either the Application Properties toolbar button or the Properties command on the local pop-up menu.

Note: In this version of CA-Visual Objects, you can also access the Application Options dialog box directly by using the Alt+Return key combination.

Using the Application Options Dialog Box

In all cases, the Application Options dialog box appears. For example, this dialog box shows the properties that were defined initially for the sample GUI application, Order Entry:



Name : Order Entry

Path for EXE and DLL files : C:\CAVO20\BIN\Order Entry.EXE

Folder : CA-VO Applications

Type: [✓ Executable] Library DLL

☐ Debug

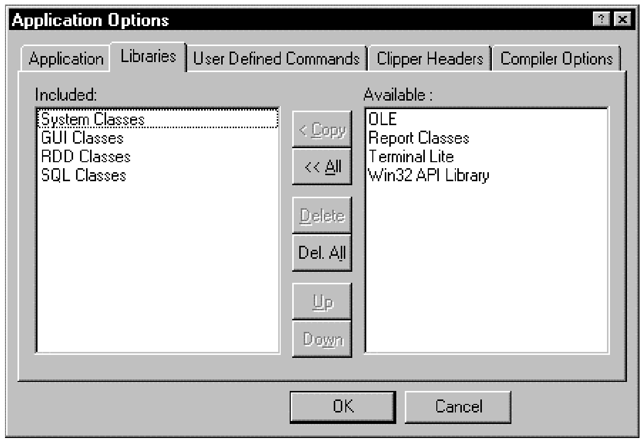
Note: The Application Options dialog box has been updated in CA-Visual Objects 2.7. It now displays the following tabs: Application, Libraries, UDCs, Clipper Headers, Compiler, and OLE Server. For more detailed information, see the online help.

Adding Items to the Search Path

Suppose, for example, you decide that the sample application should use existing OLE code. To add the OLE library to Order Entry's search path:

1. Click on the Libraries tab.

   The Application Options dialog box displays the library options:

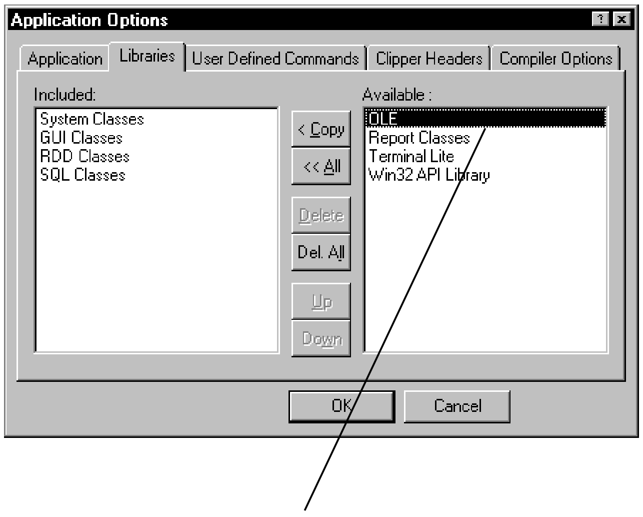


Included:
System Classes
GUI Classes
RDD Classes
SQL Classes

Available :
OLE
Report Classes
Terminal Lite
Win32 API Library

   The Libraries tab page includes two list boxes used for defining an application's search path. The Included list box contains the names of the libraries and DLLs already in the application's file path, as defined earlier using the wizard. The Available list box displays the names of the libraries and DLLs that can be added.

2. Highlight OLE in the Available list box and then click Copy (or just double-click on OLE):

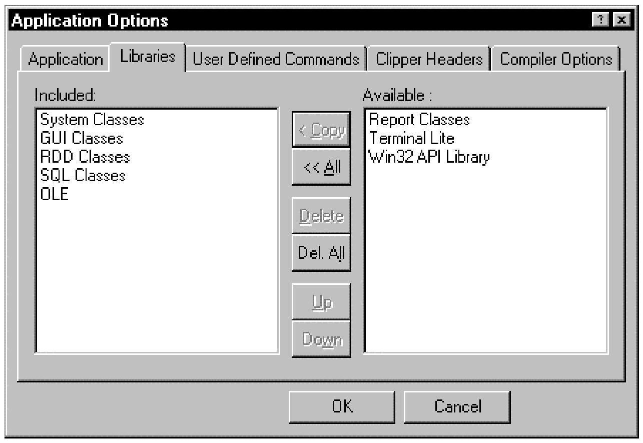


Included: System Classes, GUI Classes, RDD Classes, SQL Classes

Available: OLE, Report Classes, Terminal Lite, Win32 API Library

HighlightOLEandclickCopytoadd theOLElibrarytothesearchpath

The selected item then appears in the Included list box:



Included: System Classes, GUI Classes, RDD Classes, SQL Classes, OLE

Available: Report Classes, Terminal Lite, Win32 API Library

3. Click OK.

   Order Entry's search path is updated.

Removing Items from the Search Path

To remove a library or DLL from an application's search path, double-click on it in the Included list box (or just highlight it and click on the Delete button).

Tip: To add or remove *all* of the available libraries, click on the All or Del All button, respectively.

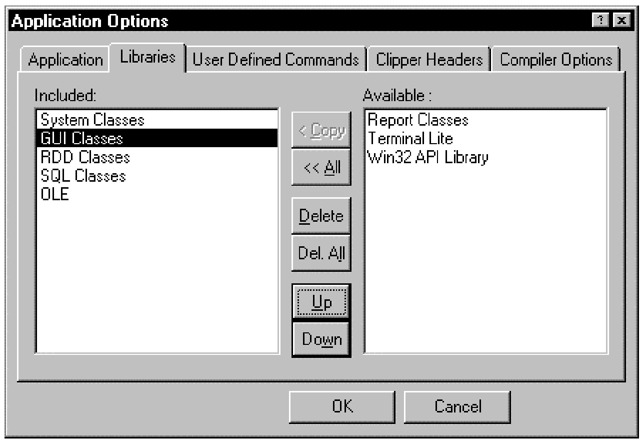
Reordering the Search Path Sequence

By default, the Included list box displays libraries/DLLs in the order in which they were added. The order you see in the Included list box reflects the sequence in which the system searches when resolving references: the library/DLL at the top of the list is searched first, the second is searched second, and so on.

You can rearrange the order in which libraries/DLLs are searched by reordering them in the Included list box. This is useful if, for example, two different libraries in an application's search path contain a function of the same name. You may want to force CA-Visual Objects to use one rather than the other and, therefore, need to ensure that it is searched first.

To move a particular library/DLL up in the list, simply select it in the Included list box and click on the Up button. Similarly, use the Down button to move a library down in the Included list box.

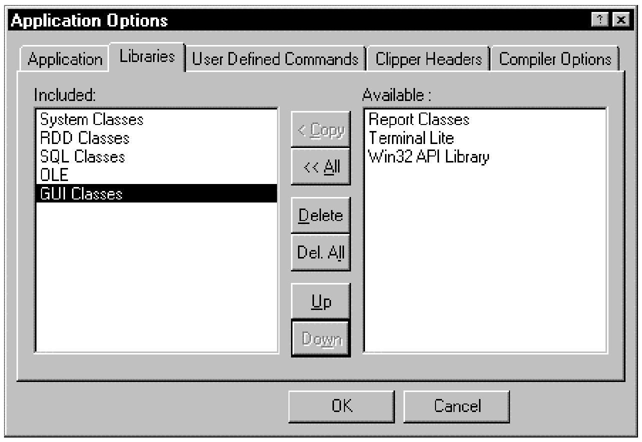
For example, to move the GUI Classes library to the end of the list, highlight it and click three times on the Down button:



Included:
System Classes
GUI Classes
RDD Classes
SQL Classes
OLE

Available :
Report Classes
Terminal Lite
Win32 API Library

The selected item is reordered in the Included list box:



Included:
System Classes
RDD Classes
SQL Classes
OLE
GUI Classes

Available :
Report Classes
Terminal Lite
Win32 API Library

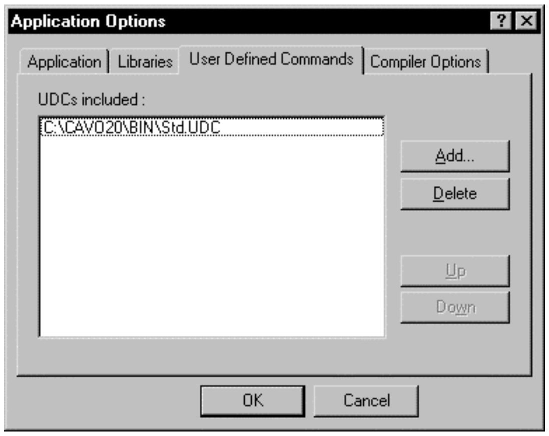
Note: For details on creating libraries and / or DLLs, see Creating Libraries and DLLs later in this chapter.

Associating .UDC Files

You can also associate .UDC files with an application after defining it initially.

To associate .UDC files with an application:

1. Click the UDCs tab in the Application Options dialog box:



2. Choose Add.

   A standard Open dialog box appears.
3. Choose the .UDC file to be included.
4. Select OK.
5. Repeat steps 2–4 for each .UDC file you want to include.

   The maximum is 16.

Tip: Just as you can rearrange the order in which libraries and DLLs are accessed, you can also rearrange the order in which the specified .UDC files are associated with your application using the Up and Down buttons.

Deleting a UDC

To delete a UDC:

1. Highlight the .UDC file in the UDCs Included list box.
2. Choose Delete.

Select OK to close this dialog box when you are finished adding and deleting UDCs.

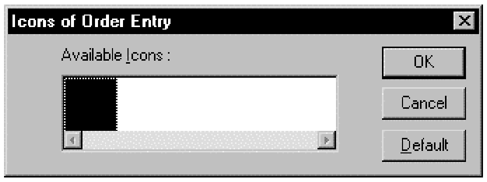
Choosing an Icon

You can also specify an icon for your application, which will be added to the application's folder for the Windows Start menu and to the application's name in the Repository Explorer tree.

To specify an icon for an application:

1. Choose the Application tab in the Application Options dialog box.
2. Click the Explorer Icon push button.

   The Icons dialog box appears:



3. Select an icon from the Available Icons list box.

   The default icon is shown.
4. Click OK.

Tip: You can create your own icons with the Image Editor and save them as entities in any module. When the application is compiled, your icons show up in the Icons dialog box. (See **Chapter 10: Using the Image** Editor for more information.)

Adding Clipper Headers

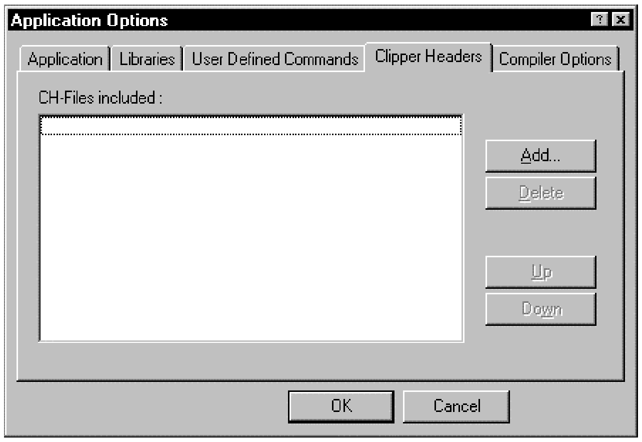
You can optionally include CA-Clipper header (.CH) files in your application, as CA-Visual Objects has a two-level preprocessor. If one or more CA-Clipper-compatible .CH files are included in an application, entities will first be processed by the CA-Clipper preprocessor, which does pure textual replacement. The preprocessed output will then be processed by the CA-Visual Objects 2.7 preprocessor and finally compiled.

If an application uses .CH files, all of the CA-Clipper directives, like #xtranslate and #include, can also be used within an entity's source code. Note that all conditional compilation (#ifdef, #ifndef, and so on) will be done by the CA-Clipper preprocessor and is not seen by the CA-Visual Objects preprocessor. The CA-Clipper preprocessor does not know about defines in the repository: it uses #defines in the header files.

Note: If an application does not use .CH files, the CA-Clipper preprocessor will not run during compilation. See the *Programmer's Guide* for more detailed information about the CA-Clipper preprocessor.

To include .CH files in your application:

1. Choose the Clipper Headers tab in the Application Options dialog box:



2. Choose Add.

   A standard Open dialog box appears.
3. Choose the .CH file to be included.
4. Select OK.
5. Repeat steps 2–4 for each .CH file you want to include.

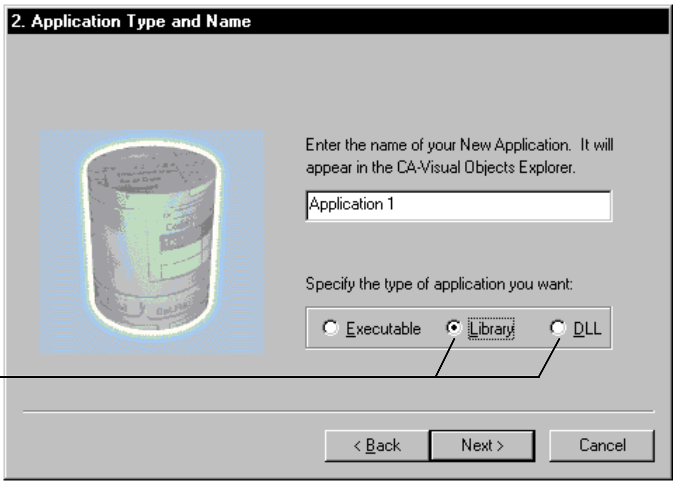
   The maximum is 16.

Creating Libraries and DLLs

In addition to applications, CA-Visual Objects also allows you to create libraries and DLLs. Either type allows you to store commonly used code in a single location.

Note: See the "Operating Environment" chapter of the *Programmer's Guide* for a more detailed discussion of the benefits and the circumstances under which you should use a DLL rather than a library.

To create either a library or DLL from “scratch,” select the appropriate radio button in the Application Type radio button group on the Application Wizard’s Application Type and Name page:


2. Application Type and Name
Executable
Library
DLL
< Back
Next >
Cancel
Select either button to create a library or DLL

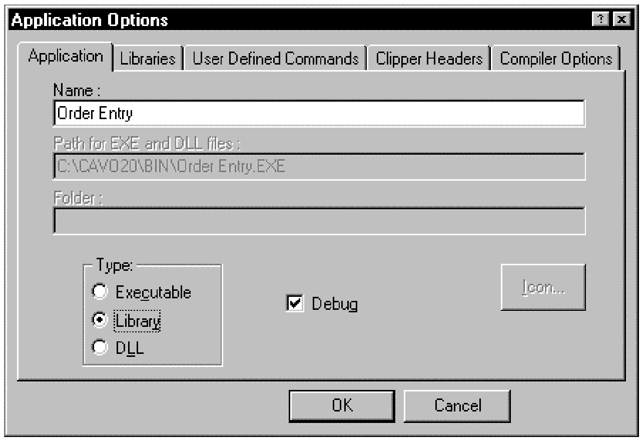
Enter the name of your New Application. It will appear in the CA-Visual Objects Explorer.

Application 1

Specify the type of application you want:

( ) Executable [✓ Library] ( ) DLL

You can also change the type of an existing application to either Library or a DLL using the Application Options dialog box. For example, select the Library radio button in the Type radio button group to change the Order Entry sample application to a library, like so:



Name : Order Entry

Path for EXE and DLL files : C:\CAVO20\BIN\Order Entry.EXE

Folder :

Type: ( ) Executable [✓ Library] ( ) DLL

[✓ Debug]

Note: As mentioned earlier, the Application Options dialog box has been updated in CA-Visual Objects 2.7. The Application tab page now displays four new options: Description, .MDF File, Create VOM File, and Executable for Debug Session. Additionally, two others have new values. Specifically, the Type radio button group box offers the following choices: Windows Application, Console Application, Library, and DLL or ActiveX. Moreover, system variables, such as %ExecutableDir%, may be entered in the Path for EXE and DLL Files edit control. For more detailed information, see the online help.

Note, however, that some options in this dialog box, such as Path for EXE and DLL Files, are not available for libraries.

Manipulating Applications

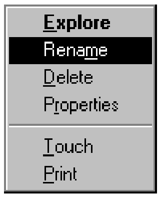
The Repository Explorer allows you to manipulate your applications and libraries—for example, you can rename an application, move an application to a different project, delete an application, or print the contents of an application.

Renaming Applications

If you want to rename an application:

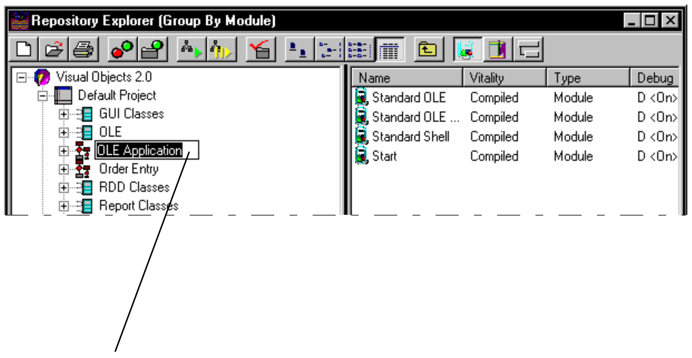
1. Right-click on the application in the Repository Explorer (for example, OLE Application).

   The local pop-up menu appears:

2. Choose Rename from the local pop-up menu.

   A single-line edit control now surrounds the specified application, indicating that you are now in edit mode:

Name	Vitality	Type	Debug
Standard OLE	Compiled	Module	D <On>
Standard OLE ...	Compiled	Module	D <On>
Standard Shell	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

   Single-line edit control

   Note: The Rename command is available only via a local pop-up menu. However, you can also just single-click on an application to open the single-line edit control.

3. Enter the new name in the single-line edit control.
4. Press Enter.

   The new application name appears in the Repository Explorer tree.

Note: You cannot rename any CA-Visual Objects system files.

Moving Applications

To move an application from one project to another:

1. Click on the application in the list view pane.
2. Choose the Cut command from the Edit menu.
3. Highlight the target project in the tree view pane, and then choose the Paste command from the Edit menu.

The specified application is removed from the original project, and added to the target project.

Copying Applications

To copy an application from one project to another:

1. Click on the application in the list view pane.
2. Choose the Copy command from the Edit menu.
3. Highlight the target project in the tree view pane, and then choose the Paste command from the Edit menu.

The specified application is copied to the target project.

Deleting Applications

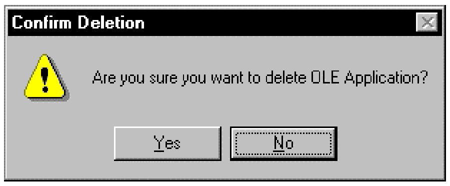
If you want to delete an application:

1. Right-click on the application in the Repository Explorer (for example, OLE Application).

   The local pop-up menu appears.

   (Alternatively, select the Delete command from the Edit menu.)
2. Choose Delete from the local pop-up menu.

   The Confirm Deletion dialog box appears:

   Are you sure you want to delete OLE Application?
3. Click Yes.

   The application is deleted from the repository.

Note: See your online help system for information about the other local pop-up menu commands.

Importing and Exporting Applications and Libraries

CA-Visual Objects allows you to import and export entire applications; they are stored in *application export files* (.AEF).

Tip: The export option can be used to create a backup copy of the application.

For details, see **Chapter 12: Importing and Exporting** Applications in this guide.

Creating Modules

From within the Repository Explorer you can:

- Create, open, copy, rename, delete, print, import and export, and debug modules
- Access other editors, the Debugger, and the Error Browser

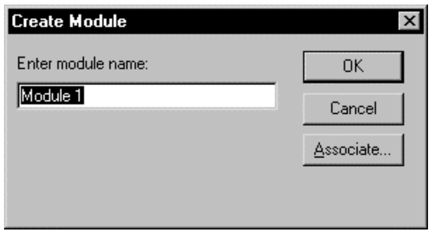
Creating an Internal Module

When creating modules in CA-Visual Objects, you typically want to create them as *internal* modules, so that they are stored and maintained by the repository. To create an internal module:

1. With the current application highlighted, click on the New toolbar button.

   Alternatively, choose the New Module command from the File menu.

   The Create Module dialog box appears:



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

Module names can be as long as 30 characters, including spaces and special characters. The default is Module <*n*>, where <*n*> is an integer.

3. Choose OK.

Creating an External Module

You can also create a module that is *external*—that is, it is not stored in the repository, but is actually a link to an external source file. The Associate button in the Create Module dialog box allows you to associate a new module with an external source file on disk.

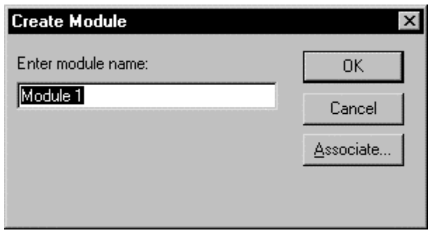
You work with an external file like any other module—viewing, editing, and so on. External modules are imported into the repository when the module is created, and CA-Visual Objects will keep the source code in the repository in synchronization with the source code in the external file. This is accomplished by reading the file from disk whenever source code is needed by the repository and writing all changes made in the Source Code Editor to the external file when you save changes.

Creating an External Module

To create an external module:

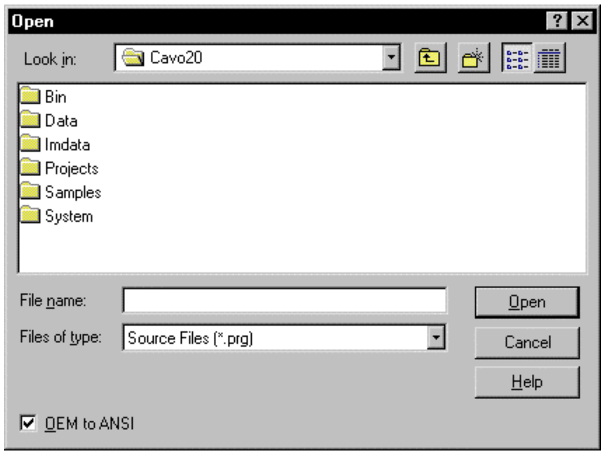
1. With the current application highlighted, click on the New toolbar button.

   The Create Module dialog box appears:

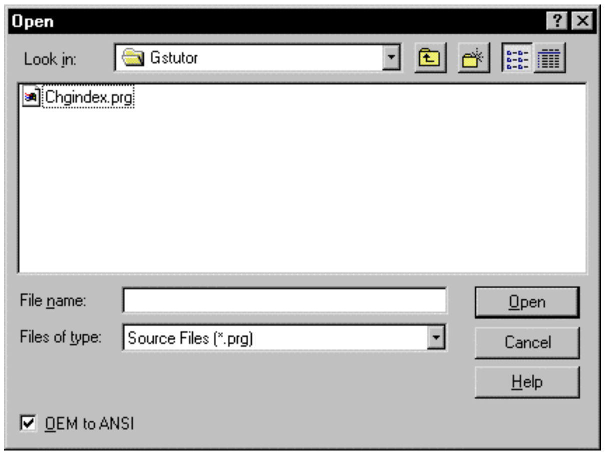


2. Choose the Associate button.

   A standard Open dialog box appears:

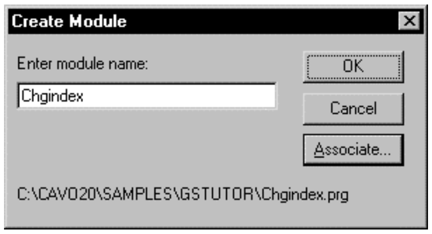


3. Select the desired text-based source file (for example, CHGINDEX.PRG in the GSTUTOR folder):



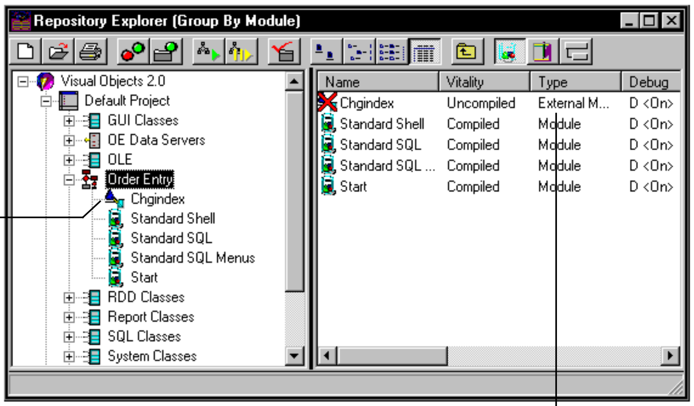
4. Choose Open.

   You are returned to the Create Module dialog box, and the Enter Module Name edit control is filled in with the source file's file name:



5. Optionally, type a new name in the Enter Module Name edit control.
6. Choose OK.

CA-Visual Objects creates a module that can be opened, renamed, and manipulated like any other. However, it associates a different graphic to the module button to indicate that it is an external module:



Name	Vitality	Type	Debug
Chgindex	Uncompiled	External M...	D <On>
Standard Shell	Compiled	Module	D <On>
Standard SQL	Compiled	Module	D <On>
Standard SQL ...	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

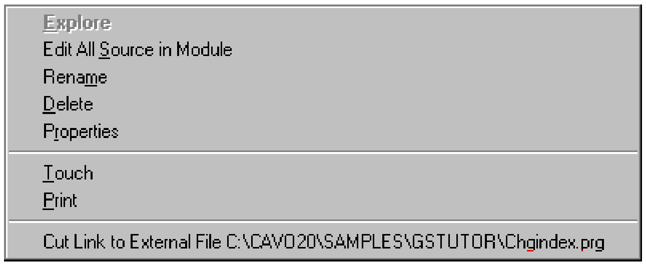
Removing the Link to an External File

Later you can optionally save the external module as an internal module by removing its link to the external text-based source file.

To remove the link between a module and an external file:

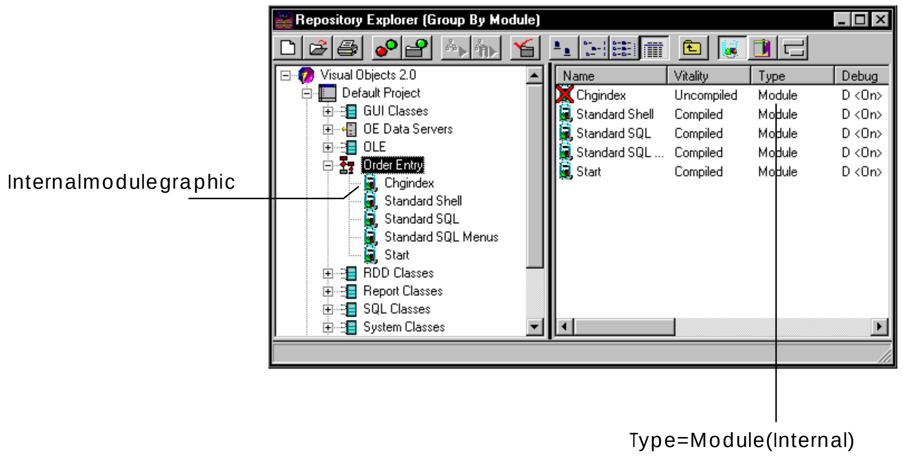
1. Right-click on the selected module.

   A local pop-up menu appears:


Explore
Edit All Source in Module
Rename
Delete
Properties
Touch
Print
Cut Link to External File C:\CAVO20\SAMPLES\GSTUTOR\Chgindex.prg

2. Select the Cut Link to External File command.

The Repository Explorer changes accordingly:



Name	Vitality	Type	Debug
Chgindex	Uncompiled	Module	D <On>
Standard Shell	Compiled	Module	D <On>
Standard SQL	Compiled	Module	D <On>
Standard SQL ...	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

For more detailed information about using external text-based source files, see the **Chapter 12: Importing and Exporting** Applications chapter.

Setting Module Properties

You can add a description for the module, as well as set debugging and compiler options at the module level in CA-Visual Objects.

Description Property

For example, to add a description that will appear in the Description column of the Repository Explorer's list view:

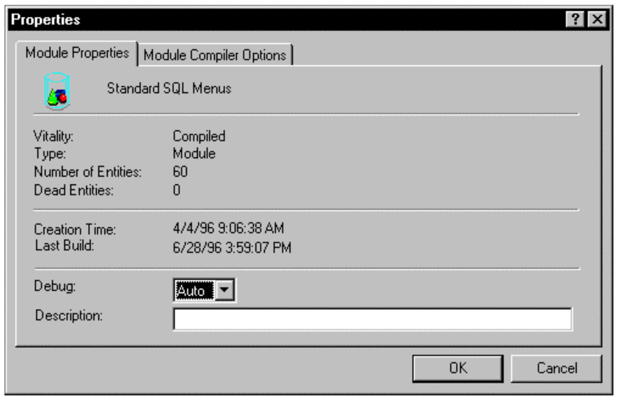
1. Right-click on the specified module.

   The local pop-up menu shown earlier appears.

   Note: In this version of CA-Visual Objects, you can also access the Properties (Module) dialog box directly by using the Alt+Return key combination.

2. Select the Properties command from the local pop-up menu.

The Properties (Module) dialog box appears:



Standard SQL Menus

Vitality:	Compiled
Type:	Module
Number of Entities:	60
Dead Entities:	0
Creation Time:	4/4/96 9:06:38 AM
Last Build:	6/28/96 3:59:07 PM
Debug:	Auto
Description:	

Notice that the Module Properties tab reflects the module's history, including vitality, number of entities, creation date, and so on.

Note: This dialog box has been changed slightly in CA-Visual Objects 2.7. The Debug drop-down list box has been replaced by three radio buttons: Auto, On, and Off. Additionally, it indicates whether or not source control is available and, if so, the source control status of the specified module. For detailed information about the new Source Code Control Interface, refer to the online help.

3. Enter a description in the Description edit control.

Debug Option

4. Optionally, reset the debugging option for this module.

   Valid choices are: <Auto>, On, and Off. See **Setting Debugging Options** in the “Debugging Your Applications” chapter for more information.

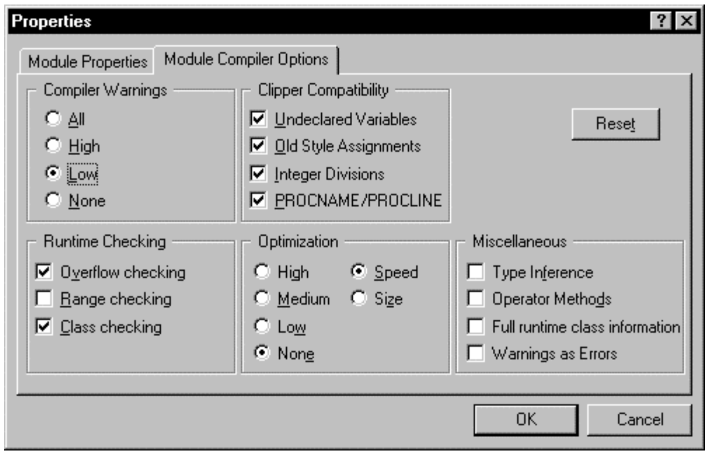
5. Click OK.

Compiler Options

Just as you can override the system-wide, default compiler options at the application level, you can do the same at the module level.

To set module-specific compiler options:

1. Click the Module Compiler Options tab in the Properties dialog box:



Properties

Module Properties | Module Compiler Options

Compiler Warnings: All High [✓ Low] None

Clipper Compatibility: [✓ Undeclared Variables] [✓ Old Style Assignments] [✓ Integer Divisions] [✓ PROCNAME/PROCLINE]

Runtime Checking: [✓ Overflow checking] Range checking [✓ Class checking]

Optimization: High Medium Low [✓ None] — [✓ Speed] Size

Miscellaneous: Type Inference; Operator Methods; Full runtime class information; Warnings as Errors

2. Reset any of the compiler options.

   Refer to **Setting Compiler Options** in “Working in the Desktop” for complete descriptions of these options.

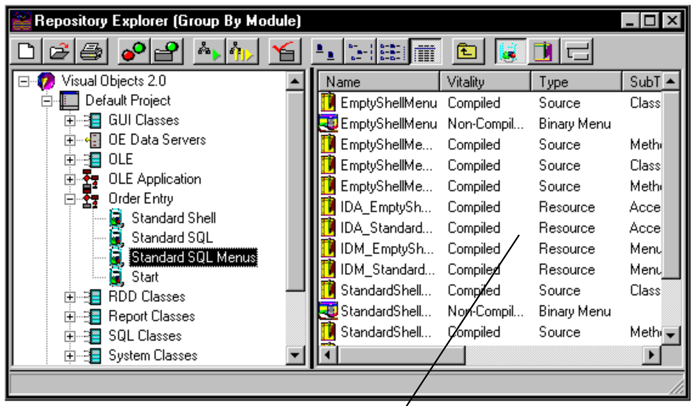
3. Click OK.

Editing Modules

Opening Modules

In CA-Visual Objects, *opening* a module simply means to display its entities in the Repository Explorer so that you can view or edit its contents.

To open an existing module, simply click on the module and all of the entities in that module are displayed. For example, if you choose the Standard SQL Menus module of the Order Entry application, the result is:



Name	Vitality	Type	SubT
EmptyShellMenu	Compiled	Source	Class
EmptyShellMenu	Non-Compil...	Binary Menu	
EmptyShellMe...	Compiled	Source	Meth
EmptyShellMe...	Compiled	Source	Class
EmptyShellMe...	Compiled	Source	Meth
IDA_EmptySh...	Compiled	Resource	Acce
IDA_Standard...	Compiled	Resource	Acce
IDM_EmptySh...	Compiled	Resource	Menu
IDM_Standard...	Compiled	Resource	Menu
StandardShell...	Compiled	Source	Class
StandardShell...	Non-Compil...	Binary Menu	
StandardShell...	Compiled	Source	Meth

StandardSQLMenus'entities

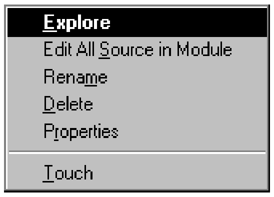
Once a module is opened, you can access a module's source code and edit it—either at the module level or the entity level.

Editing an Entire Module

To access *all* of the module's source code (not just the code for an individual entity within that module):

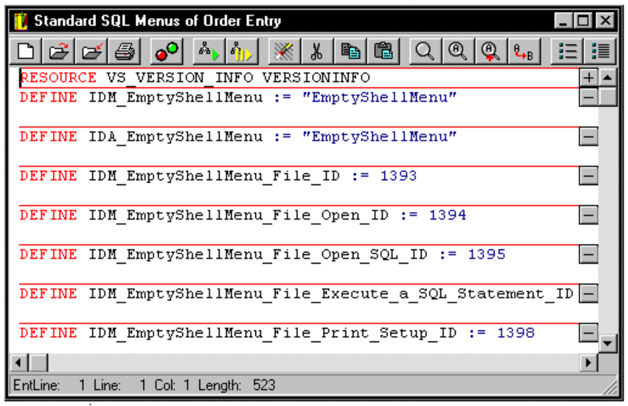
1. Right-click on the specified module (for example, Standard SQL Menus in Order Entry) in the Repository Explorer's tree view pane.

   A local pop-up menu appears:

2. Choose the Edit All Source in Module menu command.

The Source Code Editor appears:



```
RESOURCE VS_VERSION_INFO VERSIONINFO
DEFINE IDM_EmptyShellMenu := "EmptyShellMenu"

DEFINE IDA_EmptyShellMenu := "EmptyShellMenu"

DEFINE IDM_EmptyShellMenu_File_ID := 1393

DEFINE IDM_EmptyShellMenu_File_Open_ID := 1394

DEFINE IDM_EmptyShellMenu_File_Open_SQL_ID := 1395

DEFINE IDM_EmptyShellMenu_File_Execute_a_SQL_Statement_ID

DEFINE IDM_EmptyShellMenu_File_Print_Setup_ID := 1398
```

EntLine: 1 Line: 1 Col: 1 Length: 523

You can now edit all of the module's source code. For more information about editing source code, see **Chapter 6: Using the Source** Code Editor.

Manipulating Modules

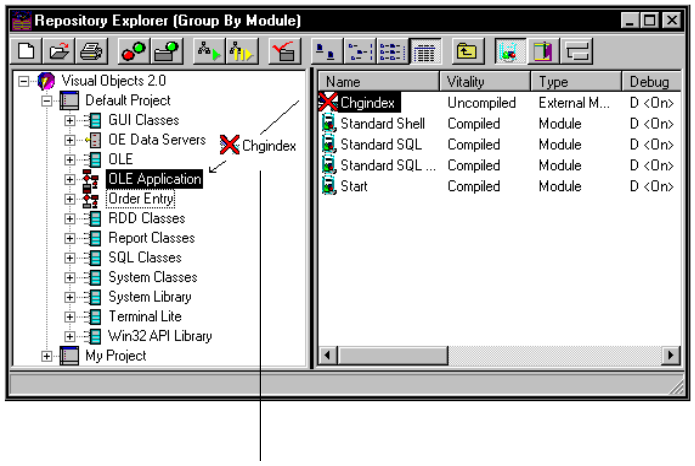
The Repository Explorer also allows you to manipulate your modules. For example, you can move and copy modules from one application to another using the standard Windows drag-and-drop techniques. You can also rename or delete modules.

Moving Modules

To move a module from one application to another:

1. With the application highlighted in the Repository Explorer's tree view pane, click on the specified module in the list view pane (for example, the Chngindex module in Order Entry).

2. Hold the left mouse button down and drag the module to the desired application (for example, OLE Application) in the tree view pane, as shown:



Repository Explorer (Group By Module)

```
Visual Objects 2.0
  Default Project
    GUI Classes
    OE Data Servers
    OLE
    OLE Application
    Order Entry
    RDD Classes
    Report Classes
    SQL Classes
    System Classes
    System Library
    Terminal Lite
    Win32 API Library
  My Project
```

Name	Vitality	Type	Debug
Chgindex	Uncompiled	External M...	D <On>
Standard Shell	Compiled	Module	D <On>
Standard SQL	Compiled	Module	D <On>
Standard SQL ...	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

Drag-and-dropmoduleontoapplication

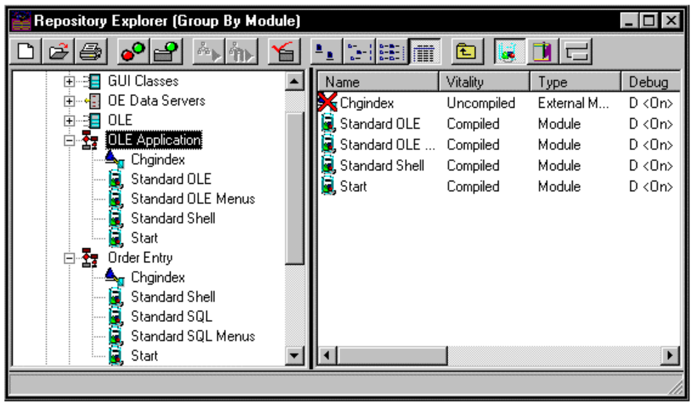
Tip: Alternatively, use the Cut and Paste commands on the Edit menu.

Copying Modules

To copy a module from one application to another:

1. Click on the module in the list view pane (for example, the Chngindex module in Order Entry).
2. Hold down the Ctrl key and drop-and-drag the module to the desired application (for example, OLE Application).

   Now both applications contain copies of the Chngindex module:



Repository Explorer (Group By Module)

```
    GUI Classes
    OE Data Servers
    OLE
    OLE Application
      Chgindex
      Standard OLE
      Standard OLE Menus
      Standard Shell
      Start
    Order Entry
      Chgindex
      Standard Shell
      Standard SQL
      Standard SQL Menus
      Start
```

Name	Vitality	Type	Debug
Chgindex	Uncompiled	External M...	D <On>
Standard OLE	Compiled	Module	D <On>
Standard OLE ...	Compiled	Module	D <On>
Standard Shell	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

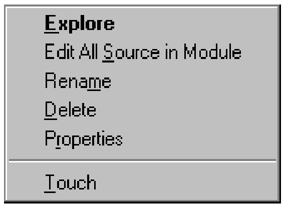
Tip: Alternatively, use the Copy and Paste commands on the Edit menu.

Renaming Modules

If you want to rename a module:

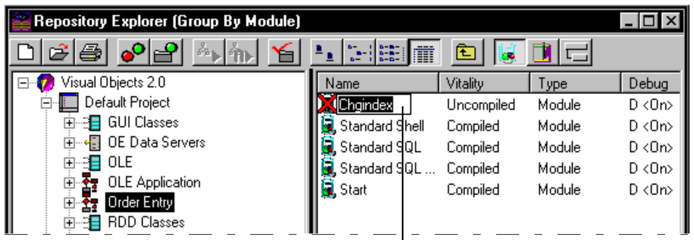
1. Right-click on the module in the Repository Explorer (for example, Chngindex in Order Entry).

   A local pop-up menu appears:



2. Choose Rename from the local pop-up menu.

   A single-line edit control now surrounds the module:



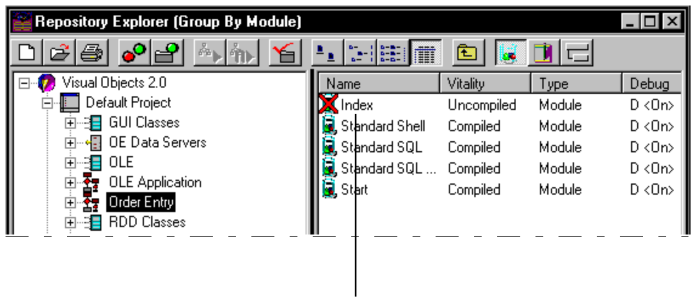
Name	Vitality	Type	Debug
Chngindex	Uncompiled	Module	D <On>
Standard Shell	Compiled	Module	D <On>
Standard SQL	Compiled	Module	D <On>
Standard SQL ...	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

Single-line edit control

Note: The Rename command is available only via a local pop-up menu. However, you can also just single-click on an item to open the single-line edit control.

3. Enter the new name in the edit control (for example, **Index**).
4. Press Enter.

The name change is reflected in the Repository Explorer:



Visual Objects 2.0
Default Project
GUI Classes
OE Data Servers
OLE
OLE Application
Order Entry
RDD Classes

Name	Vitality	Type	Debug
Index	Uncompiled	Module	D <On>
Standard Shell	Compiled	Module	D <On>
Standard SQL	Compiled	Module	D <On>
Standard SQL ...	Compiled	Module	D <On>
Start	Compiled	Module	D <On>

Deleting Modules

If you want to delete a module from the repository:

1. Right-click on the module in the Repository Explorer.

   A local pop-up menu appears.

   (Alternatively, select the Delete command from the Edit menu or use the Del key.)

2. Choose Delete from the local pop-up menu.

   The Confirm Deletion dialog box appears.

3. Click Yes.

Note: See your online help system for more information about the local pop-up menu commands for manipulating modules.

Importing and Exporting Modules

CA-Visual Objects allows you to import and export entire modules; they are stored in *module export files* (.MEF).

Tip: The export option can be used to create a backup copy of the module.

For details, see **Chapter 12: Importing and Exporting** Applications in this guide.

Creating Entities

As stated earlier, 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, a function, procedure, global, constant, or form).

To create a new entity, click on the New toolbar button at the module level in the Repository Explorer 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 binary menu entity using the Menu Editor.

CA-Visual Objects places the new entity in the *current* module. Therefore, you may first want to switch to another module or create a new one before clicking the New Entity button. (See **Accessing the Editors** in "Working in the Desktop" for details.)

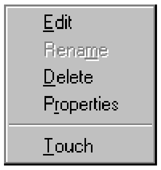
Editing Entities

Opening Entities

Opening an entity in CA-Visual Objects simply means to display the entity in its associated editor.

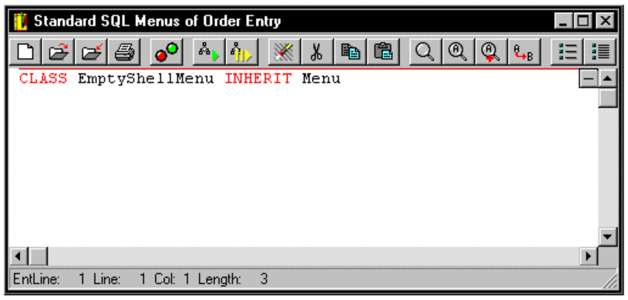
To open an existing entity:

1. Double-click on that entity or right-click on it to open a local pop-up menu:



2. Choose the Edit command.

   In all cases, the appropriate editor is invoked. For example, the Source Code Editor appears if the EmptyShellMenu source entity in the Standard SQL Menus module of Order Entry is selected:



```
CLASS EmptyShellMenu INHERIT Menu
```

EntLine: 1 Line: 1 Col: 1 Length: 3

You can now edit the source code for the EmptyShellMenu source entity.

Note: If you double-click on a source code entity that is associated with an external module, the source code for the entire external file is loaded in the Source Code Editor—not just the source code for the current entity.

Copying Entities

To copy an entity from one module to another:

1. Click on the entity in the list view pane.
2. Hold down the Ctrl key and drop-and-drag the module to the desired module.

   Now both modules contain copies of the specified entity.

Deleting Entities

If you choose the Delete command from the local pop-up menu or use the Del key, the system automatically deletes the entity from the module.

Setting Entity Properties

You can add a description for the entity, as well as set debugging and compiler options at the entity level in CA-Visual Objects.

Description Property

For example, to add a description that will appear in the Description column of the Repository Explorer's list view:

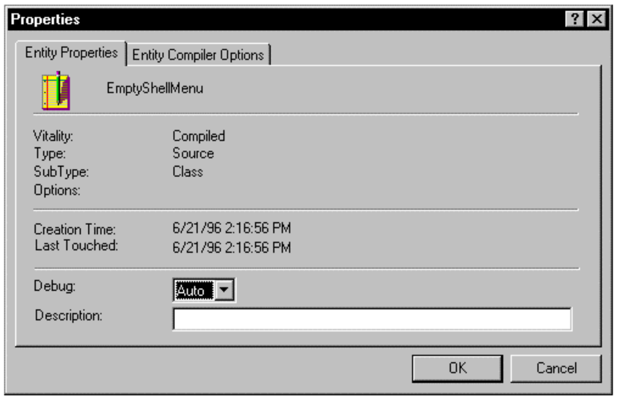
1. Right-click on the specified entity (for example, the EmptyShellMenu source entity in the Standard SQL Menus module).

   The local pop-up menu shown earlier appears.

   Note: In this version of CA-Visual Objects, you can also access the Properties (Entity) dialog box directly by using the Alt+Return key combination.

2. Select the Properties command from the local pop-up menu.

   The Properties (Entity) dialog box appears:



EmptyShellMenu

Vitality: Compiled
Type: Source
SubType: Class
Options:

Creation Time: 6/21/96 2:16:56 PM
Last Touched: 6/21/96 2:16:56 PM

Debug: Auto
Description:

   Notice that the Entity Properties tab reflects the entity's history, including vitality, type and subtype, creation date, and so on.

Note: This dialog box has been changed slightly in CA-Visual Objects 2.7. The Debug drop-down list box has been replaced by three radio buttons: Auto, On, and Off. Additionally, it indicates whether or not source control is available and, if so, the source control status of the specified entity. For detailed information about the new Source Code Control Interface, refer to the online help.

3. Enter a description in the Description edit control.

Debug Option

4. Optionally, reset the debugging option for this entity.

   Valid choices are: <Auto>, On, and Off. See **Setting Debugging Options** in the "Debugging Your Applications" chapter for more information.

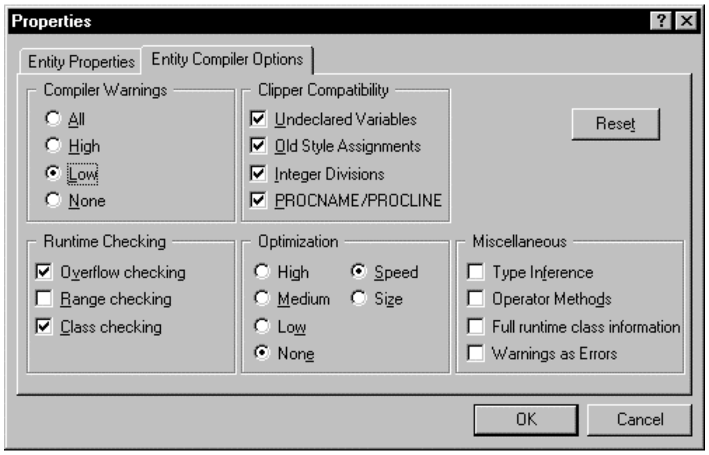
5. Click OK.

Compiler Options

Just as you can override the system-wide, default compiler options at the application or module level, you can do the same at the entity level.

To set entity-specific compiler options:

1. Click the Entity Compiler Options tab in the Properties dialog box:



Compiler Warnings: All [✓ Low] High None

Clipper Compatibility: Undeclared Variables ✓, Old Style Assignments ✓, Integer Divisions ✓, PROCNAME/PROCLINE ✓

Runtime Checking: Overflow checking ✓, Range checking ☐, Class checking ✓

Optimization: High Medium Low [✓ None]; [✓ Speed] Size

Miscellaneous: Type Inference ☐, Operator Methods ☐, Full runtime class information ☐, Warnings as Errors ☐

2. Reset any of the compiler options.

   Refer to **Setting Compiler Options** in "Working in the Desktop" for complete descriptions of these options.

3. Click OK.

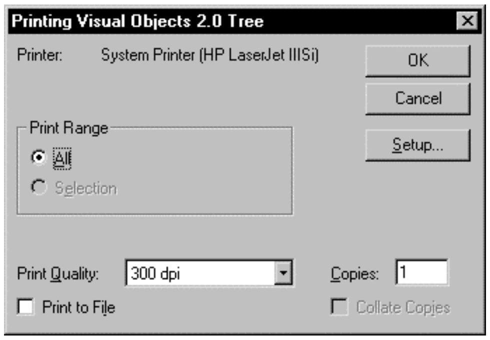
Printing in the Repository Explorer

There are multiple print options available in the Repository Explorer. You can either print the entire CA-Visual Objects 2.7 tree or just an individual listing of projects, applications, modules, or entities.

Printing the CA-Visual Objects 2.7 Tree

You can print a list of all projects, applications, modules, and entities stored in the repository using the Print toolbar button at the *root level* in the Repository Explorer. Alternatively, select the Print command from the File menu.

In either case the Printing Visual Objects 2.7 Tree dialog box appears:


Printing Visual Objects 2.0 Tree

Printer: System Printer (HP LaserJet IIISi)

Print Range: [✓ All] Selection

Print Quality: 300 dpi

Copies: 1

☐ Print to File

☐ Collate Copies

Refer to the online help system for more detailed information about this standard Print dialog box.

Printing Lists

You can also select the Print command from a local pop-up menu at the Repository Explorer's root level to print just a list of all projects within the Repository Explorer.

Additionally, at each subsequent level of the Repository Explorer, you can select the List command from a local pop-up menu to print a list of individual items within the current project, application or module.

For example, if you select the List local menu command at the Default Project level, the Printing Application List dialog box appears. Complete as desired and choose OK. A listing of all projects defined to the Default Project will be printed.

Refer to the online help system for more detailed information about the Printing Project List, Printing Application List, Printing Module List, and Printing Entity List dialog boxes.

Chapter

4

Using the Window Editor

The Window Editor allows you to create several types of windows based on subclassing the Window classes from the GUI Classes library. It provides predefined forms and drag-and-drop placement of buttons, edit controls, list boxes, combo boxes, toolbars, sliders, and other GUI and common controls. You can then specify properties for your forms and their controls, including color, font, annotation, online help keyword, and data server links. When you design and save a form, the Window Editor automatically generates object-oriented code that you can use in your application to create and activate the window.

This chapter describes how to use the Window Editor. In it, you will learn how to:

- Create a window by selecting a window type and customizing the corresponding predefined form
- Specify properties for the form (such as caption, description, type of mouse pointer, and help keyword)
- Place controls (such as push buttons and list boxes) on the form and specify properties for them
- Create predefined data and data dialog windows linked to data servers using the Auto Layout feature
- Make controls on data and data dialog windows *data-aware*, binding them to data server fields
- Modify form and control properties
- Print the form using the Window Editor
- Use the window you have created in an application

Important! *The GUI Classes and System Classes libraries must be included in the search path of your application in order to use a form generated by the Window Editor. The proper data server library, RDD Classes or SQL Classes, must also be included if the form is linked to a data server. See* ***Setting the Search Path*** *in "Using the Repository Explorer" for more information.*

Window Types

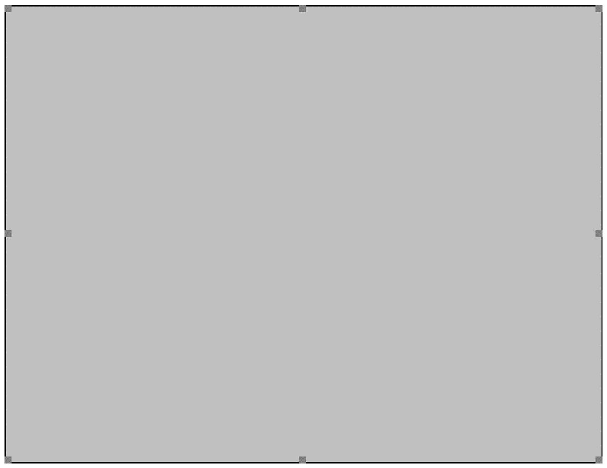
One of the key properties of a window is its *type*, which you define when you first load the Window Editor to create a new window. (See Creating a Window later in this chapter.) The following window types are available—each one represents a subclass of the Window class.

Note: A new window type—OLEDataWindow—was added in this version of CA-Visual Objects. An OLE data window is essentially the same as a regular data window, described below, except that it contains additional logic for the deactivation of an in-place active OLE object by clicking anywhere in the window. For more detailed information about OLE data windows, refer to the online help.

DataWindow

DataWindow defines a window that may be linked to a data server. You can link (bind) many types of controls to data server fields using the field name as the name of the control. Also, you must specify a field spec for a data-aware control to inherit such properties as picture, validation rules, and description. Consequently, data windows are often referred to as *data-aware* windows.

A blank data window form contains *no* default controls:



At runtime, the data window will contain appropriate default controls (such as title bar, system menu, and minimize and maximize buttons), depending on how the window is used. Scroll bars will also be added automatically at runtime when the data window is resized and needs to be scrolled.

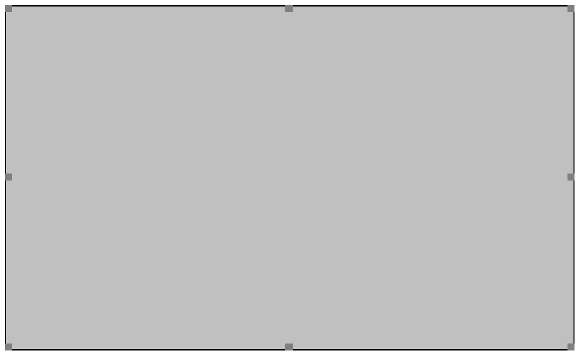
Note that when creating a data window, you can use the Auto Layout feature to create controls based on the fields in the associated data server.

See **Chapter 8: Creating Data-Aware Windows** for detailed information about creating data windows and using the Auto Layout feature. You may also want to review **Chapter 7: Defining Data Servers** and Field Specifications for more information about data servers and field specs, in general.

DataDialog

DataDialog defines a window that combines features from a data window and a dialog window, allowing for the creation of *modal* data-aware windows. Whereas a regular data window is an MDI child window, a data dialog window behaves like a modal dialog window. This means that the end user must respond to the window and close it before continuing with the application.

Like a data window, a blank data dialog form contains *no* default controls:



Similarly, at runtime the data dialog window will also contain appropriate default controls (such as title bar, system menu, and minimize and maximize buttons), depending on how the window is used. However, scroll bars will not be added automatically at runtime because the data dialog window cannot be resized.

Note that when creating a data dialog window, you can also use the Auto Layout feature to create controls based on the fields in the associated data server.

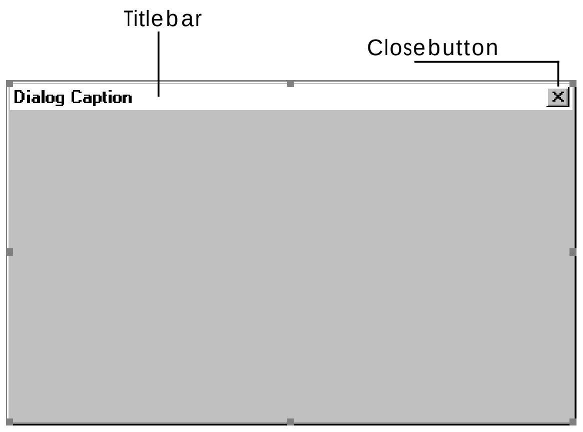
See **Chapter 8: Creating Data-Aware Windows** for more detailed information about creating data dialog windows and using the Auto Layout feature.

DialogWindow

DialogWindow defines a secondary, usually transient subwindow—a *dialog* or a *dialog box*—used to collect or display data. The program creates either modal dialog boxes that require the user to respond before the application can proceed, or modeless dialog boxes.

Note: Neither modal dialog boxes nor modeless dialog boxes are data-aware.

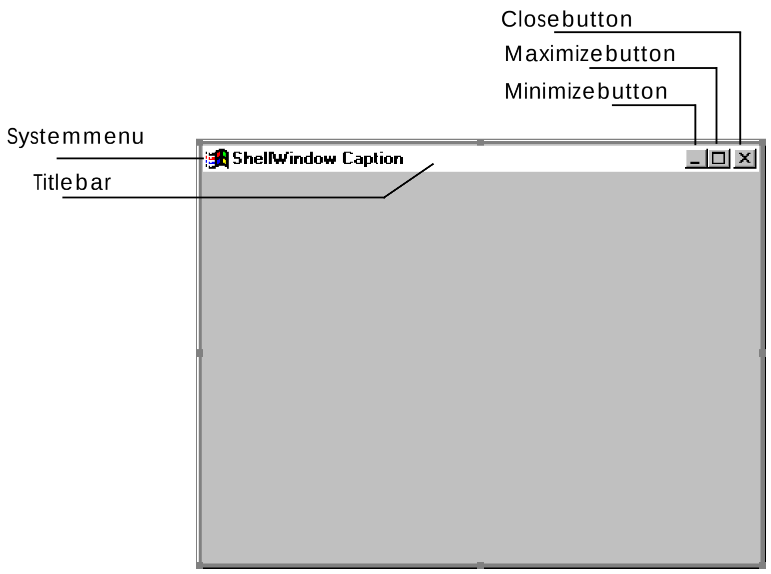
A blank dialog window form includes a predefined title bar and a Close button, both of which are fully functional:



ShellWindow

ShellWindow defines the main window within an MDI application.

A blank shell window form includes a predefined title bar, system menu, and Minimize, Maximize, and Close buttons, all of which are fully functional:

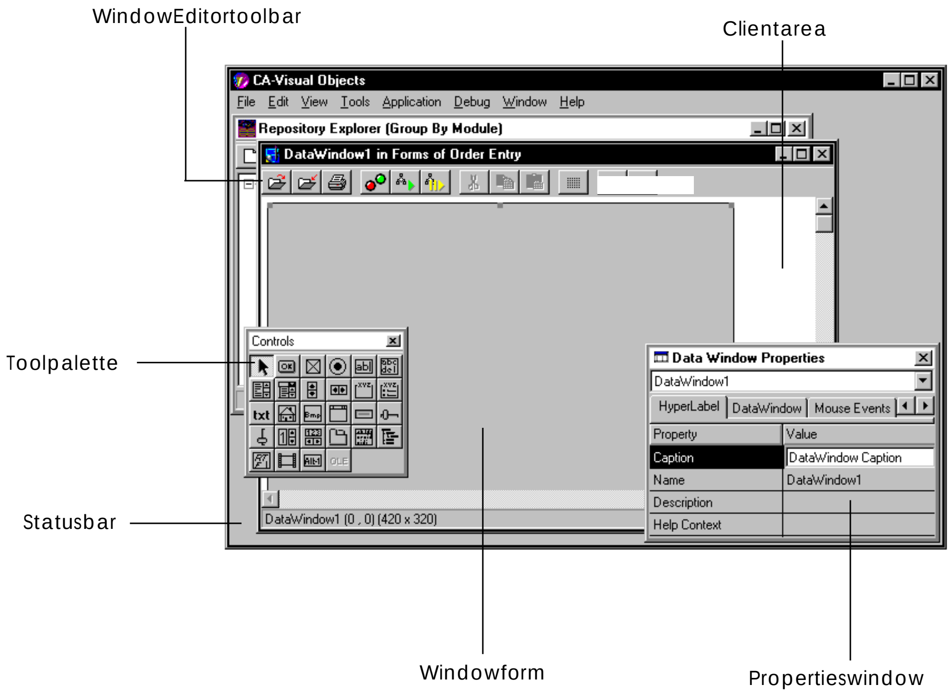


Workspace Overview

The Window Editor is the primary workspace in the IDE for creating, viewing, and modifying windows. When you are in the Window Editor, you can:

- Create, edit, cut, copy and paste window controls, and print the form
- 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 Window Editor has its own toolbar, status bar, client area, tool palette, and a Properties window. For example, when first loaded for a new data window, it looks like this:


WindowEditortoolbar
Clientarea
CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
DataWindow1 in Forms of Order Entry
Toolpalette
Controls
Data Window Properties
DataWindow1
HyperLabel DataWindow Mouse Events
Statusbar
DataWindow1 (0 , 0) (420 x 320)
Windowform
Propertieswindow

Property	Value
Caption	DataWindow Caption
Name	DataWindow1
Description	
Help Context	

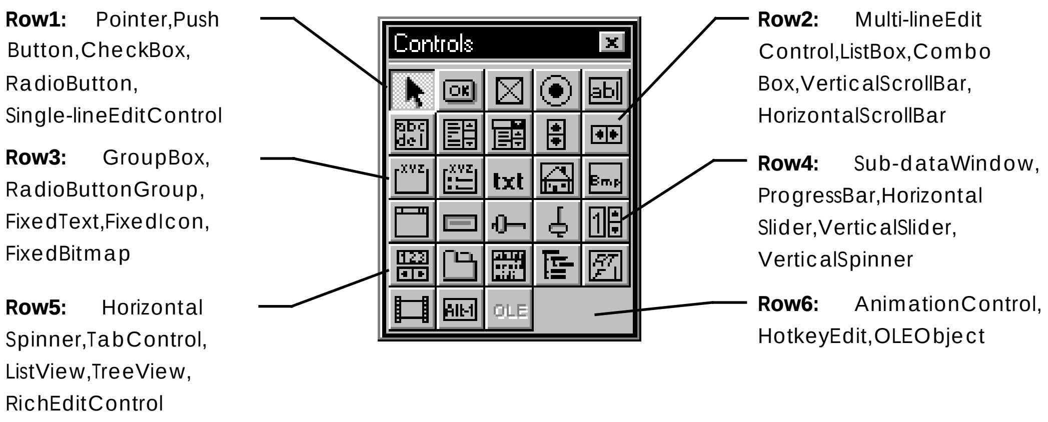
Window Forms

When you create a new window, the Window Editor's client area contains a blank, prepainted window template, or *form*, based on the window type you have selected. This form can then be customized. (These forms were shown earlier in this chapter.) For example, a ShellWindow form contains a title bar, a system menu button, and minimize and maximize buttons as defaults, whereas a DialogWindow form has only a default title bar and system menu button.

The Tool Palette

Note: In CA-Visual Objects 2.7, there are new Windows common controls—ComboBoxEx control, date time picker, IP address, and month calendar—and a new CA-Visual Objects custom control, data list view. Naturally, the Window Editor's tool palette has been updated to reflect these additions. For detailed information, see the online help.

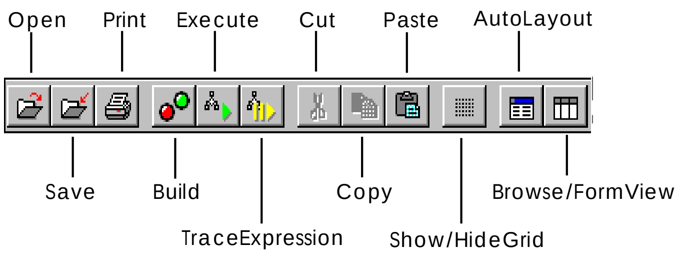
The Window Editor's tool palette contains a set of icons that allow you to quickly and easily place controls on your forms:


Row1: Pointer,Push Button,CheckBox, RadioButton, Single-lineEditControl
Row3: GroupBox, RadioButtonGroup, FixedText,FixedIcon, FixedBitmap
Row5: Horizontal Spinner,TabControl, ListView,TreeView, RichEditControl
Controls
Row2: Multi-lineEdit Control,ListBox,Combo Box,VerticalScrollBar, HorizontalScrollBar
Row4: Sub-dataWindow, ProgressBar,Horizontal Slider,VerticalSlider, VerticalSpinner
Row6: AnimationControl, HotkeyEdit,OLEObject

Note: The tool palette is not available for shell windows because shell windows, by convention, do not have controls.

The Toolbar

The Window Editor's toolbar contains the following buttons:



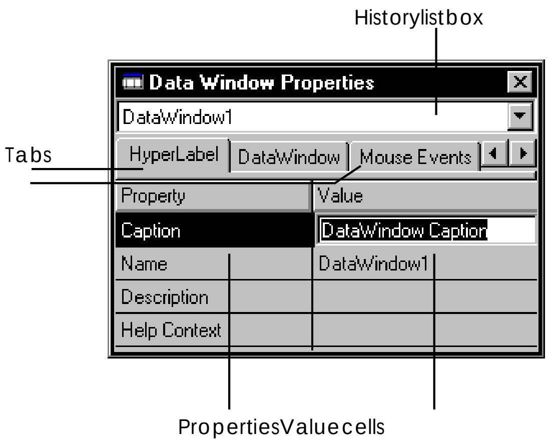
All of the buttons except Build, Execute, and Trace Expression are discussed in this chapter. See **Chapter 2: Working in the** Desktop and **Chapter 11: Debugging Your Applications** for information about these buttons.

Tip: For a quick description of each toolbar button, look at the tooltip windows as the mouse pointer passes over the buttons.

The Properties Window

Note: See the online help for detailed information about new and updated window properties.

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



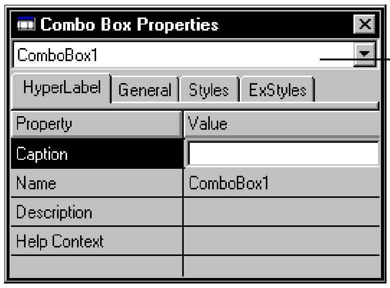
Property	Value
Caption	DataWindow Caption
Name	DataWindow1
Description	
Help Context	

The *Property* column lists all properties that can be specified for the currently selected form or control, 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, a keyword for use in a context-sensitive help system, or the type of mouse pointer. The available properties vary depending on the window type, and are grouped logically within tabbed pages, such as HyperLabel and Mouse Events.

The *history* list box at the top of the Properties window allows you to switch easily to any other control already defined to your form.

Tip: You can change both the layout of the tabbed pages in any Properties window and the defaults for all of their properties by modifying the CAVOWED.INF file. For detailed information about the CAVOWED.INF and CAVOWED.TPL files, refer to the *CA-Visual Objects 2.7 Software Development Kit.*

As you design the form, the Properties window takes on different roles, depending on the currently selected item in the workspace. For example, if you place a combo box control in the form, the Properties window is used to specify properties for the combo box control when it is selected:



Property	Value
Caption	
Name	ComboBox1
Description	
Help Context	

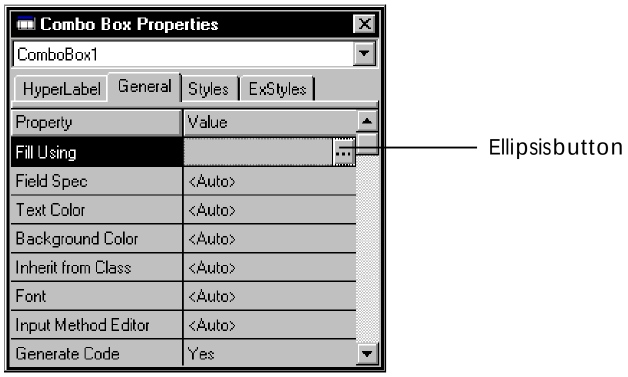
Notice that the title bar and the contents of the History list box change, and different tabs appear based on what is currently selected.

Note: See the online help for detailed information about new and updated control properties.

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

- Click on the Ellipsis button (shown below) and
  - Enter a value using the Source Code Editor or
  - Fill in a corresponding dialog box



Property	Value
Fill Using	
Field Spec	<Auto>
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Font	<Auto>
Input Method Editor	<Auto>
Generate Code	Yes

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

Note: The Properties window always remains open until explicitly closed (using the system menu or the Show Properties Window command) or until its owner, the Window Editor window, is closed or deactivated. If explicitly closed, reopen it at any time using the Show Property Window command on 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.

Creating a Window

Now that you have a general overview of what you can do with the Window Editor, you are ready to use it to define a new window. In this section, you will learn how to:

- Create a window by selecting a window type and predefined form
- Specify properties for the form
- Place controls on the form and specify properties for them
- Modify form properties and controls

This section gives you a complete description of all possible properties for all window and control types and discusses creating a window in a general sense. The “Creating Data-Aware Windows” chapter later in this guide gives you more detailed information about defining data and data dialog windows in particular.

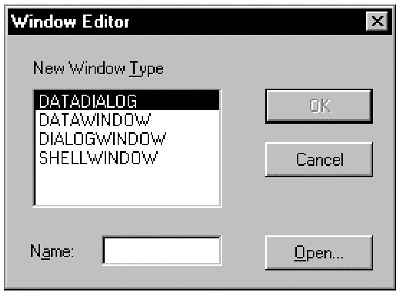
Basic Steps

To create a window, perform the following steps:

1. Start the Window Editor.

   Like all CA-Visual Objects tools, the Window Editor is accessed using the Tools menu or the New Entity toolbar button.

   For all new forms that you create, you must first define the window type. The Window Editor dialog box automatically appears, allowing you to choose a window type and enter a name for the form:

   Note: This dialog box has been updated in this version of CA-Visual Objects to reflect the addition of the new window type, OLEDATAWINDOW.

   Note: You can choose to edit an existing form instead of creating a new one by clicking Open. When clicked, the Window Editor dialog box changes so that it contains an Open Window list box, from which you can choose an existing form entity. Note that there are also a New button and a Clone button. The New button, when clicked, toggles you back to the above dialog box. The Clone button creates a duplicate of an existing form in the current module, appending an underscore and a number to the original form's name. For example, if you clone the Standard Application's HelpAbout window, the Window Editor is launched with a cloned form, HelpAbout_1.

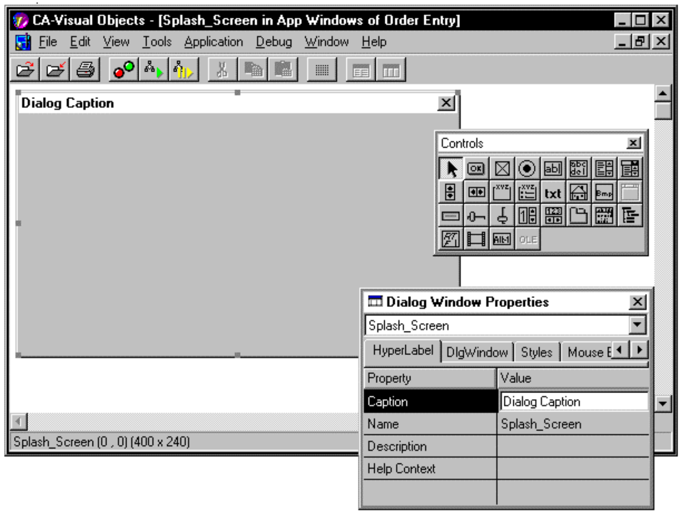
2. Choose the type of window you want to define from the New Window Type list box (for example, DIALOGWINDOW).

3. Enter a name for the new form in the Name edit control (for example, **Splash Screen**).

   Blank spaces within the name will be converted to underscores. This name will be used in the source code generated by the Window Editor to create a class entity (among others) that is a subclass of the window type that you selected above; therefore, it must not conflict with other entity names in your application. The OK button is disabled until you enter a name.

4. Choose OK to launch the Window Editor.

A predefined form corresponding to the window type is displayed in the Window Editor. For example:


CA-Visual Objects - [Splash_Screen in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Dialog Caption
Controls
Dialog Window Properties
Splash_Screen
HyperLabel DlgWindow Styles Mouse E
Splash_Screen (0 , 0) (400 x 240)

Property	Value
Caption	Dialog Caption
Name	Splash_Screen
Description	
Help Context	

5. Specify properties for the form. For example, enter **Order Entry** in the Caption property's value cell.

   See Specifying Window Properties later in this chapter for more information.

6. Customize the form by selecting controls from the tool palette (or using the Select from Palette menu commands), placing them in the desired locations on the blank form. For example, place three Fixed Text controls in the center of the sample Order Entry splash screen.

   You will also need to add action code; and you can add other controls, like a fixed icon for your company logo, or even an animation control to amuse the end user while the application is loading. See Placing Controls on a Form later in this chapter for more information.

   Note: For a data-aware window, you can also use the Auto Layout feature to place predefined controls on the form. See **Chapter 11: Debugging Your Applications** for more information.

7. Specify properties for each control.

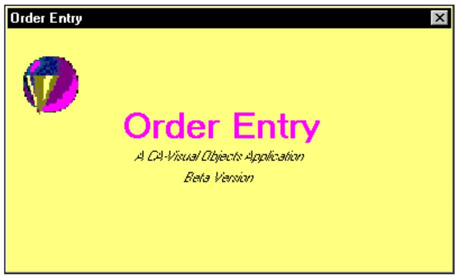
   For example, enter the following text, respectively, in the Caption property for each of the Fixed Text controls: **Order Entry**, **A CA-Visual Objects Application**, and **Beta Version**.

   Of course, you can define other properties for the text, such as font style, size, and color. See Specifying Control Properties and Style Settings later in this chapter for more detailed information. Also, refer to the online help for new and updated properties.

8. Choose the Save toolbar button to save the form. (Later you can edit the form, adding the necessary code to actually invoke Order Entry.)

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

9. Choose the Test Mode command from the View menu to view the splash screen as it will appear to the end user. For example:

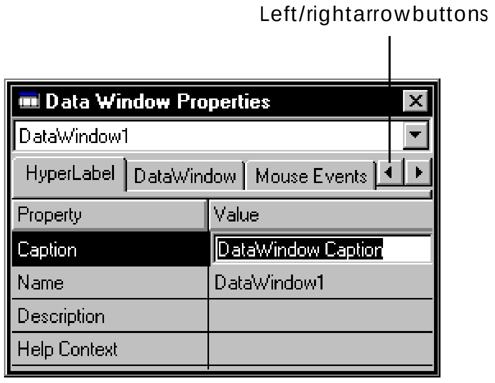


Note: The new Application Gallery can automatically create splash screens and Help About dialog boxes for your applications. For detailed information about these application options that are *exclusive* to the Application Gallery, see the online help.

Tip: The Open toolbar button can be used at any time to begin editing another form without shutting down the Window 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 Window Editor opens the new form for editing.

Specifying Window Properties

You define properties for a form using the Properties window mentioned earlier. The Properties window for a data window is shown again below:



Property	Value
Caption	DataWindow Caption
Name	DataWindow1
Description	
Help Context	

As mentioned earlier, the *Property* column in each tab of the Properties window lists the properties that can be specified for the currently selected form, and the *Value* column contains the corresponding cells where you specify a value. The HyperLabel, Mouse Events, Window Events, Control Events, Common Control Events, and Menu Events tabs are *common* to all window types. Furthermore, the CommonControl Events, TreeView Events, and ListView Events tabs are common to all windows *except* shell windows. Unique to each window type, however, is a namesake tab; and Dialog Windows also has a Styles tab.

Tip: Use the Left and Right arrow buttons to access any tabs that may be hidden from view.

Common Properties

Many of the window properties—such as Caption, Name, Description, Help Context, as well as mouse, window, control, and menu event controls—are common to *all* windows. Others are specific to each window type. Descriptions of the common properties in tab order follow immediately. (Properties that are unique to each window type are described afterwards.)

Note: In this version of CA-Visual Objects, data windows and data dialog windows have several new properties, including Browser Inherits From, Columns Inherit From, Defer USE, Allow Server Close, and Quit on Close. Other common properties have been updated. For example, the Background Color property now gives you immediate access to CA-Visual Objects standard Color dialog box for defining a form's background color. Therefore, refer to the online help for detailed information about *all* window properties.

HyperLabel Tab

Caption	Enter the text that will appear in the form's title bar.
Name	Enter the name of your form. This property value is required and is automatically filled in with the name you entered in the initial dialog box when you first create the form. The form name is used in the source code generated by the Window Editor to create a class entity, a form entity, and any resource entities needed to create the window; therefore, it must be unique.
Description	Enter the description text. This text will not appear in the application's status bar, but may be accessed since it is part of the hyperlabel for a form.

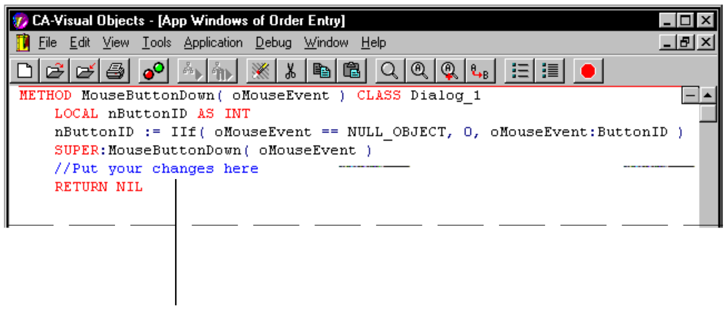
Help Context

Enter a unique keyword for the form that can be used to identify it in a context-sensitive help system. This property can consist of letters, digits, and the underscore character (_), but it cannot contain any other characters, including blanks or punctuation. (For more information on creating a help file, refer to the "GUI Classes" chapter in the *Programmer's Guide*.)

Mouse Events Tab

Enter source code that you want to run when a specific mouse-related event occurs. For example, you can specify event action code for the following events: MouseButtonDoubleClick, MouseButtonDown, MouseButtonUp, MouseDrag, and MouseMove.

To enter the source code for an undefined event in *any* of the Events tabs, click the Ellipsis button in the appropriate Value cell. The Source Code Editor will be launched with a skeleton method for the event (for example, MouseButtonDown):


CA-Visual Objects - [App Windows of Order Entry]
File Edit View Tools Application Debug Window Help

```
METHOD MouseButtonDown( oMouseEvent ) CLASS Dialog_1
    LOCAL nButtonID AS INT
    nButtonID := IIf( oMouseEvent == NULL_OBJECT, 0, oMouseEvent:ButtonID )
    SUPER:MouseButtonDown( oMouseEvent )
    //Put your changes here
    RETURN NIL
```

Entereventsourcecodehere...

You can write the code for the event and exit the Source Code Editor to save your changes. Your source code will be saved, along with the other code generated by the Window Editor; and the appropriate Value cell in the Properties window is updated from "Not Defined" to "Defined."

Tip: You can modify the generated code by changing the CAVOWED.INF and CAVOWED.TPL files. For detailed information about these files, refer to the *CA-Visual Objects* 2.7 *Software Development Kit*.

Window Events Tab

Enter source code that you want to run when a specific window-related event occurs. For example, you can specify event action code for the following events: Activate, Close, Deactivate, Draw, Expose, FocusChange, HelpRequest, Move, QueryClose, and Resize.

Control Events Tab

Enter source code that you want to run when a specific control-related event occurs. For example, you can specify event action code for the following events: ButtonClick, ButtonDoubleClick, EditChange, EditFocusChange, EditScroll, Keydown, Keyup, ListBoxClick, ListBoxSelect, HorizontalScroll, and VerticalScroll.

CommonControl Events Tab

Enter source code that you want to run when a specific common control-related event occurs. For example, you can specify event action code for the following events: AnimationStart, AnimationStop, RichEditDropFiles, RichEditProtected, RichEditSelChange, RichEditUndoLost, TabSelect, TabSelChanging, and TabKeyDown.

Note: The CommonControl Events tab is not available for shell windows.

TreeView Events Tab

Enter source code that you want to run when a specific tree view control-related event occurs. For example, you can specify event action code for the following events: TreeViewMouseDown, TreeViewMouseDoubleClick, TreeViewItemDrag, TreeViewItemEdit, TreeViewItemDelete, TreeViewItemExpanded, TreeViewItemExpanding, TreeViewKeyDown, TreeViewSelChanged, and TreeViewSelChanging.

Note: The TreeView Events tab is not available for shell windows.

ListView Events Tab

Enter source code that you want to run when a specific list view control-related event occurs. For example, you can specify event action code for the following events: ListViewItemDrag, ListViewItemEdit, ListViewColumnClick, ListViewItemDelete, ListViewKeyDown, ListViewItemMouseDown, and ListViewMouseDoubleClick.

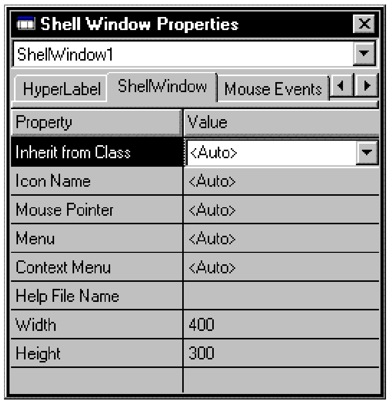
Note: The ListView Events tab is not available for shell windows.

Menu Events Tab

Enter source code that you want to run when a specific menu-related event occurs. For example, you can specify event action code for the following events: MenuCommand, MenuInit, and MenuSelect.

Shell Window Properties

The ShellWindow tab, shown below, allows you to define the physical appearance of your shell window—including its icon and mouse style—and to choose a binary menu entity and a help file to be associated with it.



Property	Value
Inherit from Class	<Auto>
Icon Name	<Auto>
Mouse Pointer	<Auto>
Menu	<Auto>
Context Menu	<Auto>
Help File Name	
Width	400
Height	300

Note: Refer to the online help for detailed information about new and updated shell window properties.

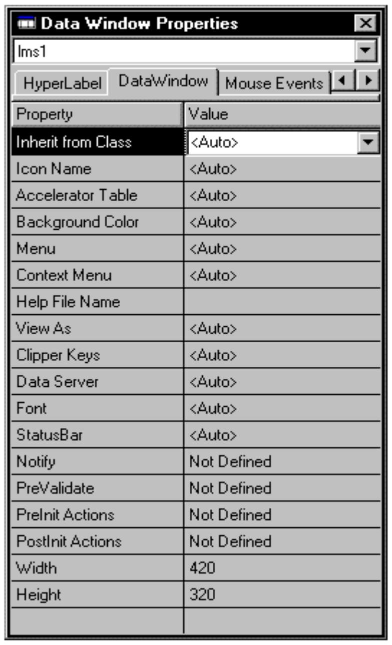
The ShellWindow tab contains the following properties:

Inherit from Class	Select the Window class or subclass from which the shell window will inherit its basic characteristics. Valid choices are <Auto>, STANDARDSHELLWINDOW, and all classes derived from the ShellWindow class that are in the current search path. The default setting is <Auto>.
Icon Name	Enter or choose the name of the Icon subclass defining the icon to be displayed when a shell window is minimized. Choices include all icon entities defined for the current application. The default value is <Auto>, which means a default icon is displayed. See **Chapter 10: Using the Image** Editor for information on defining icons.
Mouse Pointer	Enter or choose the name of the Pointer subclass defining the mouse style for a shell window. Choices include all cursor entities defined for the current application. The default value is <Auto>, which means the system pointer is used. See **Chapter 10: Using the Image** Editor for information on defining cursors.
Menu	Enter or choose the name of the menu to associate with a shell window. Valid choices are <Auto>, EMPTYSHELLMENU, STANDARDSHELLMENU, and all classes derived from Menu class that are in the current search path. The default value is <Auto>, which means no menu is used. See **Chapter 5: Using the Menu** Editor for information on defining menus.

Context Menu	Enter or choose the name for a local pop-up menu to be accessed by right-clicking. Valid choices are <Auto>, EMPTYSHELLMENU, STANDARDSHELLMENU, and all classes derived from the Menu class that are in the current search path. The default value is <Auto>, which means no menu is used.
Help File Name	The path and file name for the .HLP file you want to use with the form. Then you can assign a help topic to the form using the Help Context property. The designated help file will also be used to provide context help for the controls on the form. If no help file is assigned, the help file for the owner window will be used. (For more information on creating a help file, refer to the "GUI Classes" chapter in the *Programmer's Guide.*)
Width	Enter the horizontal size of the shell window in pixels. The default is 400.
Height	Enter the vertical size of the shell window in pixels. The default is 300.

Data Window Properties

The DataWindow tab, shown below, allows you to define the physical appearance of your data window—including its icon, background color, and font—and to choose a binary menu entity, help file, and data server to be associated with it.



Property	Value
Inherit from Class	<Auto>
Icon Name	<Auto>
Accelerator Table	<Auto>
Background Color	<Auto>
Menu	<Auto>
Context Menu	<Auto>
Help File Name	
View As	<Auto>
Clipper Keys	<Auto>
Data Server	<Auto>
Font	<Auto>
StatusBar	<Auto>
Notify	Not Defined
PreValidate	Not Defined
PreInit Actions	Not Defined
PostInit Actions	Not Defined
Width	420
Height	320

Note: Refer to the online help for detailed information about new and updated data window properties.

The DataWindow tab contains the following properties:

Inherit from Class	Select the Window class or subclass from which the data window will inherit its basic characteristics. Valid choices are <Auto>, STDDATAWINDOW, and all classes derived from the DataWindow class that are in the current search path. The default setting is <Auto>.
Icon Name	Enter or choose the name of the Icon subclass defining the icon to be displayed when a data window is minimized. Choices include all icon entities defined for the current application. The default value is <Auto>, which means a default icon is displayed. See **Chapter 10: Using the Image** Editor for information on defining icons.
Accelerator Table	This property is available for data-aware windows only. If you do not want a menu to be associated with the form, you may still want to designate a table of accelerator keys that can be used. Valid choices are: <Auto>, EMPTYSHELLMENU, STANDARDSHELLMENU, and all classes derived from the Menu class that are in the current search path. **Note:** If a menu is associated with the form, then the accelerator keys defined in the Menu Editor will be used; in this case the default <Auto> setting should be used.
Background Color	Enter or choose the color for a data window background from a drop-down list box. The default value is <Auto>, which means the currently selected system color is used.
Menu	Enter or choose the name of the menu to associate with a data window. Choices include <Auto>, EMPTYSHELLMENU, STANDARDSHELLMENU, and all classes derived from the Menu class that are in the current search path. The default value is <Auto>, which means the no menu is used. See **Chapter 5: Using the Menu** Editor for information on defining menus.
Context Menu	Enter or choose the name for a local pop-up menu to be accessed by right-clicking. Valid choices are <Auto>, EMPTYSHELLMENU, STANDARDSHELLMENU, and all classes derived from the Menu class that are in the current search path. The default value is <Auto>, which means no menu is used.
Help File Name	The path and file name for the .HLP file you want to use with the form. Then you can assign a help topic to the form using the Help Context property. The designated help file will also be used to provide context help for the controls on the form. If no help file is assigned, the help file for the owner window will be used. (For more information on creating a help file, refer to the “GUI Classes” chapter in the *Programmer’s Guide.*)

View As	This property is available for data-aware windows only. Choose the default view to use when the data window is initially accessed. Every data window can operate in either *form view*, displaying multiple fields for a single record as fixed text and data controls, or *browse view*, using a spreadsheet-like data browser that displays multiple fields and records in a table in the data window. The available choices for this property, therefore, are #FormView or #BrowseView.

The default value is <Auto>, which means #BrowseView is used for all data windows placed as controls, such as a detail window nested within another data window, and #FormView is used for all other data windows.

Note: Clicking the Browse/Form View toolbar button changes the Window Editor display to a different view, as well as changing the value of this property.

Tip: Data windows are different from other windows in that they can be placed on other forms as controls (similar to a check box or push button). When used as controls, they are referred to as *sub-data windows*. (See **Chapter 8: Creating Data-Aware Windows** for more information about sub-data window controls.)

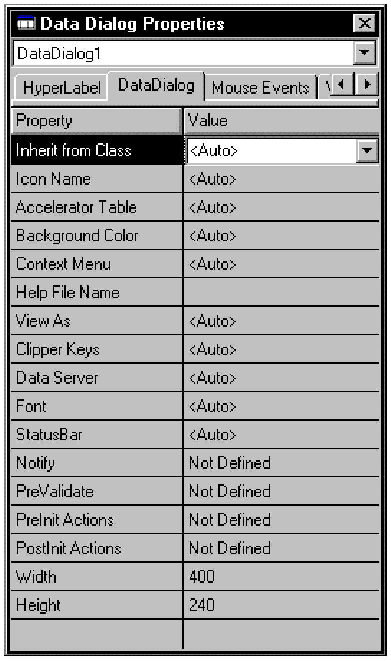
Note: You can give the end user the option of switching back and forth between form and browse view at runtime by associating a standard menu with the data window using the Menu property. The commands for switching between browse and form view are part of the View menu defined in the default StandardShellMenu binary menu entity, which is included automatically in every GUI application that you create based on the Standard Application. See **Chapter 8: Creating Data-Aware Windows** for more information on customizing these two views.

Clipper Keys	If True, allows the user to navigate through a form by using the Direction and Return keys. Otherwise the Windows default navigation keys (Tab key) are effective. Valid values are: <Auto>, True, and False. The default is <Auto>.
Data Server	This property is available for data-aware windows only. In it, you enter or choose the name of the data server that is associated with this form. This property is required and is automatically filled in when you choose the Edit Auto Layout menu command. See **Chapter 8: Creating Data-Aware Windows** for more information.
Font	Choose a font to use for the form text. A standard Font dialog box appears, displaying those fonts available on your system. The default value is <Auto>, which means the system font is used.
StatusBar	If Yes, adds a status bar to the data window. The default value is <Auto>.

Notify Event	Optionally enter additional source code for the skeleton Notify event. The default setting is Not Defined. See the online help system for detailed information about the Notify event.
PreValidate Event	Optionally enter additional source code for the skeleton PreValidate event. The default setting is Not Defined. See the online help system for detailed information about the PreValidate event.
PreInit Actions	Optionally enter additional source code for the skeleton PreInit method. The default setting is Not Defined. See the online help system for detailed information about the PreInit method.
PostInit Actions	Optionally enter additional source code for the skeleton PostInit method. The default setting is Not Defined. See the online help system for detailed information about the PostInit method.
Width	Enter the horizontal size of the data window in pixels. The default is 420.
Height	Enter the vertical size of the data window in pixels. The default is 320.

DataDialog Window Properties

The DataDialog tab, shown below, allows you to define the physical appearance of your data dialog window—including its icon, background color, and font—and to choose a binary menu entity, help file, and data server to be associated with it.



Property	Value
Inherit from Class	<Auto>
Icon Name	<Auto>
Accelerator Table	<Auto>
Background Color	<Auto>
Context Menu	<Auto>
Help File Name	
View As	<Auto>
Clipper Keys	<Auto>
Data Server	<Auto>
Font	<Auto>
StatusBar	<Auto>
Notify	Not Defined
PreValidate	Not Defined
PreInit Actions	Not Defined
PostInit Actions	Not Defined
Width	400
Height	240

Note: Refer to the online help for detailed information about new and updated data dialog window properties.

The DataDialog tab contains the following properties:

Inherit from Class	Select the Window class or subclass from which the data dialog window will inherit its basic characteristics. Valid choices are <Auto>, DATADIALOGWINDOW, and all classes derived from the DataDialog class that are in the current search path. The default setting is <Auto>.
Icon Name	Enter or choose the name of the Icon subclass defining the icon to be displayed when a data dialog window is minimized. Choices include all icon entities defined for the current application. The default value is <Auto>, which means a default icon is displayed. See **Chapter 10: Using the Image** Editor for information on defining icons.
Accelerator Table	This property is available for data-aware windows only. If you do not want a menu to be associated with the form, you may still want to designate a table of accelerator keys that can be used. Valid choices are: <Auto>, EMPTYSHELLMENU, STANDARDSHELLMENU, and all classes derived from the Menu class that are in the current search path.

Note: If a menu is associated with the form, then the accelerator keys defined in the Menu Editor will be used; in this case the default <Auto> setting should be used.

Background Color

Enter or choose the color for a data dialog window's background from a drop-down list box. The default value is <Auto>, which means the currently selected system color is used.

Context Menu

Enter or choose the name for a local pop-up menu to be accessed by right-clicking. Valid choices are <Auto>, EMPTYSHELLMENU, STANDARDSHELLMENU, and all classes derived from the Menu class that are in the current search path. The default value is <Auto>, which means no menu is used.

Help File Name

The path and file name for the .HLP file you want to use with the form. Then you can assign a help topic to the form using the Help Context property. The designated help file will also be used to provide context help for the controls on the form. If no help file is assigned, the help file for the owner window will be used. (For more information on creating a help file, refer to the "GUI Classes" chapter in the *Programmer's Guide.*)

View As

This property is available for data-aware windows only. Choose the default view to use when the data dialog window is initially accessed. Every data dialog window can operate in either *form view*, displaying multiple fields for a single record as fixed text and data controls, or *browse view*, using a spreadsheet-like data browser that displays multiple fields and records in a table in the data dialog window. The available choices for this property, therefore, are #FormView or #BrowseView.

The default value is <Auto>, which means #BrowseView is used for all data dialog windows placed as controls and #FormView for all other data dialog windows.

Note: Clicking the Browse/Form View toolbar button changes the Window Editor display to a different view, as well as changing the value of this property.

Note: You can give the end user the option of switching back and forth between form and browse view at runtime by associating a standard menu with the data dialog window using the Menu property. The commands for switching between browse and form view are part of the View menu defined in the default StandardShellMenu binary menu entity, which is included automatically in every GUI application that you create based on the Standard Application. See **Chapter 8: Creating Data-Aware Windows** for more information on customizing these two views.

Clipper Keys

If True, allows the user to navigate through a form by using the Direction and Return keys. Otherwise the Windows default navigation keys (Tab key) are effective. Valid values are: <Auto>, True, and False. The default is <Auto>.

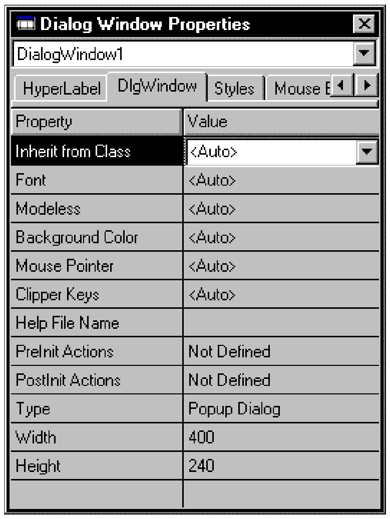
Data Server	This property is available for data-aware windows only. In it, you enter or choose the name of the data server that is associated with this form. This property is required and is automatically filled in when you choose the Edit Auto Layout menu command. See **Chapter 8: Creating Data-Aware Windows** for more information.
Font	Choose a font to use for the form text. A standard Font dialog box appears, displaying those fonts available on your system. The default value is <Auto>, which means the system font is used.
StatusBar	If Yes, adds a status bar to the data dialog window. The default value is <Auto>.
Notify Event	Optionally enter additional source code for the skeleton Notify event. The default setting is Not Defined. See the online help system for detailed information about the Notify event.
PreValidate Event	Optionally enter additional source code for the skeleton PreValidate event. The default setting is Not Defined. See the online help system for detailed information about the PreValidate event.
PreInit Actions	Optionally enter additional source code for the skeleton PreInit method. The default setting is Not Defined. See the online help system for detailed information about the PreInit method.
PostInit Actions	Optionally enter additional source code for the skeleton PostInit method. The default setting is Not Defined. See the online help system for detailed information about the PostInit method.
Width	Enter the horizontal size of the data dialog window in pixels. The default is 400.
Height	Enter the vertical size of the data dialog window in pixels. The default is 240.

Dialog Window Properties

The Properties window for dialog windows has two additional tabs that are unique: DlgWindow and Styles.

DlgWindow Tab

The DlgWindow tab, shown below, allows you to define the physical appearance of your dialog window—including its font, background color, and mouse style—and whether it is to be modal or modeless.



Property	Value
Inherit from Class	<Auto>
Font	<Auto>
Modeless	<Auto>
Background Color	<Auto>
Mouse Pointer	<Auto>
Clipper Keys	<Auto>
Help File Name	
PreInit Actions	Not Defined
PostInit Actions	Not Defined
Type	Popup Dialog
Width	400
Height	240

Note: Refer to the online help for detailed information about new and updated dialog window properties.

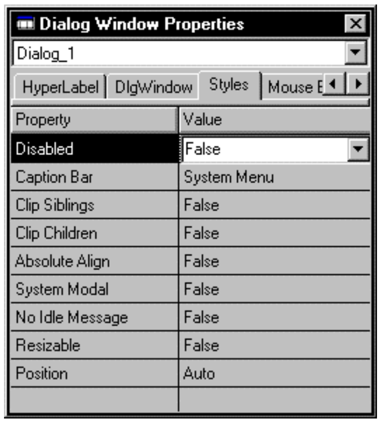
This tab contains the following properties:

Inherit from Class	Select the Window class or subclass from which the dialog window will inherit its basic characteristics. Valid choices are: <Auto>, _FORMDIALOGWINDOW, MODELESSDIALOG, HELPABOUT, and all classes derived from the DialogWindow class that are in the current search path. The default is <Auto>.
Font	Choose a font to use for the form text. A standard Font dialog box appears, displaying those fonts available on your system. The default value is <Auto>, which means the system font is used.
Modeless	This property is available for dialog windows only. A value of Yes creates a modeless dialog box. The default value is <Auto>, which means a modal dialog box will be created.
Background Color	Enter or choose the color for a dialog window background from a drop-down list box. The default value is <Auto>, which means the currently selected system color is used.
Mouse Pointer	Enter or choose the name of the Pointer subclass defining the mouse style for a dialog window. Choices include all cursor entities defined for the current application. The default value is <Auto>, which means the system pointer is used. See **Chapter 10: Using the Image** Editor for information on defining cursors.

Clipper Keys	If True, allows the user to navigate through a form by using the Direction and Return keys. Otherwise the Windows default navigation keys (Tab key) are effective. Valid values are: <Auto>, True, and False. The default is <Auto>.
Help File Name	The path and file name for the .HLP file you want to use with the form. Then you can assign a help topic to the form using the Help Context property. The designated help file will also be used to provide context help for the controls on the form. If no help file is assigned, the help file for the owner window will be used. (For more information on creating a help file, refer to the “GUI Classes” chapter in the *Programmer’s Guide.*)
PreInit Actions	Optionally enter additional source code for the skeleton PreInit method. The default setting is Not Defined. See the online help system for detailed information about the PreInit method.
PostInit Actions	Optionally enter additional source code for the skeleton PostInit method. The default setting is Not Defined. See the online help system for detailed information about the PostInit method.
Type	Select the type of dialog window to be created. Valid choices are: Popup Dialog, a standard dialog window, and Tab Page, a dialog window that will become a page within a tab control. The default is Popup Dialog.
Width	Enter the horizontal size of the dialog window in pixels. The default is 400.
Height	Enter the vertical size of the dialog window in pixels. The default is 240.

Styles Tab

The Styles tab, shown below, is available only for dialog windows. It allows you to define various style settings.



Property	Value
Disabled	False
Caption Bar	System Menu
Clip Siblings	False
Clip Children	False
Absolute Align	False
System Modal	False
No Idle Message	False
Resizable	False
Position	Auto

Note: Refer to the online help for detailed information about new and updated dialog window properties.

This tab contains the following options:

Disabled	Creates a dialog box that is initially disabled, if True. The default setting is False.
Caption Bar	Specifies whether a caption bar should be added to the dialog box and, if so, defines its contents. Valid choices are: No Caption Bar, Caption Bar Only, System Menu, Minimize Box, Maximize Box, and Both Boxes. The default setting is System Menu.
Clip Siblings	Clips "sibling" windows from the area occupied by another child window that currently has focus, if True. The default setting is False.
Clip Children	Clips child windows from the area occupied by the window that currently has focus, if True. The default setting is False.
Absolute Align	If True, aligns the dialog box coordinates relative to the upper-left corner of the screen instead of the current form. The default setting is False.
System Modal	If True, the dialog window utilizes the WN_EX_TOPMOST style which keeps it on top of all other windows while open. The default setting is False.
No Idle Message	If True, suppresses Windows idle messages while a system-modal dialog box is displayed. This option is effective only if the System Modal option is selected. The default setting is False.
Resizable	If True, the dialog box may be resized by dragging one of the borders. When this type of dialog box loses focus, its thick border does not change. The default setting is False.
Position	Specifies the position of the dialog window in relation to the workspace or current window. Valid values are:

- <Auto>

  The dialog window is positioned as designed.
- Center

  The dialog box is centered within the workspace.
- Center Mouse

  The dialog box appears under the mouse.

The default setting is <Auto>.

Generating Code

When you save a form, the Window Editor automatically generates:

- A form entity for the window; you can double-click on this entity in the Repository Explorer’s list view pane to begin editing it with the Window Editor
- A subclass using the window name and the appropriate subclass of the Window class: DataWindow, DialogWindow, DataDialog Window, or ShellWindow
- Other default source code that can be modified, including instance variables related to the controls on the form (code will be generated to declare an instance variable for each control object unless its Generate Code property is set to No)

Note that for an MDI application, the generated code automatically supports the ability to open the same window multiple times.

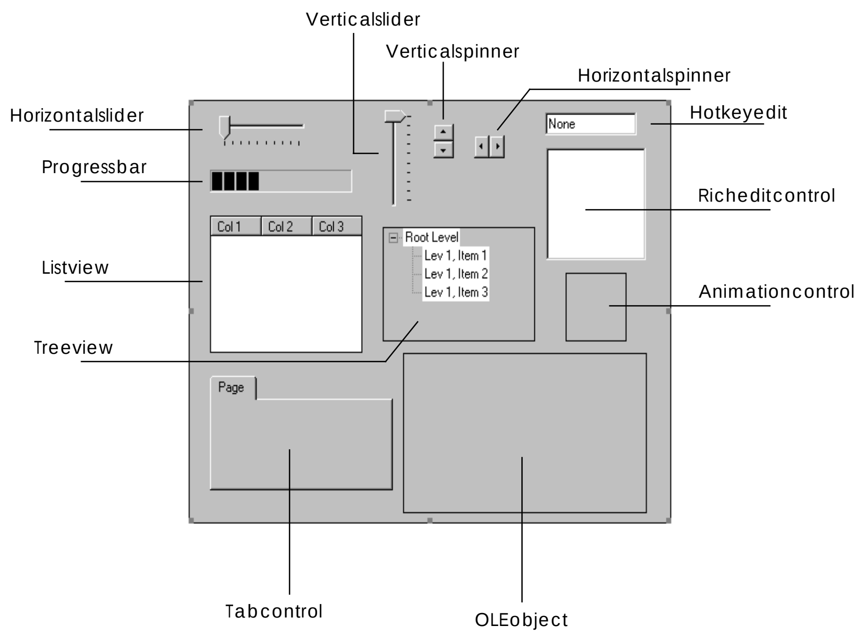
Also, the Init() methods for *all* generated forms take an additional parameter, <*uExtra*>. This allows you to pass additional data to a window or dialog. For an example using these extra parameters, see the OptionDlgs module in the Private Eye sample application (CAVO27\SAMPLES\WINSDK\PE). These same parameters are also passed to the PreInit() and PostInit() methods of these windows, if defined.

Tip: You can modify the generated code by changing the CAVOWED.INF and CAVOWED.TPL files. For detailed information about these files, refer to the *CA-Visual Objects* 2.7 *Software Development Kit*.

Control Types

Other key properties of a window are its *controls*, which the end user manipulates while interacting with the graphical user interface (GUI). Experienced users of CA-Visual Objects are already familiar with the following GUI control types: check box, combo box, list box, single-line edit control, multi-line edit control, push button, radio button, radio button group, group box, fixed text, fixed icon, fixed bitmap, vertical scroll bar, horizontal scroll bar, and sub-data window.

Newer GUI control types are the *Windows common controls*, shown below, which are actually specialized, predefined child windows:



Note: As mentioned earlier, there are new common controls in CA-Visual Objects 2.7, including ComboBoxEx control, date time picker, IP address, and month calendar. There is also a new CA-Visual Objects custom control, data list view. For detailed information, see the online help.

These controls are supported by CA-Visual Objects new classes, and they allow you to provide advanced I/O features for the end user. See Specifying Control Properties and Style Settings for more detailed information about each of the common controls, and refer to the online help system for information about the new common control classes.

Placing Controls on a Form

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

Using the 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 the easiest and fastest method, and it is known as *drag-and-drop*. Note that as you start to drag the mouse, a positioning icon in the form of cross hairs appears: **+**

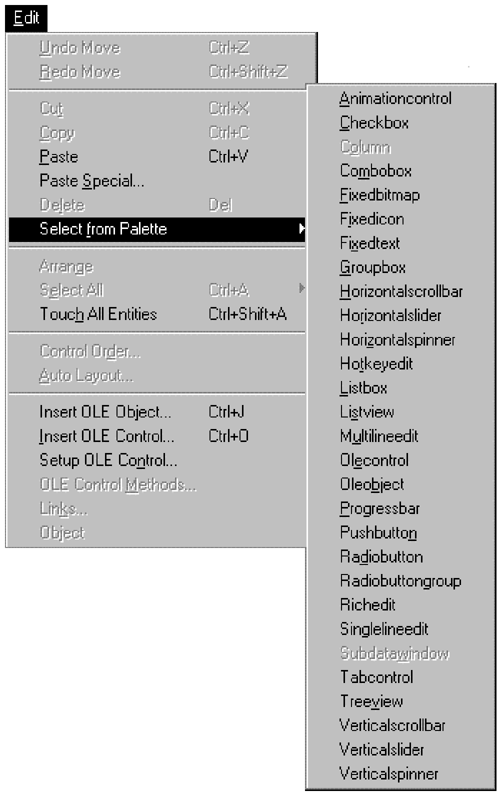
- Place the mouse pointer on an icon in the tool palette and press the left mouse button, this time releasing the button. Move the mouse pointer to the desired position in the form and then click the mouse again.

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

Tip: If you want to create multiple occurrences of a control on a form, hold the Shift key down while placing the control on the form. Release the Shift key when finished.

Using the Select from Palette Menu Commands

As an alternative to using the tool palette to place controls, you can use the Edit Select from Palette menu command and its pop-up menu:



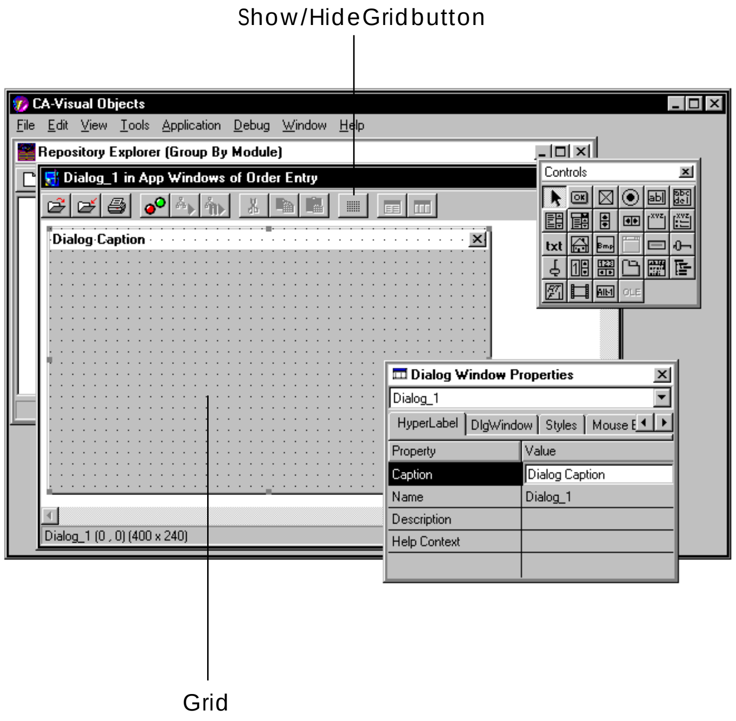
Note: Just as the Window Editor's tool palette has been updated to reflect the new CA-Visual Objects 2.7 controls, so too has the Edit menu. See the online help for detailed information about the corresponding Select from Palette menu commands for the ComboBoxEx, date time picker, IP address, month calendar, and data list view controls.

After choosing the menu command that corresponds to the type of control you want, move the mouse pointer to the desired position in the form and click the mouse to place the new control.

Using the Grid

When placing/arranging controls on a form, you might find it useful to use a *grid*. Turning on the grid allows you to quickly and easily align controls with accurate placement.

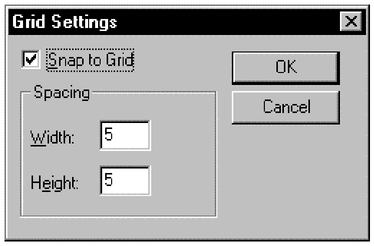
To enable the grid, click the Show/Hide Grid toolbar button:


Show/Hide Grid button
CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Dialog_1 in App Windows of Order Entry
Dialog Caption
Controls
Dialog Window Properties
Dialog_1
HyperLabel DlgWindow Styles Mouse E
Dialog_1 (0 , 0) (400 x 240)
Grid

Property	Value
Caption	Dialog Caption
Name	Dialog_1
Description	
Help Context	

This button acts as a toggle. You can click it again to hide the grid when you are finished using it.

Note that you can also customize the grid display. To do so, choose the Grid Settings command from the File menu. Selecting this command displays the following dialog box:



[✓] Snap to Grid

Spacing

Width: 5

Height: 5

Spacing

To override the default spacing between grid dots:

1. Enter different values in the Width and Height edit controls.

   The default setting for both is 5. The minimum width or height is 2, and the maximum is 12.

2. Select OK.

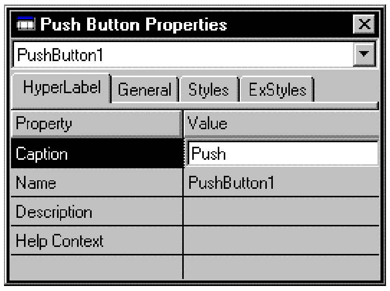
Snap to Grid

The Snap to Grid option, when selected, causes the controls to align or "snap" precisely to the grid dot settings as you place them within the form.

Note: Turning off the grid also turns off the Snap to Grid feature.

Specifying Control Properties and Style Settings

After you have placed a control on your form, the next step is to define its properties and style settings. Properties define how the control will behave. As mentioned earlier, the role of the Properties window changes depending on the currently selected item in the workspace. For example, the following Push Button Properties window is displayed when a push button control is selected in the current form:



PushButton1

Property	Value
Caption	Push
Name	PushButton1
Description	
Help Context	

The Window Editor will generate default settings for many of the property and style settings when you place the control on a form. These defaults are sufficient to save the form; however, there are other properties that you must define to make certain a control is fully operational at runtime.

Note: An object variable is associated with each control, unless the control is already associated with a field (such as a single-line edit). The control name appears as part of the object name in the Init() method for the variable.

Common Properties

Note: See the online help for detailed information about new common properties, such as Left, Right, and Use for Tooltip. Other common properties have been updated. For example, the Background Color property gives you immediate access to CA-Visual Objects standard Color dialog box for defining a control's background color. Additionally, all text controls are now data aware (see the AWARE.AEF application in the CAVO27\SAMPLES\CONTROLS\AWARE directory for examples). For detailed information about *all* control properties, refer to the online help.

Many of the control properties—such as Generate Code and Tab Stop—are common to *all* controls, and others—such as Caption and Inherit from Class—are common to all with the exception of the sub-data window control. Still others, such as Background Color, are common to a majority of the controls. Therefore, descriptions of these common properties in tab page order follow immediately. (Properties that are unique to a specific type of control are described later in this chapter.)

HyperLabel Tab

The HyperLabel tab contains the following options, common to all controls:

Caption

Enter the caption text that identifies a control in the form display. This property is available for all controls except sub-data windows, and is used differently, depending on the control type.

For a push button, for example, this is the label that is displayed on the button.

For a fixed icon, the caption is used to specify the icon file to associate with the control. Icon files are identified as resource entities in your application. The icon is assigned a name or number. You enter the resource name or its number (which must be preceded by a # symbol) as the caption.

Note: It is typical to have code similar to the following to define an icon resource:

```
DEFINE MY_ICON := 101
RESOURCE MY_ICON ICON myicon.ico
```

In this case, the icon resource is defined using a number, and you would enter **#101** as the caption because the Window Editor would not recognize the constant MY_ICON.

If you use an icon created with the Image Editor, you will use the name you gave the icon. To gain a better understanding of how icons are declared in CA-Visual Objects, you may want to take a look at the code generated for an icon.

For other controls, such as scroll bars, the caption text is not used; you can use the caption property to store information that can be accessed from your application. The default value is defined by the control type when you place it on the form (for example, the default caption for all push buttons is "Push"). This property is available for all controls except sub-data windows.

Tip: Push buttons and other controls sometimes feature a single underlined letter indicating how to select the control with the keyboard. To add this functionality to your controls, simply preface the letter that is to be underlined with an ampersand (&).

Note: The Caption property has a hierarchical nature like the Description property described below.

Name

Enter the name of the control. This property value is required for all controls and is 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 "PushButton1."

Every type of control has a Name property that is used in the generated source code to identify individual controls within the form. For push buttons, this name also determines the method, Window, or ReportQueue name that is invoked when the control is clicked. Therefore, control names must be unique within a single form design.

Note: For sub-data window controls only, this property can be found on the General tab of its Properties window.

Description

The Description property is available for all controls except sub-data windows. Enter the text that should appear in the status bar when the control has focus in the form display. If a control cannot have focus, the description will not be displayed in the status bar; instead, it may be accessed through the control's hyperlabel. Note that a control may use either its own descriptive text or that of an associated field specification or field. Therefore, the default value of <Auto> means that the system should use a description from a lower level of the hierarchy.

For example, assume that you already have a field specification named **X_CUSTNO** that has the cryptic description, **CUSTID**. Also assume that you already have a field called **ID** that has an equally cryptic description, **VENDNUM/CUSTNUM,** and is used for both vendor and customer numbers.

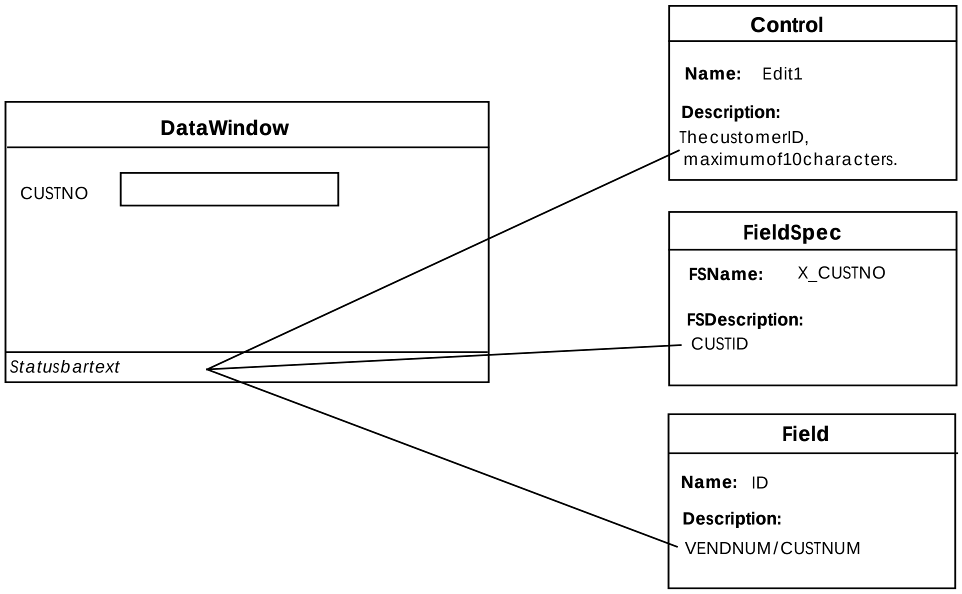
If you now create a data window for maintaining customer information that has a single-line edit control with the caption, **CUSTNO**, you have three choices for the status bar text:

1. The text entered in the Description value cell for the *edit control*—for example, **The customer ID, maximum of 10 characters.**
2. The text entered in the FS Description value cell (available in the DB Server Editor or Field Spec Editor) for the *field specification* (for example, **CUSTID**).
3. The text entered in the Description value cell (available in the DB Server Editor) for the *field* (for example, **VENDNUM/CUSTNUM**).

   (This last choice is poor, as the end user might confuse the field and enter a vendor ID.)

CA-Visual Objects searches sequentially through the hierarchy for a value for the Description property and associates the *first* definition it encounters with the control. That is, if it finds a description at the control level, that description is used in the status bar. If, instead, it finds the default <Auto> value, then the system searches for a definition at the *field specification* level, and so on. If the default <Auto> value is specified for all levels, then *no* description appears in the status bar.

The following diagram illustrates the hierarchical nature of the Description property:



See Specifying Control Properties and Style Settings in “Defining Data Servers and Field Specs” for more information about field and *field specification* properties.

Help Context

The Help Context property is available for all controls except sub-data windows. Enter a unique keyword for the control. This keyword will be used to identify the control in a context-sensitive help system. A help file can be specified using the Help File Name property for the form. See the Help Context and Help File Name properties in Specifying Window Properties earlier in this chapter for specifics.

Note: The Help Context property has a hierarchical nature like the Caption and Description properties.

General Tab

The General tab contains the following properties that are common to all controls with the exception of sub-data window controls. Other options that are specific to each control type are described later under the General tab for the corresponding control.

Inherit from Class

Select the Control class or subclass from which the specified window control will inherit its basic characteristics. For example, a value of <Auto> for a push button generates an instance of CA-Visual Objects PushButton class. If, however, you have a derived, specialized class (for example, OKButton), this class also appears in the Inherit from Class list box and you can use it instead.

The default value is <Auto>.

Context Menu

Enter or choose the name for a local pop-up menu to be accessed by right-clicking a control. Valid choices are <Auto>, EMPTYSHELLMENU, STANDARDSHELLMENU, and all classes derived from the Menu class that are in the current search path. The default value is <Auto>, which means no menu is used.

Generate Code

Specify whether certain source code will be generated for the control. This is a Yes / No option that you change by selecting a value from a drop-down list. The default value of Yes indicates that source code will be generated.

Changing this property to No can cut down on the resources needed to use the window, but you can only do this if you do not need access to the control in your application. For example, controls that are not data-aware (such as fixed text and fixed icon) are seldom referenced in your application source code, so you can change the Generate Code property to No to prevent the Window Editor from generating unnecessary code.

Note: The code to display a control is always generated, regardless of the status of this property.

Width	Enter a value for the horizontal size of the specified window control.
Height	Enter a value for the vertical size of the specified window control.

Note: See the online help system for the default Width and Height settings for each specific control type. Also, refer to the *CA-Visual Objects 2.7 Software Development Kit* for information about changing the default settings for Width and Height for any control type, as well as any other property.

Additionally, the following two options are common to the push button, check box, radio button, radio button group, list box, group box, fixed text, single-line edit, multi-line edit, and rich edit controls:

Font	Enter or choose the name of the font used to display text directly associated with the control (for example, a push button label or an item in a list box). The available fonts are the same as those for the window Font property (see Specifying Window Properties).
Input Method Editor	If set to #ON, allows foreign language users to edit fields using an independent editor. Valid choices are: <Auto>, #ON, and #OFF.

See the Microsoft *Win32 Software Development Kit* for more information.

The following option, Text Color, is common to the check box, combo box, list box, radio button, radio button group, group box, fixed text, single-line edit, multi-line edit, and rich edit controls:

Visible	If Yes, specifies that the control be displayed. If set to No, the control is hidden from view. The default value is Yes.
Text Color	Enter or choose the color used to display text directly associated with the control. The default value is <Auto>, which means the currently selected system text color is used. If you click on the right side of this cell, you can select a color from a list of choices.

Note: Text for push buttons and some of the new common controls—such as list views, tree views, tab controls, and hot key edit controls—use the default system text color.

Background Color	Choose the color to be used for shading the background of the control display. The default value is <Auto>, which means the currently selected system background color is used.

Note: This option is common to all window controls *except* push button, horizontal scroll bar, vertical scroll bar, sub-data window, and OLE object controls.

Tip: In the case of radio buttons, check boxes, group boxes, and radio button group boxes, only the character area is shaded. Therefore, for aesthetic purposes you may want to use spaces to pad the character area of each radio button or check box within a group.

Styles Tab

The Styles tab contains the following options that are common to all controls, with the exception of the sub-data window control:

Disabled	If True, creates a control that is initially disabled, or grayed. The default setting is False.
Tab Stop	If True (the default option), allows the user to press the Tab key to move through any number of controls defined with this option. **Note:** By default, the cursor tabbing order follows a left-to-right, top-to-bottom progression, based on the position on the controls on the form. You can, however, change the tab order using the Control Order dialog box. (See Changing Tab Order by Reordering Controls later in this chapter for more information.)
Group	If True, indicates the first control in a group of controls that can be accessed using the Direction keys. Successive controls belong to the same group until another control is defined with the Group option. That is, the next control specified with the Group option defines the end of the previous style group and the beginning of a new style group. The default setting is False.

ExStyles Tab

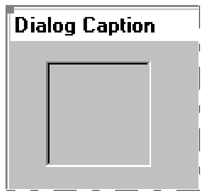
The ExStyles tab contains the following extended window options that are common to all controls:

Right-To-Left Reading	If True, specifies that form text be displayed using left-to-right reading order properties. The default setting is False.
No Parent Notify	If True, specifies that a child window not send the WM_PARENTNOTIFY message to its parent window when it is created or destroyed. The default setting is False.

Accept Files	Specifies that a window or control accept drag-and-drop files, if True. That is, in the case of a drop event (WM_DROPFILES), the Drop method of the control's parent window is called and a DragEvent object containing the file names is passed as a parameter. (See the DragDropClient:Drop access / assign topic in the online help system for additional information.) The default setting is False.
Transparent	If True, specifies that a window or control be transparent (that is, the current window should not obscure any windows beneath it). The default setting is False.

Client Edge

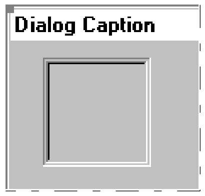
Specifies that a window or a control have a border with a "sunken" edge, if True. For example:



The default setting is False.

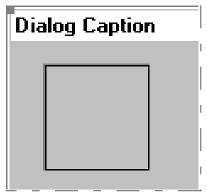
Note: Rich text controls and hot key edit controls will always have Client Edge set to True.

Note: If you want a control to have a "deep sunken" edge, set both Client Edge and Static Edge to True. For example:



Static Edge

If True, specifies that a window or a control have a three-dimensional border style to indicate that its items do not accept user input. For example:



The default setting is False.

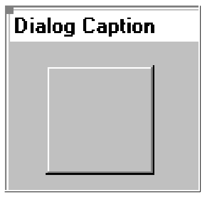
Note: Progress bars will always have Static Edge set to True.

Note: In the case of sub-data window controls, the Client Edge and Static Edge properties are applicable in form view only.

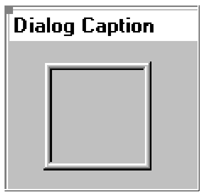
Modal Frame

If True, creates a window that has a double border. This window can, optionally, be created with a title bar by specifying the WS_CAPTION style in the *dwStyle* parameter. The default setting is False.

Note: If you want a control to appear raised on the form, set both Static Edge and Modal Frame to True. For example:



Note: If you want a raised border around a control, set Client Edge, Static Edge, and Modal Frame all to True. For example:



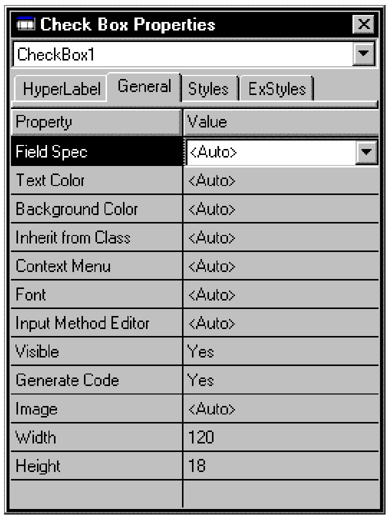
Check Box Properties and Style Settings

A standard check box is used to display a logical TRUE / FALSE status for a field or variable, as well as allow the user to change the status. You can bind a check box to a logical field in a database. When a user clicks on a check box, the Checked and Value accesses for the related object are changed. You can also change the state using the Checked or Value assigns in your source code.

If you wish, you can write custom code to respond to events for a specific check box using the Window:ButtonClick() or Window:ButtonDoubleClick() methods; see the online help system for details. You can also create a 3-state check box to display a TRUE / FALSE / UNDETERMINED status.

General Tab

The General tab, shown below, allows you to specify a check box's general properties, including physical appearance and class inheritance:



Property	Value
Field Spec	<Auto>
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Input Method Editor	<Auto>
Visible	Yes
Generate Code	Yes
Image	<Auto>
Width	120
Height	18

Note: Refer to the online help for detailed information about new and updated check box properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Field Spec

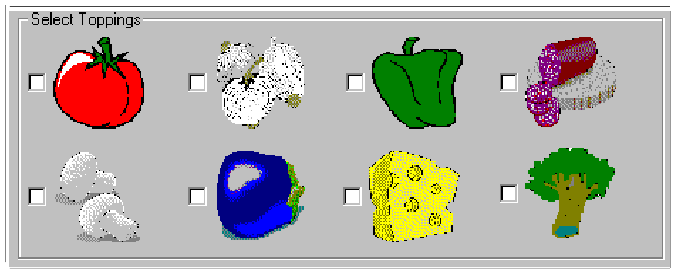
Enter or choose a field specification to associate with the control.

When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications for more information.

This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.

Image

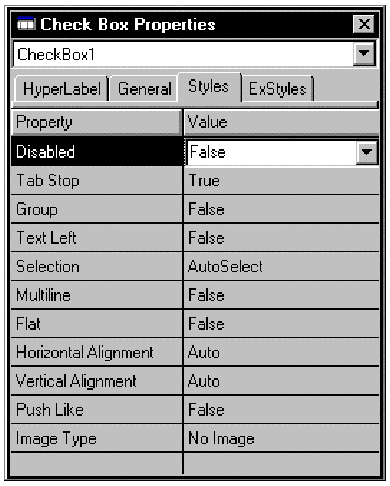
Optionally, assign a bitmap image or icon object to the check box. The image displays at runtime. Below are examples from the PIZZA.AEF application in the CAVO27\SAMPLES directory:



Note: To display an image object with a check box, the Image Type property *must* also be set. See the related properties, Image Type and Push Like, described below under Styles Tab.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your check box control:



Property	Value
Disabled	False
Tab Stop	True
Group	False
Text Left	False
Selection	AutoSelect
Multiline	False
Flat	False
Horizontal Alignment	Auto
Vertical Alignment	Auto
Push Like	False
Image Type	No Image

Note: Refer to the online help for detailed information about new and updated check box properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Text Left

If True, creates a check box with its caption displayed to the left. If False, the caption is displayed to the right of the check box. This is a valid option for all check box styles.

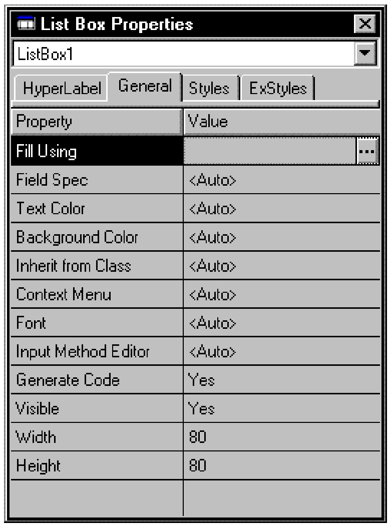
Selection	This property is used to define a check box's state. Valid options are: ■ AutoSelect If selected, creates a check box as an option button that allows the user to automatically toggle between checked and unchecked states. Unlike radio buttons, any number of check boxes can be selected at the same time. AutoSelect is the default setting for the Check Box Styles option. ■ No AutoSelect If selected, creates a check box that will not accept any input from the user. That is, it will not retain the checked status when clicked. ■ 3 State If selected, creates a check box that can be dimmed, or *grayed*, as well as checked and unchecked. The grayed state indicates that the state of the check box is undetermined. The state can be changed only by code events, not by the user clicking the check box. If the check box is bound to a logical field in a server, the third state will not be reflected in the database. ■ AutoSelect 3 State If selected, creates a check box identical to the 3 State option above, except that the state of the box changes when selected. The status of the check box cycles through the checked, grayed, and normal states.
Multiline	If True, button text is wrapped, allowing multiple lines of text. The default setting is False.
Flat	If True, the button appears flat (non-3D). The default setting is False.
Horizontal Alignment	Specifies the horizontal alignment of the button text. Valid values are: Auto, Left, Center, and Right. The default is Auto.
Vertical Alignment	Specifies the vertical alignment of the button text. Valid values are: Auto, Top, Center, and Bottom. The default is Auto.
Push Like	If True, the check box appears like a push button. The default setting is False.
Image Type	Specifies the type of image button. Valid values are: No Image (text only), Icon, or Bitmap. The default is No Image.

List Box Properties and Style Settings

A list box allows users to select from a list of choices. You can bind a list box to a field in a database. When a user selects a different choice, the TextValue and Value accesses for the related object are changed. You can write custom code to respond to events for a specific list box object using the Window:ListBoxSelect() or Window:ListBoxClick() methods; see the online help system for details.

General Tab

The General tab, shown below, allows you to specify a list box's general properties, including physical appearance and class inheritance:



Property	Value
Fill Using	
Field Spec	<Auto>
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Input Method Editor	<Auto>
Generate Code	Yes
Visible	Yes
Width	80
Height	80

Note: Refer to the online help for detailed information about new and updated list box properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Fill Using

This property allows you to define the items in a list box using the contents of any array expression, data server, or customized method.

See Defining Arrays for List and Combo Boxes later in this chapter for detailed information about this option and the Fill Using dialog box that appears.

Field Spec

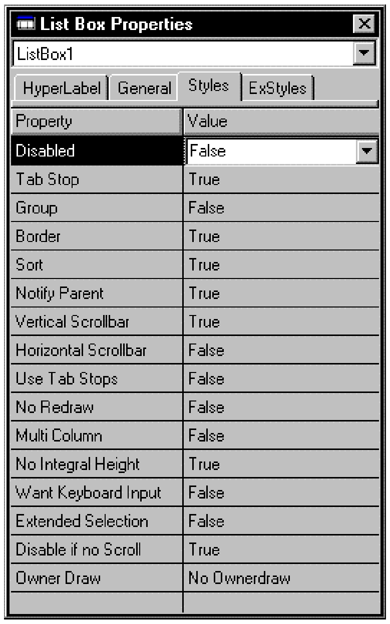
Enter or choose a field specification to associate with the control.

When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information about these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications for more information.

This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your list box control:



Property	Value
Disabled	False
Tab Stop	True
Group	False
Border	True
Sort	True
Notify Parent	True
Vertical Scrollbar	True
Horizontal Scrollbar	False
Use Tab Stops	False
No Redraw	False
Multi Column	False
No Integral Height	True
Want Keyboard Input	False
Extended Selection	False
Disable if no Scroll	True
Owner Draw	No Ownerdraw

Note: Refer to the online help for detailed information about new and updated list box properties.

The Styles tab contains the following properties that are specific to this type of control:

Border

Creates a border around the list box (if True). True is the default setting for the Border option.

Sort

Sorts list box items alphabetically (if True). True is the default setting for the Sort option.

Notify Parent	If True, notifies the owner window with an input message whenever the user selects a list box item. True is the default setting for the Notify option.
Vertical Scrollbar	If True, adds a vertical scroll bar to the list box if the number of items exceeds the border of the list box. True is the default setting.
Horizontal Scrollbar	Adds a horizontal scroll bar to the list box if the width of the items exceeds the list box's border, if True.
Use Tab Stops	If True, enables a list box to recognize and expand tab characters when items are added. **Note:** Windows default tab positions are 32 dialog box units, which are computed based on the height and width of the current system font. One horizontal dialog box unit is equal to one-fourth of the current dialog box base width unit.
No Redraw	Specifies that the list box should not be redrawn when changes are made (if True).
Multi Column	If True, defines the list box as multi-columnar.
No Integral Height	If True, specifies that the size of the list box remain exactly the same size as when it was created. Normally, Windows resizes a list box after being filled, so that it does not display partial items.
Want Keyboard Input	If True, specifies that the list box's owner window receive notification messages whenever the list box has focus and the user enters or selects an item(s) from the list box.
Extended Selection	If True, allows the user to choose multiple list box items by highlighting them using the mouse and the Shift key or special key combinations.
Disable If No Scroll	If True, displays the list box's vertical scroll bar as disabled when there are not enough items in the list box to scroll. Otherwise, the scroll bar is hidden.
Owner Draw	Specifies whether or not the owner window is responsible for drawing the contents of the list box in the child window. Valid choices are: ■ No Ownerdraw No Ownerdraw is the default for the Owner Draw option, meaning the owner window is not responsible for drawing the contents of the list box. ■ Fixed Specifies that the owner window is responsible for drawing the contents of the list box, ensuring that the items in the list box are the same height.

- Variable

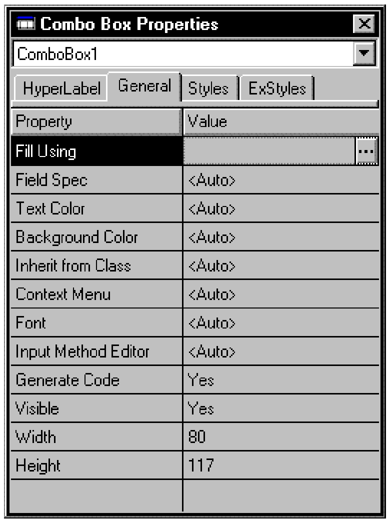
  Specifies that the owner window is responsible for drawing the contents of the list box and that the items in the list box vary in height.

Combo Box Properties and Style Settings

A combo box allows users to select from a list of choices or type in their own choice. A combo box can be bound to a field in a database. When a user changes the displayed choice, the TextValue and Value accesses for the related object are changed. You can write custom code to respond to events for a specific combo box object using the Window:ListBoxSelect() or Window:ListBoxClick() methods; see the online help system
for details.

General Tab

The General tab, shown below, allows you to specify a combo box's general properties, including physical appearance and class inheritance:



Property	Value
Fill Using	
Field Spec	<Auto>
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Input Method Editor	<Auto>
Generate Code	Yes
Visible	Yes
Width	80
Height	117

Note: Refer to the online help for detailed information about new and updated combo box properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Fill Using

This property allows you to define the items in a combo box using the contents of any array expression, data server, or customized method.

See Defining Arrays for List and Combo Boxes later in this chapter for detailed information about this option and the Fill Using dialog box that appears.

Field Spec

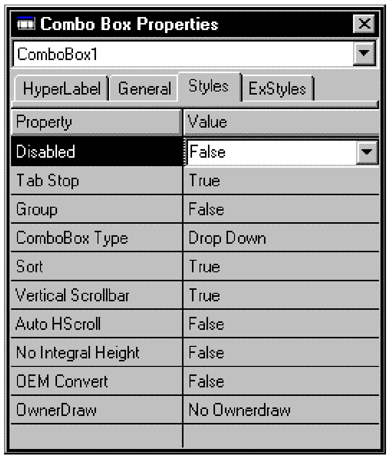
Enter or choose a field specification to associate with the control.

When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications for more information.

This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your combo box control:



Property	Value
Disabled	False
Tab Stop	True
Group	False
ComboBox Type	Drop Down
Sort	True
Vertical Scrollbar	True
Auto HScroll	False
No Integral Height	False
OEM Convert	False
OwnerDraw	No Ownerdraw

Note: Refer to the online help for detailed information about new and updated combo box properties.

The Styles tab contains the following properties that are specific to this type of control:

ComboBox Type	This property is used to define a combo box's style. Valid options are:

- Simple

  If selected, creates a simple combo box that displays its list box component at all times with the current item displayed in its edit control component.

- Drop Down

  If selected, creates a drop-down combo box that displays only the current item in a single-line edit control but allows the user to display its list box by clicking on a drop-down arrow button to the right of its edit control.

  This option also allows the user to start typing in text which, if matched by the system, is highlighted in the list box component of the control. The user then clicks on the highlighted item to complete the selection.

- Drop Down List

  If selected, creates a combo box with a drop-down list box that is similar to the Drop Down option except that the user cannot type in the edit control.

Drop Down is the default setting.

Sort	Sorts the items in the combo box alphabetically, if True.
Vertical Scrollbar	If True, adds a vertical scroll bar to the combo box.
Auto HScroll	If True, any text in the edit control component of the combo box is automatically scrolled to the right as the user types a character at the end of the line. Otherwise, only text that fits within the physical boundaries of the edit control is allowed.
No Integral Height	If True, specifies that the size of the combo box remain the same as when it was created. Normally, Windows resizes a combo box after being filled so that it does not display partial items.
OEM Convert	If True, converts any user-entered text in the edit control from the Windows character set to the OEM character set and back again.

Tip: The OEM Convert style option is especially useful for any type of edit control that contains a file name, because it ensures that a correct conversion from Windows or Windows NT to the OEM character set is possible.

OwnerDraw	Specifies whether or not the owner window is responsible for drawing the contents of the combo box in the child window.

Valid choices are:

- No Ownerdraw

  No Ownerdraw is the default for the Owner Draw option, meaning the owner window is not responsible for drawing the contents of the combo box.

- Fixed

  Specifies that the owner window is responsible for drawing the contents of the combo box, ensuring that the items in the combo box are the same height.

- Variable

  Specifies that the owner window is responsible for drawing the contents of the combo box and that the items in the combo box vary in height.

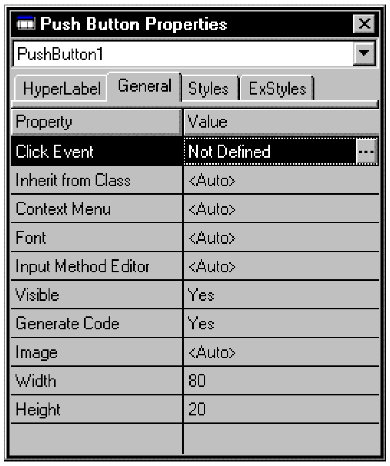
Push Button Properties and Style Settings

A push button (also called a *command button*) allows users to interactively trigger actions by activating the button. Either a key press or mouse click can be used to initiate a window, report, or other related source code that you specify using the push buttons Click Event property.

You can also write custom code to respond to events for a specific push button using the Window:ButtonClick() or Window:ButtonDoubleClick() methods. See Push Button Controls and Actions later in this chapter for more information. Also, refer to the online help system for complete information about the Window:ButtonClick() or Window:ButtonDoubleClick() methods.

General Tab

The General tab, shown below, allows you to specify a push button's general properties, including physical appearance and class inheritance:



Property	Value
Click Event	Not Defined
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Input Method Editor	<Auto>
Visible	Yes
Generate Code	Yes
Image	<Auto>
Width	80
Height	20

Note: Refer to the online help for detailed information about new and updated push button properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Click Event

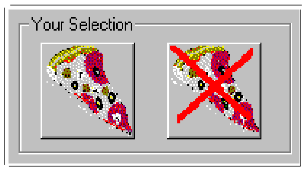
This property is available for push buttons only and allows you to define what happens when the button is clicked. Click the Ellipsis button to launch the Source Code Editor and enter the source code for a method that will be executed when the button is clicked.

The default value is Not Defined.

See Push Button Controls and Actions later in this chapter for more information.

Image

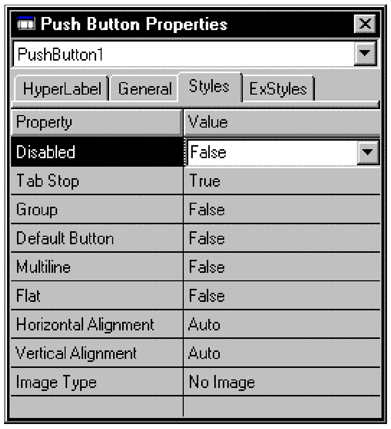
Optionally, assign a bitmap image or icon object to the push button. The image displays at runtime. Below are examples from the PIZZA.AEF application in the CAVO27\SAMPLES directory:



See the related property, Image Type, described below under Styles Tab.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your push button control:



Push Button Properties

PushButton1

Property	Value
Disabled	False
Tab Stop	True
Group	False
Default Button	False
Multiline	False
Flat	False
Horizontal Alignment	Auto
Vertical Alignment	Auto
Image Type	No Image

Note: Refer to the online help for detailed information about new and updated push button properties.

The Styles tab contains the following properties that are specific to this type of control:

Default Button	If True, creates a push button that appears as the default option for the window. The default push button has focus when the window is initially displayed and continues to have focus unless the user highlights another push button (for example, by tabbing to it). Therefore, the user can normally select the default push button by simply pressing the Enter key. The push button also appears with a heavy black border, indicating that it is the default option. If False, creates a push button that does not appear as the default option for the window. False is the default setting.
Multiline	If True, button text is wrapped, allowing multiple lines of text. The default setting is False.
Flat	If True, the button appears flat (non-3D). The default setting is False.
Horizontal Alignment	Specifies the horizontal alignment of the button text. Valid values are: Auto, Left, Center, and Right. The default is Auto.
Vertical Alignment	Specifies the vertical alignment of the button text. Valid values are: Auto, Top, Center, and Bottom. The default is Auto.
Image Type	Specifies the type of image button. Valid values are: No Image (text only), Icon, or Bitmap. The default is No Image.

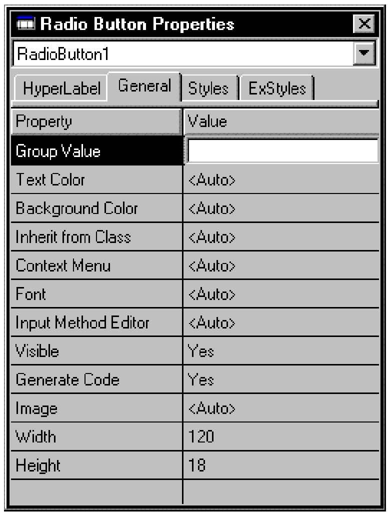
Radio Button Properties and Style Settings

A radio button is used to display a logical TRUE/FALSE status, as well as allow the user to change the status. Radio buttons are generally grouped into a radio button group, which allows one of several mutually exclusive choices to be selected. When a user clicks on a radio button, the Pressed and Value accesses for the related object are changed.

You can also change the state using the Pressed or Value assigns in your source code. Additionally, you can write custom code to respond to events for a specific radio button using the Window:ButtonClick() or Window:ButtonDoubleClick() methods. (See the online help system for complete information about these methods.)

General Tab

The General tab, shown below, allows you to specify a radio button's general properties, including physical appearance and class inheritance:



Property	Value
Group Value	
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Input Method Editor	<Auto>
Visible	Yes
Generate Code	Yes
Image	<Auto>
Width	120
Height	18

Note: Refer to the online help for detailed information about new and updated radio button properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Group Value

This property is available for radio buttons only. In it, you enter the value to be returned to the associated radio button group when the radio button is selected. The value can be numeric or string.

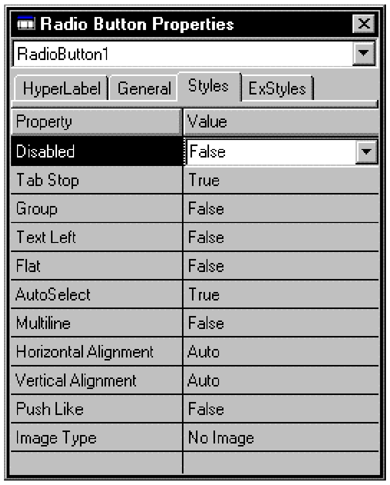
A radio button used by itself returns its actual value (for example, T or F, 1 or 0, or Yes or No). On the other hand, if you create a group box of five radio buttons, the value 1 indicates that RadioButton1 is selected, 2 that RadioButton2 is selected, and so on.

Image

Optionally, assign a bitmap image or icon object to the radio button. The image displays at runtime. See the related properties, Image Type and Push Like, described below under Styles Tab.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your radio button control:



Property	Value
Disabled	False
Tab Stop	True
Group	False
Text Left	False
Flat	False
AutoSelect	True
Multiline	False
Horizontal Alignment	Auto
Vertical Alignment	Auto
Push Like	False
Image Type	No Image

Note: Refer to the online help for detailed information about new and updated radio button properties.

The Styles tab contains the following properties that are specific to this type of control:

Text Left

If True, creates a radio button with its caption displayed to the left. If False, the caption is displayed to the right of the check box.

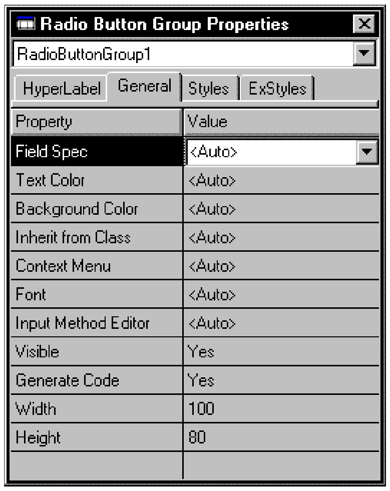
AutoSelect	If True, creates an option radio button that automatically highlights itself and clears any other buttons in the same group of related but mutually exclusive choices. True is the default setting for the AutoSelect option. If False, creates a radio button that will not accept any input. That is, it will not retain the selected status when clicked.
Multiline	If True, button text is wrapped, allowing multiple lines of text. The default setting is False.
Flat	If True, the button appears flat (non-3D). The default setting is False.
Horizontal Alignment	Specifies the horizontal alignment of the button text. Valid values are: Auto, Left, Center, and Right. The default is Auto.
Vertical Alignment	Specifies the vertical alignment of the button text. Valid values are: Auto, Top, Center, and Bottom. The default is Auto.
Push Like	If True, the radio button appears like a push button. The default setting is False.
Image Type	Specifies the type of image button. Valid values are: No Image (text only), Icon, or Bitmap. The default is No Image.

Radio Button Group Properties and Style Settings

A radio button group is used to allow one selection out of a set of related, but mutually exclusive, choices. Each radio button in the group is assigned a Group Value as one of its properties. Then, the Value access for the radio button group indicates which of the button values has been selected.

General Tab

The General tab, shown below, allows you to specify a radio button's general properties, including physical appearance and class inheritance:


Radio Button Group Properties

RadioButtonGroup1

HyperLabel | General | Styles | ExStyles

Property	Value
Field Spec	<Auto>
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Input Method Editor	<Auto>
Visible	Yes
Generate Code	Yes
Width	100
Height	80

Note: Refer to the online help for detailed information about new and updated radio button group properties.

All of these properties have been described previously under Common Properties, with the exception of the following property that is specific to this type of control:

Field Spec

Enter or choose a field specification to associate with the control.

When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications for more information.

This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.

Styles Tab

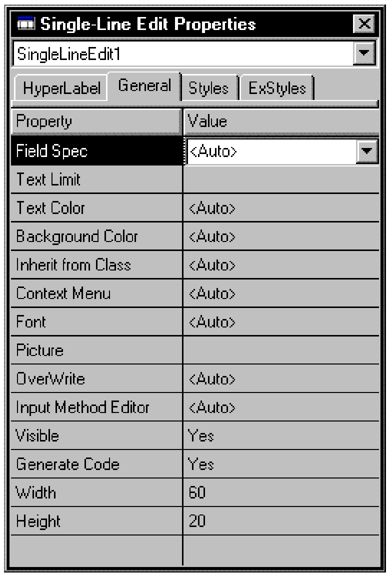
All of the style settings for a radio button group have been described previously under Common Properties.

Single-Line Edit Properties and Style Settings

A single-line edit control is used to display and modify text on a single line. This type of control can be bound to a field in a database. Its efficient use of space on a data window makes this a very useful edit control.

General Tab

The General tab, shown below, allows you to specify a single-line edit control's general properties, including physical appearance and class inheritance:


Single-Line Edit Properties

SingleLineEdit1

HyperLabel | General | Styles | ExStyles

Property	Value
Field Spec	<Auto>
Text Limit	
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Picture	
OverWrite	<Auto>
Input Method Editor	<Auto>
Visible	Yes
Generate Code	Yes
Width	60
Height	20

Note: In this version of CA-Visual Objects, single-line edit controls have two new properties, Focus Select and Scroll Mode. Refer to the online help for detailed information about these properties, as well as other updated properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Field Spec

Enter or choose a field specification to associate with the control.

When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications for more information.

This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.

Text Limit	Enter the maximum number of characters an end user may enter in the single-line edit control.
Picture	Optionally, enter a picture string for the single-line edit control. If there is a field specification with a picture string already associated with the control, the SingleLineEdit:Picture access/assign overwrites the field specification's picture string.

OverWrite

If the Picture property is defined for the single-line edit control, select a constant value for SingleLineEdit:Picture. Valid values are:

- OVERWRITE_NEVER

  If selected, specifies that the single-line edit control always be in INSERT mode.
- OVERWRITE_ONKEY

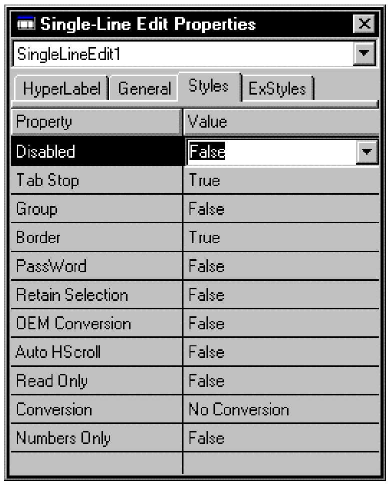
  If selected, specifies that the mode depends on the state of the INSERT key.
- OVERWRITE_ALWAYS

  If selected, specifies that the single-line edit control always be in OVERWRITE mode.
- <Auto>

  If selected, specifies that the single-line edit will use its inherited mode. By default, this will be OVERWRITE_NEVER.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your single-line edit control:


Single-Line Edit Properties

SingleLineEdit1

Property	Value
Disabled	False
Tab Stop	True
Group	False
Border	True
PassWord	False
Retain Selection	False
OEM Conversion	False
Auto HScroll	False
Read Only	False
Conversion	No Conversion
Numbers Only	False

Note: The single-line edit control does not have an alignment property as does the multi-line edit control; by default it has left alignment. If you want a control with center or right alignment, use the multi-line edit control with its height set appropriately.

Note: Refer to the online help for detailed information about new and updated single-line edit control properties.

The Styles tab contains the following properties that are specific to this type of control:

Border

Creates a single-line edit control with a border (if True). True is the default setting for the Border option.

Password

If True, displays all characters as asterisks (*) as they are entered in the single-line edit control.

Retain Selection

If False, the highlighting will be removed when the edit control loses focus (for example, if you Tab to another control). When the edit control regains focus, the previous selection will not be highlighted. False is the default choice.

If True, the highlighted or selected text will remain highlighted when you Tab to another control. In this fashion, any edit control you have edited stays marked (until you move to another record in a database or close the window).

OEM Conversion

If True, converts any user-entered text in the single-line edit control from the Windows character set to the OEM character set and back again.

Auto HScroll

If True, any text in the edit control is automatically scrolled 10 characters to the right when the user reaches the end of the line while typing. True is the default setting for the Auto HScroll option.

Tip: If you want a horizontal scroll bar to assist in scrolling the text, then create a multi-line edit control with the Horz. Scroll style setting checked. Adjust the height of the multi-line edit control to display the single line of text.

Read Only

If True, this option stops the user from entering/editing text in the single-line edit control.

Conversion

Specifies whether or not characters are converted to uppercase or lowercase. Valid choices are:

- No Conversion

  If No Conversion is selected (the default option), characters are not converted.

- Uppercase

  If Uppercase is selected, all characters are changed to uppercase as they are entered in the single-line edit control.

- Lowercase

  If Lowercase is selected, all characters are changed to lowercase as they are entered in the single-line edit control.

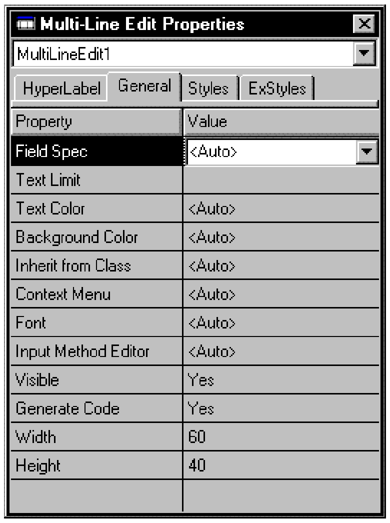
Numbers Only	If True, allows numerical input only. The default setting is False.

Multi-Line Edit Properties and Style Settings

A multi-line edit control is used to display and modify text on one or more lines. You can use the Home, End, or Direction keys to navigate within this control. This type of control can be bound to a field in a database. The wide variety of style settings makes this a very versatile control. In fact, you can set its height to simulate that of a single-line edit control and make use of its additional style settings (such as the Alignment choice and Horz. Scroll).

General Tab

The General tab, shown below, allows you to specify a multi-line edit control's general properties, including physical appearance and class inheritance:



Property	Value
Field Spec	<Auto>
Text Limit	
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Input Method Editor	<Auto>
Visible	Yes
Generate Code	Yes
Width	60
Height	40

Note: Refer to the online help for detailed information about new and updated multi-line edit control properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Field Spec

Enter or choose a field specification to associate with the control.

When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications for more information.

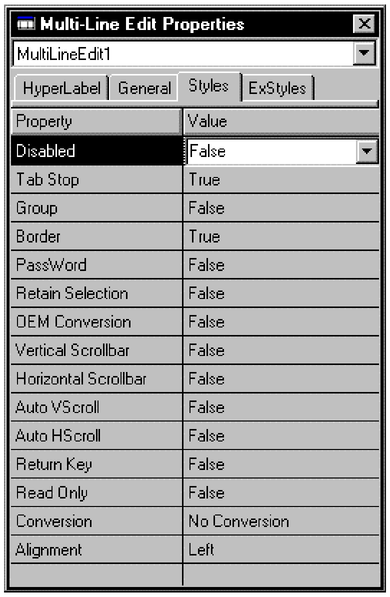
This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.

Text Limit

Enter the maximum number of characters the end user may enter in the multi-line edit control. The maximum is 65,535.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your multi-line edit control:



Property	Value
Disabled	False
Tab Stop	True
Group	False
Border	True
PassWord	False
Retain Selection	False
OEM Conversion	False
Vertical Scrollbar	False
Horizontal Scrollbar	False
Auto VScroll	False
Auto HScroll	False
Return Key	False
Read Only	False
Conversion	No Conversion
Alignment	Left

Note: Refer to the online help for detailed information about new and updated multi-line edit control properties.

The Styles tab contains the following properties that are specific to this type of control:

Border

Creates a multi-line edit control with a border (if True). True is the default setting for the Border option.

Password	If True, displays all characters as asterisks (*) as they are entered in the multi-line edit control.
Retain Selection	If False, the highlighting will be removed when the edit control loses focus (for example, if you Tab to another control). When the edit control regains focus, the previous selection will not be highlighted. False is the default choice. If True, the highlighted or selected text will remain highlighted when you Tab to another control. In this fashion, any edit control you have edited stays marked (until you move to another record in a database or close the window).
OEM Conversion	If True, converts any user-entered text in the multi-line edit control from the Windows character set to the OEM character set and back again.
Vertical Scrollbar	Creates a multi-line edit control with a vertical scroll bar (if True).
Horizontal Scrollbar	Creates a multi-line edit control with a horizontal scroll bar, where all the text is on the first line. In effect, this can be used to create a single-line edit control with a horizontal scroll bar.
Auto VScroll	If True, text is automatically scrolled up or down one line when the cursor reaches the beginning or end of a line.
Auto HScroll	If True, any text in the multi-line edit control is automatically scrolled 10 characters to the right when the user reaches the end of the line while typing.
Return Key	If True, the Enter key acts as a return key, moving the cursor to the next line of the multi-line edit control. If False, the Enter key activates the default push button for the window and has no effect on the edit control.
Read Only	If True, this option stops the user from entering/editing text in the multi-line edit control.
Conversion	Specifies whether or not characters are converted to uppercase or lowercase. Valid choices are: ■ No Conversion If No Conversion is selected (the default option), characters are not converted. ■ Uppercase If Uppercase is selected, all characters are changed to uppercase as they are entered in the multi-line edit control. ■ Lowercase If Lowercase is selected, all characters are changed to lowercase as they are entered in the multi-line edit control.

Alignment

This property is used to defined how the text is to be aligned. Valid options are:

- Left

  If selected, left-justifies text in the multi-line edit control. Left is the default setting for the Alignment option.

- Center

  If selected, centers text in the multi-line edit control.

- Right

  If selected, right-justifies text in the multi-line edit control.

Group Box Properties and Style Settings

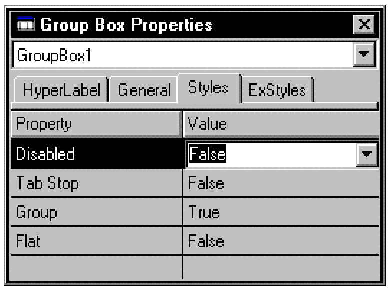
A group box is a passive single-line box that visually groups other controls on a form. The caption appears along the top border. Unlike a radio button group, a group box does not have application within your source code.

General Tab

All of the general properties for a group box have been described previously under Common Properties.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your group box control:



Property	Value
Disabled	False
Tab Stop	False
Group	True
Flat	False

Note: Refer to the online help for detailed information about new and updated group box properties.

All of these properties have been described previously under Common Properties, with the exception of the following property that is specific to this type of control:

Flat

If True, the group box appears flat (non-3D). The default setting is False.

Fixed Icon Properties and Style Settings

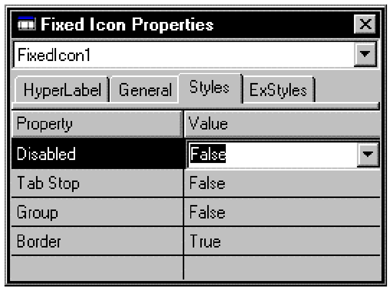
A fixed icon object provides a static display of an image on a form. The caption is used to specify the name or number of the icon resource to be displayed.

General Tab

All of the general properties for a fixed icon control have been described previously under Common Properties.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your fixed icon control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
Border	True

Note: Refer to the online help for detailed information about new and updated fixed icon properties.

All of these properties have been described previously under Common Properties, with the exception of the following property that is specific to this type of control:

Border	Creates a border around the fixed icon control, if True. True is the default setting for the Border option.

Fixed Text Properties and Style Settings

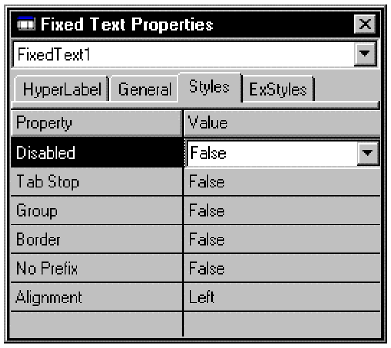
A fixed text object provides a static display of text on a form. You can modify the text to be displayed by using the TextValue assign in your source code.

General Tab

All of the general properties for fixed text have been described previously under Common Properties.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your fixed text control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
Border	False
No Prefix	False
Alignment	Left

Note: Refer to the online help for detailed information about new and updated fixed text properties.

The Styles tab contains the following properties that are specific to this type of control:

Border

If True, creates a border around the text. The default setting for the Border option is False.

No Prefix

If True, allows the use of ampersands (&) as part of the fixed text caption. If False, an ampersand in a caption text string is used to indicate that the next character is to be interpreted as the underlined mnemonic character used to activate the control with the keyboard.

Tip: If you want to use *both* ampersands and underlined mnemonic characters, set No Prefix to False and use two ampersands (&&) where you want the ampersand character to appear. For example, "&Visual &&Objects" becomes Visual &Objects at runtime.

Alignment

This property is used to defined how the text is to be aligned. Valid options are:

- Simple

  If selected, left-justifies the fixed text and displays it on a single line. The line of fixed text is not shortened or altered in any way.

- Left No Wrap

  If selected, left-justifies the control's fixed text *without* wrapping. Text that exceeds the width of the control is clipped.

- Left

  If selected, left-justifies and *wraps* the fixed text to fit within the physical boundaries of the control. Wrapping means that any text extending beyond the end of the line—that is, exceeds the horizontal size of the control—is brought over to the beginning of the next line.

  Left is the default setting for the Alignment option.

- Center

  If selected, centers and wraps the fixed text within the control's boundaries.

- Right

  If selected, right-justifies and wraps the fixed text within the control's boundaries.

Fixed Bitmap Properties and Style Settings

A fixed bitmap object provides a static display of an image on a form. The caption is used to specify the name or number of the bitmap resource to be displayed.

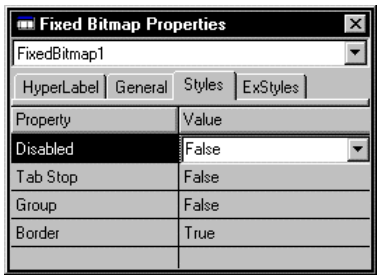
General Tab

All of the general properties for a fixed bitmap control have been described previously under Common Properties.

Note: Refer to the online help for detailed information about new and updated fixed bitmap properties.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your fixed bitmap control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
Border	True

Border — Creates a border around the fixed bitmap control, if True. True is the default setting for the Border option.

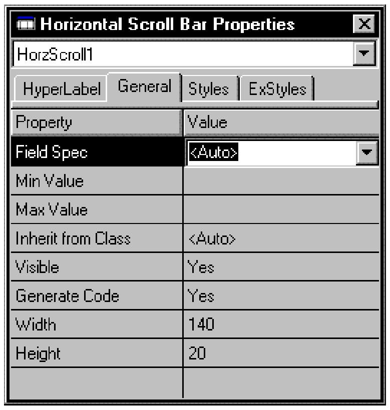
Horizontal Scroll Bar Properties and Style Settings

A horizontal scroll bar allows users to interactively select a number from a range by dragging the thumb position or using the up or down arrow buttons. A horizontal scroll bar can be bound to a field in a database. The number selected can be keyed to specific choices, to rows in an array, and so on.

When a user changes the thumb position, the TextValue, ThumbPosition, and Value accesses for the related scroll bar are changed. You can also change the thumb position from your source code using the ThumbPosition assign. You can write custom code to respond to events for a specific horizontal scroll bar object using the Window:HorizontalScroll() method (see the online help system for details).

General Tab

The General tab, shown below, allows you to specify a horizontal scroll bar's general properties, including physical appearance and class inheritance:



Property	Value
Field Spec	<Auto>
Min Value	
Max Value	
Inherit from Class	<Auto>
Visible	Yes
Generate Code	Yes
Width	140
Height	20

Note: Refer to the online help for detailed information about new and updated horizontal scroll bar properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Field Spec

Enter or choose a field specification to associate with the control.

When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications chapter for more information.

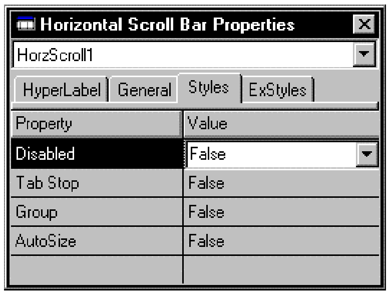
This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.

Min Value	Enter a number that corresponds to the extreme left or top position of the thumb control in the scroll bar. The maximum range (Max Value less Min Value) for a scroll bar is 65,535.
Max Value	Enter a number that corresponds to the extreme right or bottom thumb position, or *scroll box*, of the scroll bar.

Note: CA-Visual Objects automatically generates scroll bars for your data windows when they are resized. The vertical and horizontal scroll bar icons on the tool palette, therefore, are used to create *additional* scroll bars for specific controls. If you add your own scroll bars using the tool palette, you must enter source code manually via the Source Code Editor to manipulate them. (See **Chapter 8: Creating Data-Aware Windows** for more detailed information about data windows.)

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your horizontal scroll bar control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
AutoSize	False

Note: Refer to the online help for detailed information about new and updated horizontal scroll bar properties.

All of these properties have been described previously under Common Properties, with the exception of the following property that is specific to this type of control:

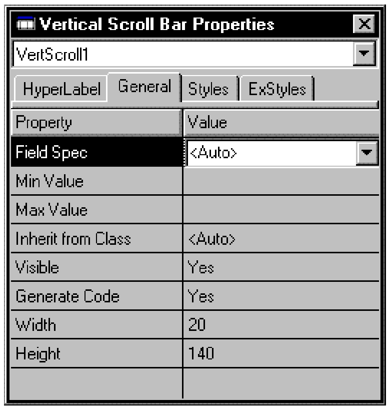
AutoSize	If True, automatically sizes the scroll bar using the system default size. The default is False.

Vertical Scroll Bar Properties and Style Settings

A vertical scroll bar allows users to interactively select a number from a range by dragging the thumb position, or using the up or down arrow buttons. A vertical scroll bar can be bound to a field in a database. The number selected may be keyed to specific choices, to rows in an array, etc. When a user changes the thumb position, the TextValue, ThumbPosition, and Value accesses for the related scroll bar are changed. You can also change the thumb position from your source code using the ThumbPosition assign. You can write custom code to respond to vertical scroll bar events using the Window:VerticalScroll() method; see the online help system for details.

General Tab

The General tab, shown below, allows you to specify a vertical scroll bar's general properties, including physical appearance and class inheritance:



Property	Value
Field Spec	<Auto>
Min Value	
Max Value	
Inherit from Class	<Auto>
Visible	Yes
Generate Code	Yes
Width	20
Height	140

Note: Refer to the online help for detailed information about new and updated vertical scroll bar properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Field Spec	Enter or choose a field specification to associate with the control.

When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications chapter for more information.

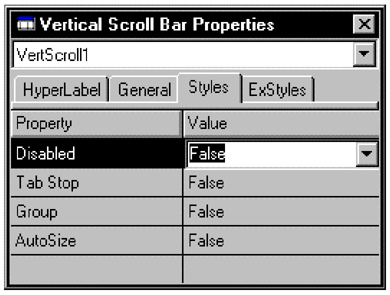
This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.

Min Value	Enter a number that corresponds to the extreme left or top position of the thumb control in the scroll bar. The maximum range (Max Value less Min Value) for a scroll bar is 65,535.
Max Value	Enter a number that corresponds to the extreme right or bottom thumb position, or *scroll box*, of the scroll bar.

Note: As mentioned earlier, CA-Visual Objects automatically generates scroll bars for your data windows when they are resized. The vertical and horizontal scroll bar icons on the tool palette, therefore, are used to create *additional* scroll bars for specific controls. If you add your own scroll bars using the tool palette, you must enter source code manually via the Source Code Editor to manipulate them.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your vertical scroll bar control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
AutoSize	False

Note: Refer to the online help for detailed information about new and updated vertical scroll bar properties.

All of these properties have been described previously under Common Properties, with the exception of the following property that is specific to this type of control:

AutoSize	If True, automatically sizes the scroll bar using the system default size. The default is False.

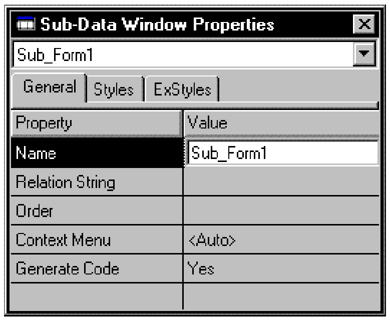
Sub-Data Window Properties and Style Settings

A data window can be placed on another data window in an owner-child relationship using a *sub-data window control.* Sub-data windows are often used to display the detail table in a master-detail data window. When a data window is “nested” within another data window in this way, it is referred to as a *sub-data window*. There is no difference between a sub-data window and a data window other than the sub-data window control defaults to browse view.

Note: For a complete discussion of data windows, data dialog windows, and sub-data windows, refer to **Chapter 8: Creating Data-Aware Windows**.

General Tab

The General tab, shown below, allows you to specify a sub-data window control's general properties, including physical appearance and class inheritance:



Property	Value
Name	Sub_Form1
Relation String	
Order	
Context Menu	<Auto>
Generate Code	Yes

Note: Refer to the online help for detailed information about new and updated sub-data window properties.

The General tab contains the following properties:

Name

Enter the name of the control. This property is required for all controls and is 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 "PushButton1."

The Name property is used in the generated source code to identify individual controls within the form. Therefore, control names must be unique within a single form design.

Relation String

Enter a *relation string*, i.e. the name of a field from the master data server that will be used to perform a lookup operation in the detail data server's controlling order. This means that the detail server must have an open index file with a controlling order whose key is based on a similar field.

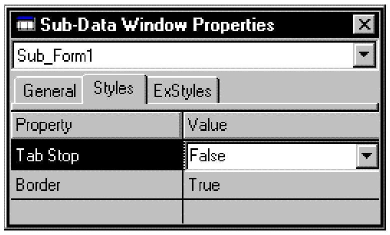
The field name in the detail server does not have to be the same as the field name that you use for the relation string, but in some cases it will be. For example, #CustNum is the relation string for the CustOrd data window in the Order Entry sample application that you replicated in the *Getting Started* guide. (If you have not done this yet, see **Chapter 8: Creating Data-Aware Windows** for instructions.) The important thing is that the data in the relation string field matches the data in the key field of the detail server's controlling order.

Note: When you select a sub-data window control, it will show only the Name, Relation String, Order, and Generate Code properties. If you double-click on the sub-data window control, however, the program launches a new copy of the Window Editor in which you can view and change all of the standard data window properties described earlier in Specifying Window Properties. See **Chapter 8: Creating Data-Aware Windows** for more information on using the sub-data window control for nesting data windows.

Order	Enter the name of the index file with the *controlling order* for the detail server (for example, **CUSTNAME.NTX**). (For more information about index files and orders, see Adding Index Files in **Chapter 7: Defining Data Servers** and Field Specifications.)
Generate Code	Specify whether certain source code will be generated for the control. This is a Yes / No option that you change by selecting a value from a drop-down list. The default value of Yes indicates that source code will be generated. Changing this property to No can cut down on the resources needed to use the window, but you can only do this if you do not need access to the control in your application. For example, controls that are not data-aware (such as fixed text and fixed icon) are seldom referenced in your application source code, so you can change the Generate Code property to No to prevent the Window Editor from generating unnecessary code. **Note:** The code to display a control is always generated, regardless of the status of this property.

Styles Tab

The Styles tab, shown below, allows you to add a border and to set tab stops for a sub-data window box control:



Property	Value
Tab Stop	False
Border	True

Note: Refer to the online help for detailed information about new and updated sub-data window properties.

The Styles tab contains the following properties:

Tab Stop	If True, allows the user to press the Tab key to move through any number of controls defined with this option. **Note:** By default, the cursor tabbing order follows a left-to-right, top-to-bottom progression, based on the position on the controls on the form. You may, however, change the tab order using the Control Order dialog box. (See Changing Tab Order by Reordering Controls later in this chapter for more information.)
Border	Creates a border around the sub-data control, if True. True is the default setting for the Border option.

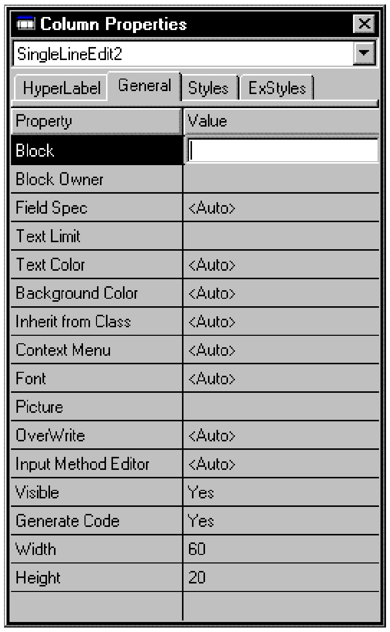
Column Properties Window and Styles Settings

A data window or a sub-data window control can be expanded by inserting one or more columns and defining its properties.

Note: The Column toolbar icon and its corresponding menu command, Edit Insert Column, are available only when the data window is in *browse view*. For more detailed information, refer to **Chapter 8: Creating Data-Aware Windows**.

General Tab

The General tab, shown below, allows you to specify a column's general properties, including physical appearance and class inheritance:



Property	Value
Block	
Block Owner	
Field Spec	<Auto>
Text Limit	
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Picture	
OverWrite	<Auto>
Input Method Editor	<Auto>
Visible	Yes
Generate Code	Yes
Width	60
Height	20

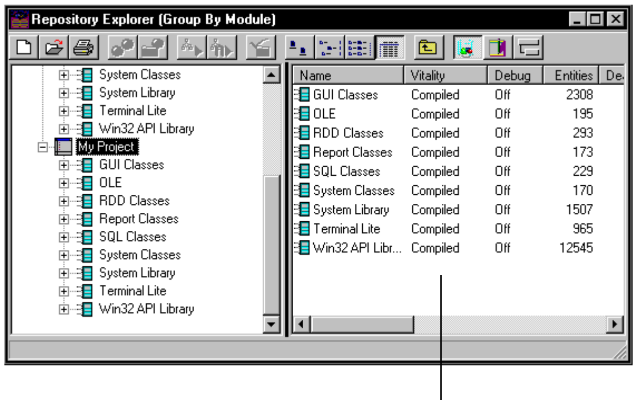
Note: Refer to the online help for detailed information about new and updated column properties.

The General tab contains the following properties that are specific to this type of control:

Block	Optionally specify the code block to be associated with the column. This option is used to filter the data in/out of the data column.
Block Owner	Enter the owner of the code block associated with a column, if specified. The owner would typically be the data server or `SELF:Server`.
Field Spec	Enter or choose a field specification to associate with the column. When you associate a field specification with a column control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications for more information. This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.
Text Limit	Enter the maximum number of characters allowed in the column control.
Picture	Optionally, enter a picture string for the single-line edit control. If there is a field specification with a picture string already associated with the control, the SingleLineEdit:Picture access/assign overwrites the field specification's picture string.
OverWrite	If the Picture property is defined for the single-line edit control, select a constant value for SingleLineEdit:Picture. Valid values are: ■ OVERWRITE_NEVER If selected, specifies that the single-line edit control always be in INSERT mode. ■ OVERWRITE_ONKEY If selected, specifies that the mode depends on the state of the INSERT key. ■ OVERWRITE_ALWAYS If selected, specifies that the single-line edit control always be in OVERWRITE mode.

List View Properties and Style Settings

A list view control allows the user to view, add, delete, and arrange a list of items wherein each item consists of an icon and a label. For example, the right pane of the Repository Explorer is a list view control:



Name	Vitality	Debug	Entities	De
GUI Classes	Compiled	Off	2308	
OLE	Compiled	Off	195	
RDD Classes	Compiled	Off	293	
Report Classes	Compiled	Off	173	
SQL Classes	Compiled	Off	229	
System Classes	Compiled	Off	170	
System Library	Compiled	Off	1507	
Terminal Lite	Compiled	Off	965	
Win32 API Libr...	Compiled	Off	12545	

Listviewcontrol

The contents of a list view control can be displayed in one of four view types: Icons, Small Icons, List, and Report. (See Styles Tab below for full descriptions of each view type.) Other options allow the end user to edit labels, scroll items, and select more than one item at a time.

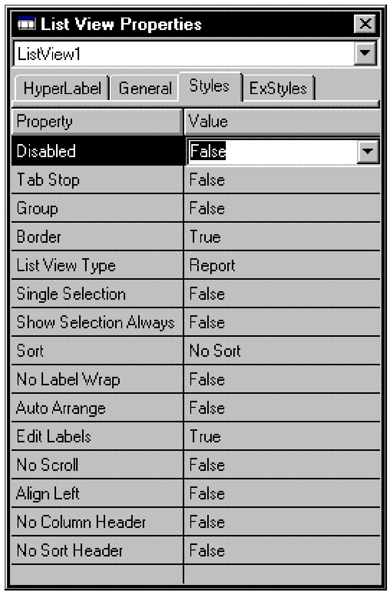
General Tab

Note: Refer to the online help for detailed information about new and updated list view properties.

All of the general properties for a list view have been described previously under Common Properties.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your list view control:



List View Properties

ListView1

Property	Value
Disabled	False
Tab Stop	False
Group	False
Border	True
List View Type	Report
Single Selection	False
Show Selection Always	False
Sort	No Sort
No Label Wrap	False
Auto Arrange	False
Edit Labels	True
No Scroll	False
Align Left	False
No Column Header	False
No Sort Header	False

Note: Refer to the online help for detailed information about new and updated list view properties.

The Styles tab contains the following properties that are specific to this type of control:

Border	Creates a border around the list view control, if True. True is the default setting for the Border option.

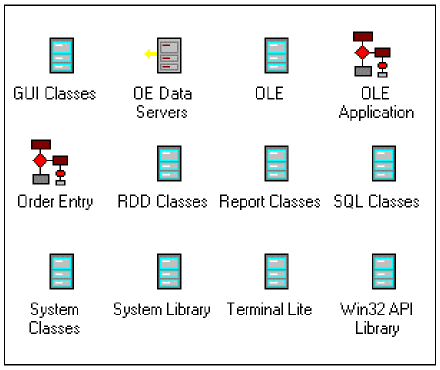
List View Type

Specifies the display format for a list view. Valid choices are:

- Icons

  If selected, displays an item as a full-sized icon, with a label underneath it, that the end user can drag to any location within the list view.

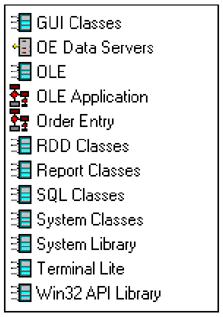
  For example:



- Small Icons

  If selected, displays an item as a small icon, with a label to the right of it, that the end user can drag to any location within the list view.

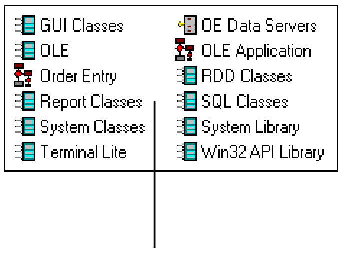
  For example:



- List

  If selected, displays items as small icons with labels to the right. Items are arranged in columnar format and cannot be manipulated by the end user.

  For example:



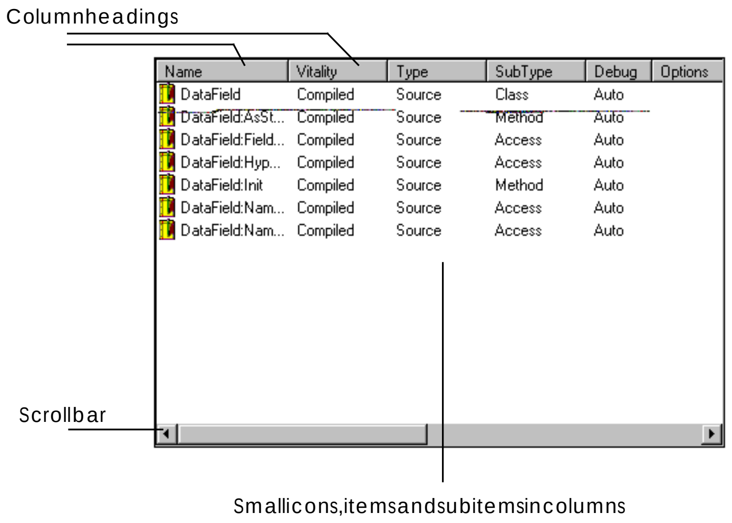
Listtype:
Smallicons,itemsincolumns,nosubitems

- Report

  If selected, displays items as small icons with labels in the leftmost column. Subitems, as specified by the application, are arranged on the same line in additional columns to the right. Each column has a heading (unless the No Header Column option is set to False).

  For example:

Reportview:



Name	Vitality	Type	SubType	Debug	Options
DataField	Compiled	Source	Class	Auto	
DataField:AsSt...	Compiled	Source	Method	Auto	
DataField:Field...	Compiled	Source	Access	Auto	
DataField:Hyp...	Compiled	Source	Access	Auto	
DataField:Init	Compiled	Source	Method	Auto	
DataField:Nam...	Compiled	Source	Access	Auto	
DataField:Nam...	Compiled	Source	Access	Auto	

Single Selection — If True, this option limits the end user’s selection to a single item.

Show Selection Always — If False, the highlighting will be removed when an item in a list view loses focus (for example, if you Tab to another control). When the list view regains focus, the previous selection will not be highlighted.

If True, the highlighted or selected text will remain highlighted when you Tab to another control. In this fashion, any item in the list view stays marked (until you move to another record in a database or close the window).

False is the default setting.

Sort — The Sort option is available for all view types, and is used to specify a sort order for the list of items. Items can be sorted by label, by subitem, or by any other property. Valid choices are:

- No Sort

  Items in list are left unsorted.

- Ascending

  Sorts items in ascending order by label only. (No comparison function is needed.)

- Descending

  Sorts items in descending order by label only. (No comparison function is needed.)

The default is No Sort.

No Label Wrap	If True, specifies that label text should not wrap to the next line in Icons or Small Icons view only. Rather, text that exceeds a column's width will not wrap.
Auto Arrange	If True, repositions items on a grid at all times.
Edit Labels	If True, this option allows the end user to edit the item labels. The default is True.
No Scroll	If True, the end user cannot scroll to view any items that do not fit in the client area of the control window. The default is False.
Align Left	If True, items in either the Icons or Small Icons view are aligned along the left edge of the list view window. If False, the items are aligned along the top of the list view window. The default is False.
No Column Header	Specifies that columns in Report view do not have headings, if True. If False, column headers appear for item and subitem. False is the default setting. **Note:** Columns can be resized by the end user by dragging the column guides between columns.
No Sort Header	If False, column headers do not act like buttons. This is useful when there should be no actions triggered by clicking on a column's header.

Tree View Properties and Style Settings

A tree view control presents the end user with a hierarchical list of items in a tree structure that can be expanded or collapsed. Each item in the tree consists of a label with an optional icon and may have an associated list of subitems. For example, the left pane of the Repository Explorer is a tree view that lists projects, applications, modules, and entities in a top-down hierarchy.

The contents of a tree view control can be displayed with buttons that expand and collapse subitems (also called *child items*) and lines that link subitems to their parent items and/or to the hierarchy's root level. Other options allow the end user to select more than one item at a time and to edit item labels.

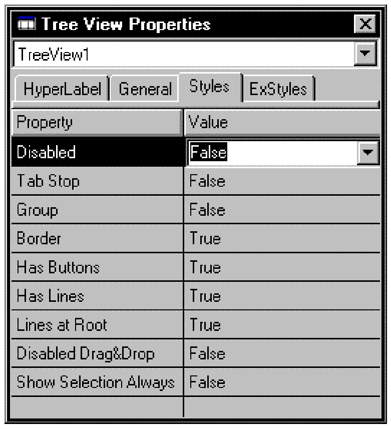
General Tab

Note: Refer to the online help for detailed information about new and updated tree view properties.

All of the general properties for a tree view have been described previously under Common Properties.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your tree view control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
Border	True
Has Buttons	True
Has Lines	True
Lines at Root	True
Disabled Drag&Drop	False
Show Selection Always	False

Note: Refer to the online help for detailed information about new and updated tree view properties.

The Styles tab contains the following properties that are specific to this type of control:

Border	Creates a border around the tree view control, if True. True is the default setting for the Border option.
Has Buttons	If True, specifies that a button be placed to the left of a parent item so that the end user can click the button to expand or collapse its subitems.

Note: The Has Buttons by itself option does not add buttons at the root level of the hierarchy. To do so, you must set all three options—Has Buttons, Has Lines, and Lines at Root—to True.

Has Lines	Draws lines that link subitems to their parent items in the tree hierarchy, if True. The default is True.

Lines at Root	If True, specifies that lines be drawn linking parent items to the *root item*, which is an item that has no parent and is, therefore, the topmost level in the hierarchy. The default is True.
Disabled Drag&Drop	If True, does not allow the end user to use the drag-and-drop method of manipulation items in the tree structure.
Show Selection Always	If False, the highlighting will be removed when an item in a tree view loses focus (for example, if you Tab to another control). When the tree view regains focus, the previous selection will not be highlighted. If True, the highlighted or selected text will remain highlighted when you Tab to another control. In this fashion, any item in the tree view stays marked (until you move to another record in a database or close the window). False is the default setting.

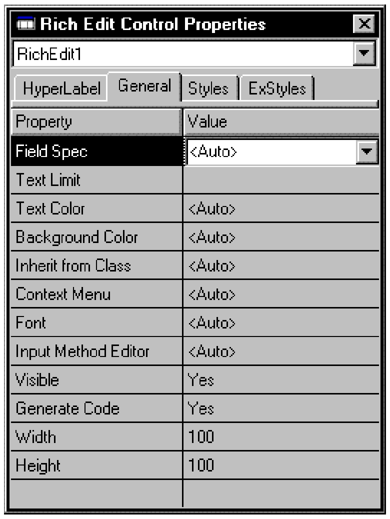
Rich Edit Control Properties and Style Settings

Unlike a multi-line edit control which lets you only enter, edit, and delete straight text, a rich edit control allows you to format and print text as well. You can set tabs, use indentation, align and number text; and for characters, you can specify font, size, text and background color, italics, and so on.

In addition to character and paragraph formatting, you can also embed OLE objects in rich edit controls.

General Tab

The General tab, shown below, allows you to specify a rich edit control's general properties, including physical appearance and class inheritance:



Property	Value
Field Spec	<Auto>
Text Limit	
Text Color	<Auto>
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Input Method Editor	<Auto>
Visible	Yes
Generate Code	Yes
Width	100
Height	100

Note: Refer to the online help for detailed information about new and updated rich edit control properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Field Spec

Enter or choose a field specification to associate with the control.

When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. For more information on these properties, refer to **Chapter 7: Defining Data Servers** and Field Specifications for more information.

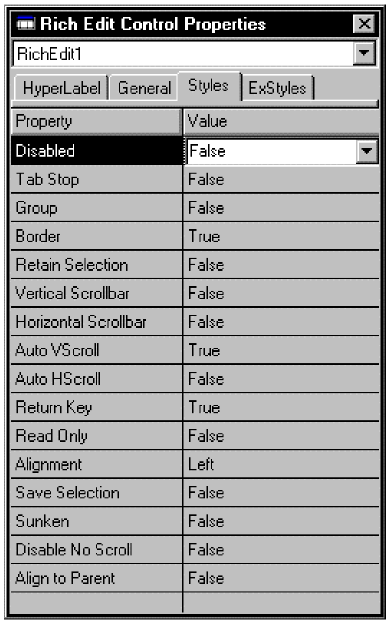
This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.

Text Limit

Enter the maximum number of characters allowed in the rich edit control.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your rich edit control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
Border	True
Retain Selection	False
Vertical Scrollbar	False
Horizontal Scrollbar	False
Auto VScroll	True
Auto HScroll	False
Return Key	True
Read Only	False
Alignment	Left
Save Selection	False
Sunken	False
Disable No Scroll	False
Align to Parent	False

Note: Refer to the online help for detailed information about new and updated rich edit control properties.

The Styles tab contains the following properties that are specific to this type of control:

Border	If True, creates a rich edit control with a border.
Retain Selection	If False, the highlighting will be removed when the edit control loses focus (for example, if you Tab to another control). When the edit control regains focus, the previous selection will not be highlighted. False is the default setting. If True, the highlighted or selected text will remain highlighted when you Tab to another control. In this fashion, any edit control you have edited stays marked (until you move to another record in a database or close the window).
Vertical Scrollbar	Creates a rich edit control with a vertical scroll bar (if True).
Horizontal Scrollbar	Creates a rich edit control with a horizontal scroll bar, where all the text is on the first line.
Auto VScroll	If True, text is automatically scrolled up or down one line when the cursor reaches the beginning or end of a line.
Auto HScroll	If True, any text in the rich edit control is automatically scrolled 10 characters to the right when the user reaches the end of the line while typing.
Return Key	If True, the Enter key acts as a return key, moving the cursor to the next line of the multi-line edit control. If False, the Enter key activates the default push button for the window and has no effect on the edit control.
Read Only	If True, this option stops the user from entering/editing text in the rich edit control.
Alignment	This property is used to defined how the text is to be aligned. Valid options are: ■ Left If selected, left-justifies text in the rich edit control. Left is the default setting for the Alignment option. ■ Center If selected, centers text in the rich edit control. ■ Right If selected, right-justifies text in the rich edit control.
Save Selection	If True, saves the current selection when the rich edit control loses focus. When it regains focus, the entire contents of the control appear.
Sunken	Draws the rich edit control with a sunken border so that it appears recessed within the window, if True.

Disable No Scroll	If True, disables the scroll bar instead of hiding it if one is not needed.
Align to Parent	Aligns the rich edit control with the parent window's client area, if True.

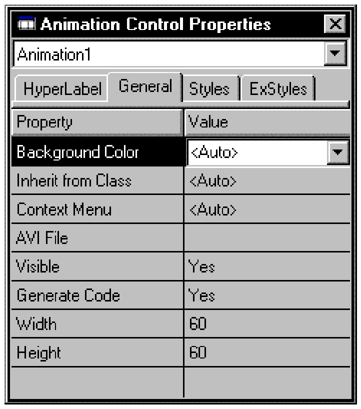
Animation Control Properties and Style Settings

An animation control is used to display a video clip (.AVI file), such as the one used in Windows when a file is deleted and placed in the Recycle Bin:

Note: Only silent Audio Video Interleaved (AVI) video clips can be used with animation controls.

General Tab

The General tab, shown below, allows you to specify an animation control's general properties, including physical appearance and class inheritance:



Property	Value
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
AVI File	
Visible	Yes
Generate Code	Yes
Width	60
Height	60

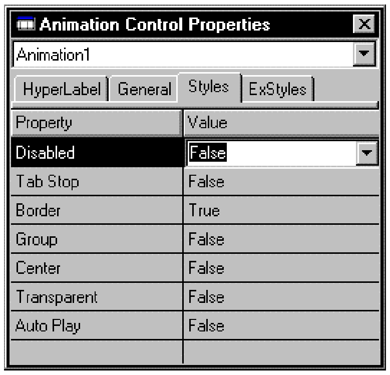
Note: Refer to the online help for detailed information about new and updated animation control properties.

All of these properties have been described previously under Common Properties, with the exception of the following property that is specific to this type of control:

AVI File	Enter the name of the .AVI file to be associated with this control.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your animation control:



Property	Value
Disabled	False
Tab Stop	False
Border	True
Group	False
Center	False
Transparent	False
Auto Play	False

Note: Refer to the online help for detailed information about new and updated animation control properties.

The Styles tab contains the following properties that are specific to this type of control:

Border	If True, creates a border around the animation control.
Center	If True, centers the video clip within the animation control's window.
Transparent	If True, specifies that the video clip will overlay the window with a transparent background, rather than using the background color specified in the video clip.
Auto Play	Starts playing the video clip as soon as the parent window is opened, if True.

Hotkey Edit Control Properties and Style Settings

A hotkey edit control enables the end user to select a valid key combination as a shortcut for performing an operation or accessing another window—for example, F1 to access an online help system or Ctrl+S to save data entries.

General Tab

Note: Refer to the online help for detailed information about new and updated hotkey edit control properties.

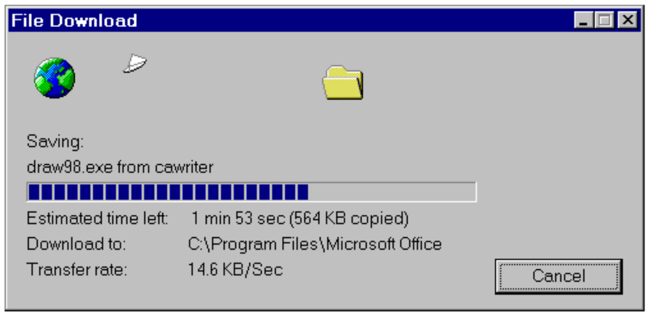
All of the general properties for hotkey edit controls have been described previously under Common Properties.

Styles Tab

All of the style settings for hotkey edit controls have been described previously under Common Properties.

Progress Bar Properties and Style Settings

Progress bar controls are used to visually indicate the progress of lengthy tasks, such as installation and compilation operations. For example, below is a typical Internet file download dialog box with a progress bar:



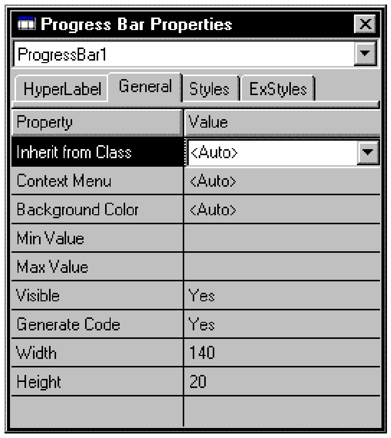
Saving:
draw98.exe from cawriter

Estimated time left:	1 min 53 sec (564 KB copied)
Download to:	C:\Program Files\Microsoft Office
Transfer rate:	14.6 KB/Sec

Every progress bar has two features: a range and a current position. The range denotes the length of the task from start to completion, and the current position indicates the progress made. The system uses the range and current position to calculate progress as a percentage and colors a corresponding percentage of the progress bar.

General Tab

The General tab, shown below, allows you to specify a progress bar's general properties, including physical appearance and class inheritance:



Property	Value
Inherit from Class	<Auto>
Context Menu	<Auto>
Background Color	<Auto>
Min Value	
Max Value	
Visible	Yes
Generate Code	Yes
Width	140
Height	20

Note: Refer to the online help for detailed information about new and updated progress bar properties.

All of these properties have been described previously under Common Properties, with the exception of the following property that is specific to this type of control:

Min Value	Enter an unsigned integer that corresponds to the starting point in the range. The allowable range is 0 to 65,535.
Max Value	Enter an unsigned integer that corresponds to the completion point in the range. The allowable range is 0 to 65,535.

Note: The default minimum and maximum settings are 0 and 100, respectively.

Styles Tab

All of the style settings for progress bars have been described previously under Common Properties.

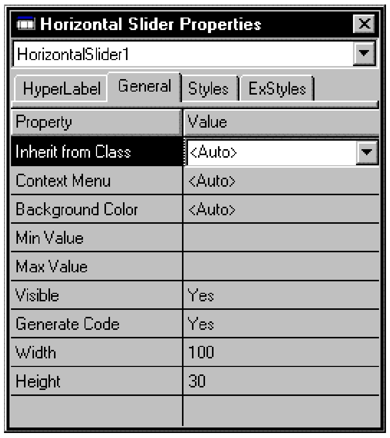
Note: Refer to online help for detailed information about new and updated progress bar properties.

Horizontal Slider Properties and Style Settings

A horizontal slider control, or *trackbar*, is used to select a specific value or set of consecutive values in a range of records or options. It typically includes a horizontal slider, or *thumb*, and *tick marks* that indicate the incremental values in the range. For example, Windows provides a horizontal slider control for setting the double-click speed of your mouse.

General Tab

The General tab, shown below, allows you to specify a horizontal slider control's general properties, including physical appearance and class inheritance:


Horizontal Slider Properties

HorizontalSlider1

HyperLabel | General | Styles | ExStyles

Property	Value
Inherit from Class	<Auto>
Context Menu	<Auto>
Background Color	<Auto>
Min Value	
Max Value	
Visible	Yes
Generate Code	Yes
Width	100
Height	30

Note: Refer to the online help for detailed information about new and updated horizontal slider properties.

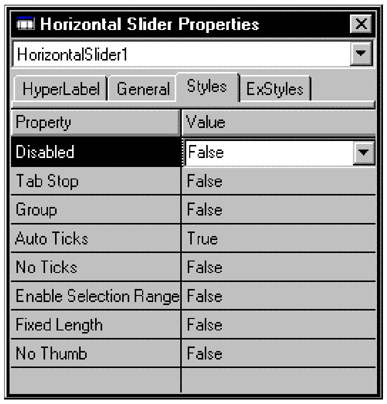
All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Min Value — Enter a value that corresponds to the extreme left position for the selection range. The default is 0.

Max Value — Enter a value that corresponds to the extreme right position for the selection range. For example, if the total number of increments in a range is 10, then the maximum value would
be 11, as position 0 would be the thumb's starting point and
1 to 10 would the range's incremental positions.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your horizontal slider control:


Horizontal Slider Properties

HorizontalSlider1

HyperLabel | General | Styles | ExStyles

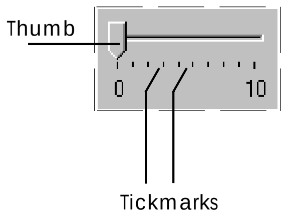
Property	Value
Disabled	False
Tab Stop	False
Group	False
Auto Ticks	True
No Ticks	False
Enable Selection Range	False
Fixed Length	False
No Thumb	False

Note: Refer to the online help for detailed information about new and updated horizontal slider properties.

The Styles tab contains the following properties that are specific to this type of control:

Auto Ticks

If True, creates a horizontal slider control with a tick mark for each increment in its range of values. For example:

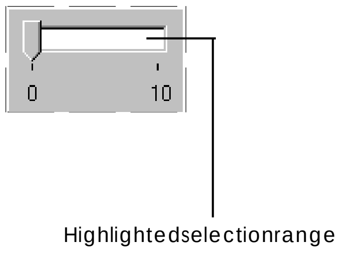


No Ticks

If True, creates a horizontal slider control without tick marks.

Enable Selection Range

If True, creates a horizontal slider control with a selection range that is highlighted. For example:

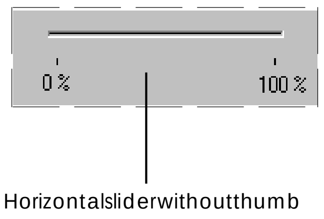


Fixed Length

If True, the length of the horizontal slider control does not vary as the selection range changes. The default setting is False.

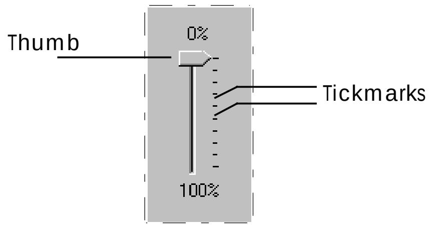
No Thumb

If True, creates a horizontal slider control without a thumb. For example:



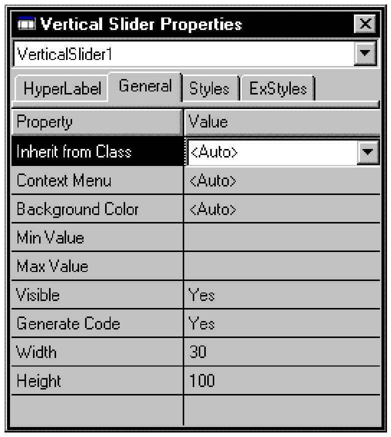
Vertical Slider Properties and Style Settings

A vertical slider control, or *trackbar*, is used to select a specific value or set of consecutive values in a range of records or options. It typically includes a vertical slider, or *thumb*, and *tick marks* that indicate the incremental values in the range. For example:



General Tab

The General tab, shown below, allows you to specify a vertical slider control's general properties, including physical appearance and class inheritance:



Property	Value
Inherit from Class	<Auto>
Context Menu	<Auto>
Background Color	<Auto>
Min Value	
Max Value	
Visible	Yes
Generate Code	Yes
Width	30
Height	100

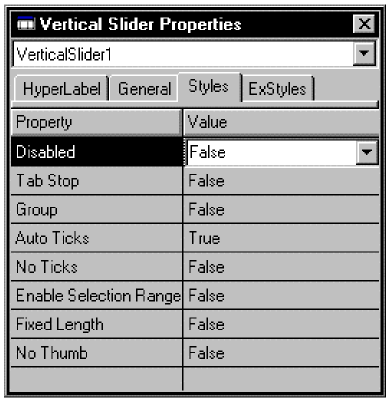
Note: Refer to the online help for detailed information about new and updated vertical slider properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Min Value	Enter a number that defines the minimum value in the slider's selection range.
Max Value	Enter a number that defines the maximum value in the slider's selection range. For example, if the total number of increments in a range is 10, then the maximum value would be 11, as position 0 would be the thumb's starting point and 1 to 10 would the range's incremental positions.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your vertical slider control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
Auto Ticks	True
No Ticks	False
Enable Selection Range	False
Fixed Length	False
No Thumb	False

Note: Refer to the online help for detailed information about new and updated vertical slider properties.

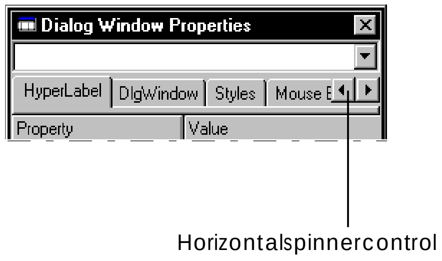
The Styles tab contains the following properties that are specific to this type of control:

Auto Ticks	If True, creates a vertical slider control with a tick mark for each increment in its range of values. The default setting is True.
No Ticks	If True, creates a vertical slider control without tick marks. The default setting is False.
Enable Selection Range	If True, creates a vertical slider control with a selection range that is highlighted. The default setting is False.
Fixed Length	If True, the length of the vertical slider control does not vary as the selection range changes. The default setting is False.
No Thumb	If True, creates a vertical slider control without a thumb. The default setting is False.

Horizontal Spinner Properties and Style Settings

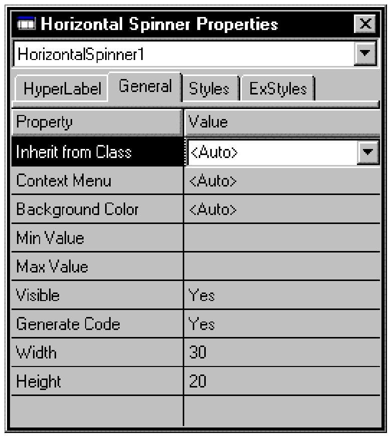
A horizontal spinner control consists of a pair of Left and Right arrow buttons that are used to increment or decrement a value, respectively. Note that a horizontal spinner is typically paired with a *companion control* that allows the end user to enter a valid value or select one from those displayed, using the arrow buttons.

For example, the Properties window for the various window forms uses a horizontal spinner control that allows you to scroll left to right (and right to left) through its tab pages:



General Tab

The General tab, shown below, allows you to specify a horizontal spinner's general properties, including physical appearance and class inheritance:



Property	Value
Inherit from Class	<Auto>
Context Menu	<Auto>
Background Color	<Auto>
Min Value	
Max Value	
Visible	Yes
Generate Code	Yes
Width	30
Height	20

Note: Refer to the online help for detailed information about new and updated horizontal spinner properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Min Value	Enter a number that defines the minimum value in the spinner's selection range.

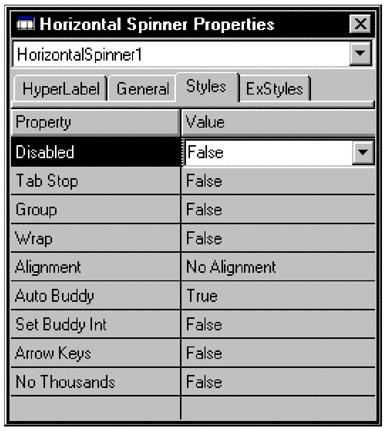
Max Value

Enter a number that defines the maximum value in the spinner's selection range.

Note: If the maximum value is less than the minimum value, the Left arrow button actually *decreases* the current position as it moves towards the maximum value.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your horizontal spinner control:


Horizontal Spinner Properties

HorizontalSpinner1

Property	Value
Disabled	False
Tab Stop	False
Group	False
Wrap	False
Alignment	No Alignment
Auto Buddy	True
Set Buddy Int	False
Arrow Keys	False
No Thousands	False

Note: Refer to the online help for detailed information about new and updated horizontal spinner properties.

The Styles tab contains the following properties that are specific to this type of control:

Wrap

If True, wraps the spinner's current position to the opposite extreme if the end user attempts to go beyond either the minimum or maximum value. The default setting is False.

Alignment

Specifies the type of alignment between the spinner and its companion control. Valid choices are: No Alignment, Left, and Right. The default setting is No Alignment.

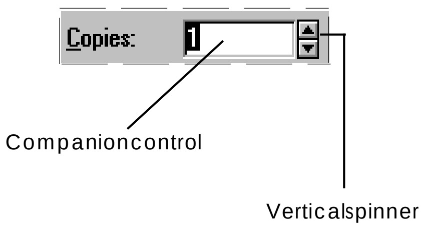
Auto Buddy

If True, the window control defined most recently is automatically selected as the companion, or *buddy*, control for
the spinner. Companion controls are typically single-line edit controls or list boxes. The default setting is True.

Set Buddy Int	If True, sets the caption of the companion, or buddy, control for the spinner whenever the current position changes. If the companion control is a list box, the spinner control sets its current selection instead of its caption. The default setting is False.
Arrow Keys	Specifies that keyboard equivalents (LEFT and RIGHT arrow keys) may be used for the Left and Right arrows, if True. The default setting is False.
No Thousands	Specifies that no separators be inserted in a decimal string to denote thousands, if True. The default setting is False.

Vertical Spinner Properties and Style Settings

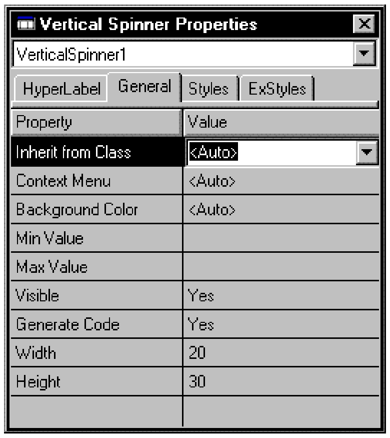
A vertical spinner control—sometimes called a *spin control* or *Up-Down control*—consists of a pair of Up and Down arrow buttons that are used to increment or decrement a value, respectively. For example, the Copies field on a standard Print dialog box is usually a vertical spinner control:



Note that a vertical spinner is typically paired with a *companion control* that allows the end user to enter a valid value or select one from those displayed, using the arrow buttons.

General Tab

The General tab, shown below, allows you to specify a vertical spinner's general properties, including physical appearance and class inheritance:



Property	Value
Inherit from Class	<Auto>
Context Menu	<Auto>
Background Color	<Auto>
Min Value	
Max Value	
Visible	Yes
Generate Code	Yes
Width	20
Height	30

Note: Refer to the online help for detailed information about new and updated vertical spinner properties.

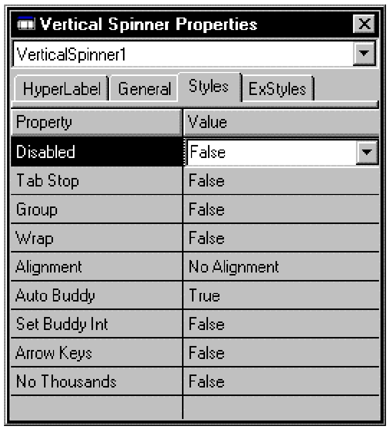
All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Min Value	Enter a number that defines the minimum value in the vertical spinner's selection range.
Max Value	Enter a number that defines the maximum value in the vertical spinner's selection range.

Note: If the maximum value is less than the minimum value, the Up arrow button actually *decreases* the current position as it moves towards the maximum value.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your vertical spinner control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
Wrap	False
Alignment	No Alignment
Auto Buddy	True
Set Buddy Int	False
Arrow Keys	False
No Thousands	False

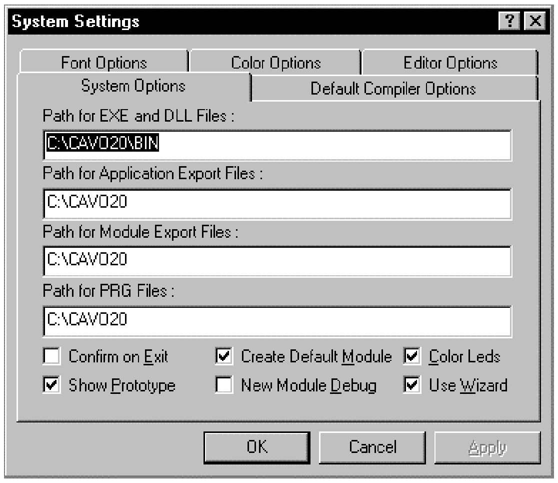
Note: Refer to the online help for detailed information about new and updated vertical spinner properties.

The Styles tab contains the following properties that are specific to this type of control:

Wrap	If True, wraps the vertical spinner's current position to the opposite extreme if the end user attempts to go beyond either the minimum or maximum value. The default is False.
Alignment	Specifies the type of alignment between the vertical spinner and its companion control. Valid choices are: No Alignment, Left, and Right.
Auto Buddy	If True, the window control defined most recently is automatically selected as the companion control for the vertical spinner. Companion controls are typically single-line edit controls or list boxes. True is the default setting.
Set Buddy Int	If True, sets the caption of the companion, or buddy, control for the spinner whenever the current position changes. If the companion control is a list box, the spinner control sets its current selection instead of its caption. The default setting is False.
Arrow Keys	If True, specifies that keyboard equivalents (UP and DOWN arrow keys) may be used for the Up and Down arrows.
No Thousands	Specifies that no separators be inserted in a decimal string to denote thousands, if True.

Tab Control Properties and Style Settings

A tab control is used to present data or a series of choices in a multiple-page format. It consists of one or more tabbed pages that resemble file folders. When the end user clicks on one of the tabs, the corresponding page moves to the forefront and allows access to its data and controls. For example, the System Settings dialog box in CA-Visual Objects is a tab control that has five pages:



Path for EXE and DLL Files : C:\CAVO20\BIN

Path for Application Export Files : C:\CAVO20

Path for Module Export Files : C:\CAVO20

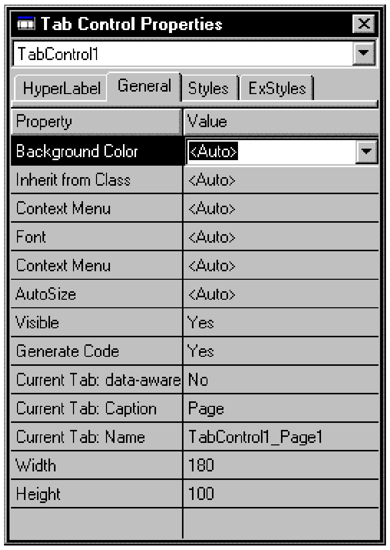
Path for PRG Files : C:\CAVO20

☐ Confirm on Exit	☑ Create Default Module	☑ Color Leds
☑ Show Prototype	☐ New Module Debug	☑ Use Wizard

Note: When a tab control is placed on a form, it initially consists of a single page. See Manipulating Tab Control Pages later in this chapter for information about adding pages to a tab control and editing it.

General Tab

The General Tab, shown below, allows you to specify a tab control's general properties, including physical appearance and class inheritance:



Property	Value
Background Color	<Auto>
Inherit from Class	<Auto>
Context Menu	<Auto>
Font	<Auto>
Context Menu	<Auto>
AutoSize	<Auto>
Visible	Yes
Generate Code	Yes
Current Tab: data-aware	No
Current Tab: Caption	Page
Current Tab: Name	TabControl1_Page1
Width	180
Height	100

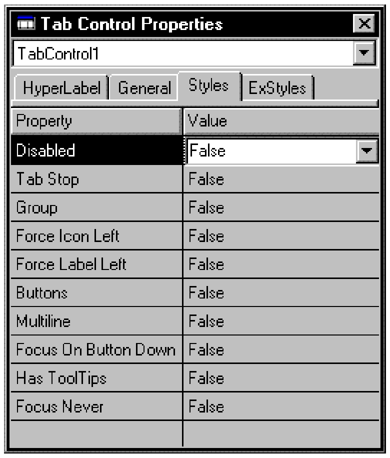
Note: Refer to the online help for detailed information about new and updated tab control properties.

All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

AutoSize	If True, allows the tab control to size itself to the size of its pages when created. Otherwise, it respects the size specified in the Window Editor.
Current Tab: Data Aware	If Yes, the current page is a data-aware control. The default value is Yes.
Current Tab: Caption	Enter the caption that should appear on the tab. The default is Page.
Current Tab: Name	Enter the page name for the current tab. The default is TabControl*n*_Page *n*.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your tab control:

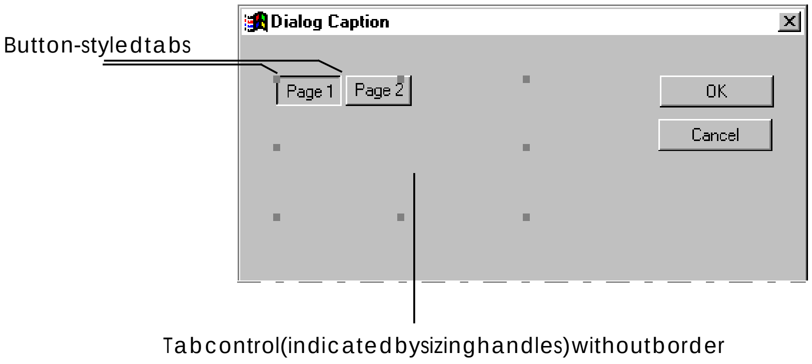


Property	Value
Disabled	False
Tab Stop	False
Group	False
Force Icon Left	False
Force Label Left	False
Buttons	False
Multiline	False
Focus On Button Down	False
Has ToolTips	False
Focus Never	False

Note: Refer to the online help for detailed information about new and updated tab control properties.

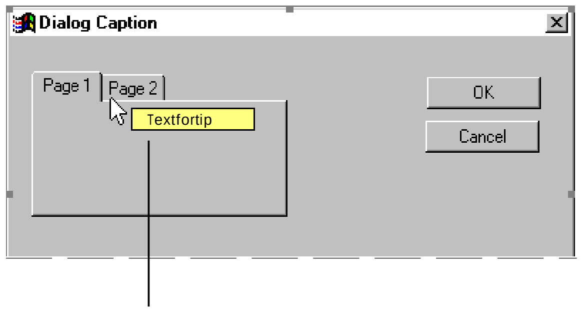
The Styles tab contains the following properties that are specific to this type of control:

Force Icon Left	As a tab may have both an icon and a caption, this option forces the icon to the left edge of the tab and centers the caption at the same time, if True.
Force Label Left	If True, this option forces both a label and its icon (optional) to left edge of the tab.
Buttons	If True, creates a tab control with tabs that look like buttons. For example:



Note: Button-styled tabs should be used like other button controls; that is, they should perform a function or execute a command instead of displaying a page. Therefore, the remaining area of the tab control is typically unused and, consequently, is not surrounded by a border.

Multiline	If True, displays multiple rows of tabs rather than a single row with a vertical spinner control that indicates to the end user that scrolling is required to view additional tabs. (A single row of left-aligned tabs and a vertical spinner—if necessary—is the default style.)
Focus On Button Down	If True, draws the tab that currently has focus in a recessed format. In our example just above, the "Page 1" button emulates a button that has been pressed down by the end user. In fact, this option is customarily used only with the Buttons option.
Has ToolTips	If True, adds a tooltip control to each tab. Tooltip controls are small pop-up windows that display descriptive text when the mouse moves over them. For example:



Tooltip control

Focus Never	If True, specifies that a tab is never the recipient of the input focus when clicked.

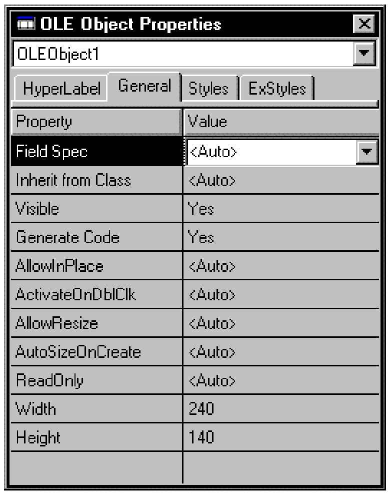
OLE Object Control Properties and Style Settings

An OLE object control (OCX) allows you to seamlessly embed other applications into the application you are currently designing. For example, if you were creating a financial application like our Order Entry sample application and you wanted to add spreadsheet capability to it, you could do so by inserting Microsoft Excel as an OLE object control.

Note: You can also *link* OLE objects to your applications. See Linking and Embedding OLE Objects later in this chapter for more detailed information about using OLE objects and OCX controls in CA-Visual Objects.

General Tab

The General tab, shown below, allows you to specify an OLE object control's general properties, including physical appearance and class inheritance:



Property	Value
Field Spec	<Auto>
Inherit from Class	<Auto>
Visible	Yes
Generate Code	Yes
AllowInPlace	<Auto>
ActivateOnDblClk	<Auto>
AllowResize	<Auto>
AutoSizeOnCreate	<Auto>
ReadOnly	<Auto>
Width	240
Height	140

Note: Refer to the online help for detailed information about new and updated OLE object properties.

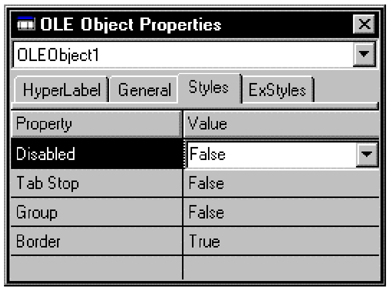
All of these properties have been described previously under Common Properties, with the exception of the following properties that are specific to this type of control:

Field Spec	Enter or choose a field specification to associate with the control. When you associate a field specification with a control, all properties defined for the field specification (such as validation rules and picture) are automatically inherited and used by the control. This property is available for check boxes, single- and multi-line edit controls, rich edit controls, list and combo boxes, scroll bars, columns, radio button groups, and OLE objects.
Inherit from Class	Select the Control class or subclass from which the specified window control will inherit its basic characteristics.
AllowInPlace	If True, specifies that the control behave like the actual application that is embedded as an OLE object inside the CA-Visual Objects container application for the duration of the current editing session. During in-place activation, the embedded application's and the container application's menus and toolbars are merged, and the object can be edited inside the CA-Visual Objects container application's user interface context.

ActivateOnDblClk	If True, allows the end user to double-click on the OLE object control to activate it.
AllowResize	Allows the end user to resize an OLE object control, if True.
AutoSizeOnCreate	If True, allows the OLE control to size itself when created. Otherwise, it respects the size specified in the Window Editor.
ReadOnly	If True, this option prevents the end user from editing the OLE object control.

Styles Tab

The Styles tab, shown below, allows you to define the style settings for your tab control:



Property	Value
Disabled	False
Tab Stop	False
Group	False
Border	True

Note: Refer to the online help for detailed information about new and updated OLE object properties.

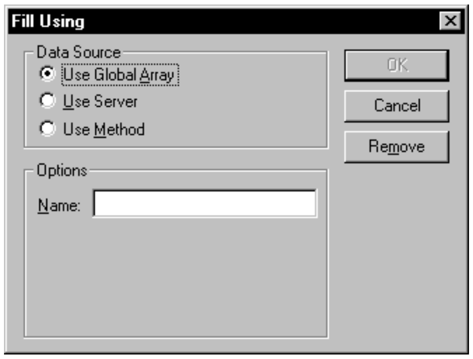
The Styles tab contains the following property that is specific to this type of control:

Border Creates a border around the OLE object control (if True). True is the default setting for the Border option.

Defining Arrays for List and Combo Boxes

Note: In CA-Visual Objects 2.7, combo boxes and list boxes can be with filled with the contents of *any* array expression, not just a global array variable. See the online help for more information about the Use Array Expression option, which replaces the User Global Array option.

Select the Fill Using property on the General tab of the Properties window to access the Fill Using dialog box:

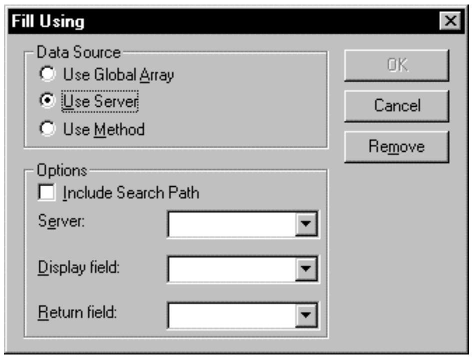


This dialog box allows you to specify how a list box or combo box will be filled. The Use Global Array and Use Server radio buttons determine whether the dialog box will be filled with the contents of an array or a data server, respectively. The Use Method radio button allows you to create a customized method for filling the list box or combo box.

If you choose Use Global Array or Use Method, you must enter the name of the array or method in the Name edit control. For example, to populate a list box of states or provinces for customer addresses, you might use an array named "States."

Note: Arrays must be created manually outside the Window Editor. See the *Programmer's Guide* for detailed information about creating dynamic and dimensioned arrays.

If you choose Use Server, the Options group box changes slightly, with the Name edit control replaced by the Server combo box as shown below:



Check the Include Search Path check box if you want to choose from data servers that are defined in your application’s search path (for example, a data server library). Leave it unchecked if you want to choose only from data servers defined in the current application.

In the Server combo box, specify the name of a data server. In the Display Field combo box, specify the name of the field you want to display. In the Return Field combo box, specify the name of the field whose value you want to return when the user makes a selection.

Note: The names of the Display Field and Return Field will often be the same.

Click OK to close the dialog box.

Push Button Controls and Actions

The Window Editor makes it easy for you to associate actions with push button controls. When a push button is clicked, the window always calls a predefined *event* based on the push button’s Name property. First, it will attempt to invoke a method with the same name. The Window Editor creates a skeleton for this method; to add your own source code to the method, click the Ellipsis button for the Click Event property. If no method by that name is defined in the current window class or any of its superclasses, the window will look for a Window class entity and then a ReportQueue class entity with the same name to invoke. In cases where there is no appropriate event defined for a push button, nothing happens when it is clicked.

Note: This same technique is used by the Menu Editor to associate events with menu selections. See “GUI Classes” in the *Programmer’s Guide* for a complete discussion of predefined menu and button events.

You can, therefore, use a carefully named push button control to cause a particular event to happen when the push button is clicked. For example, if you want a push button on a dialog window to open another dialog window, use the name of the subordinate dialog window as the push button control name. Similarly, if you want a push button on a data window to perform a standard database operation, name the push button using the corresponding DataWindow class method (for example, a push button named Skip will invoke the DataWindow:Skip() method when clicked). See the online help system for information about the methods of a particular class.

The system is quite flexible in that, at any time, you can override the default by defining a customized method using the Click Event property. Doing this invokes the Source Code Editor with a predefined METHOD declaration statement for your particular window subclass.

You enter the source code that suits your particular needs (which may include a call to a similar method in a superclass), and the Window Editor saves this method in the current module along with the other code that it generates. Then, when the push button is clicked, it will automatically invoke your customized method.

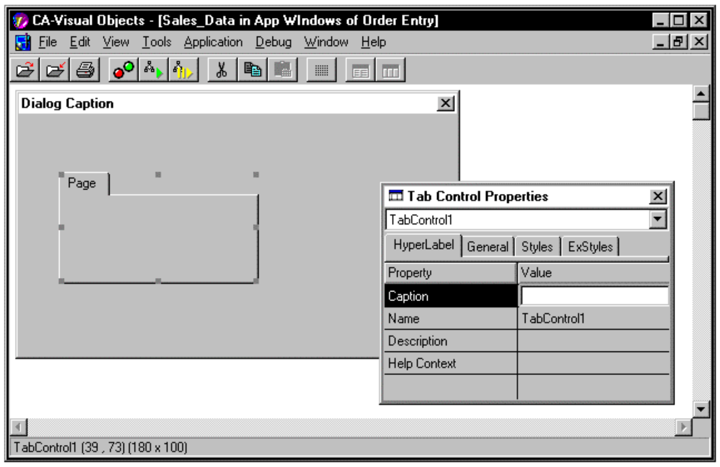
Manipulating Tab Control Pages

As mentioned earlier in this chapter, a tab control when first placed on a form consists initially of a single page. To create a multi-page tab control, you must add pages to the original control. To do so:

Creating a Tab Control

1. Use the tool palette to place a tab control on the specified form.

   For example, you may want to expand the sample Order Entry application to include a dialog window with a multi-page tab control for sales data:



Property	Value
Caption	
Name	TabControl1
Description	
Help Context	

TabControl1 (39 , 73) (180 x 100)

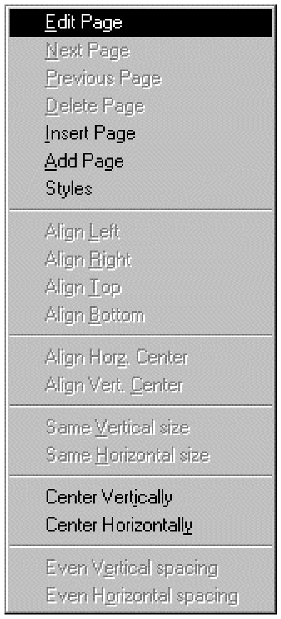
2. Specify the tab control's basic properties such as Caption and Current Tab:Caption (for example, **Sales Data** and **Salespersons**, respectively).

   See the General Tab and Styles Tab sections, as well as Common Properties, for full descriptions of the available properties.

Adding Another Page

3. Right-click on the tab control.

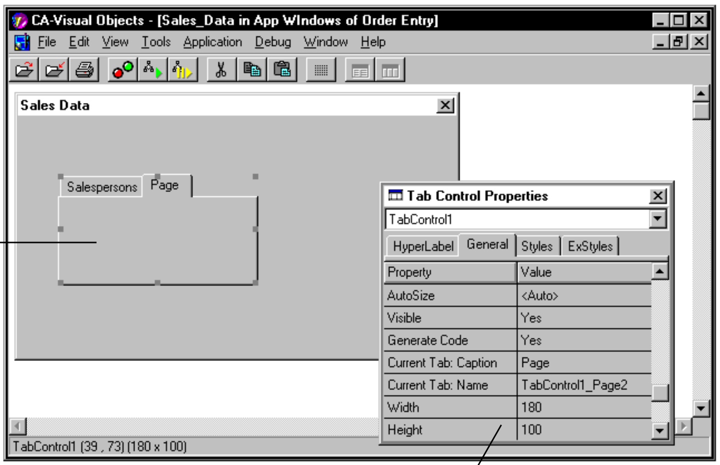
   A local pop-up menu appears with several commands exclusively for tab controls, in addition to standard editing commands like Center Vertically:



Note: The Edit Page, Next Page, Previous Page, Delete Page, Insert Page, Add Page, and Styles commands are accessible only via the local pop-up menu, whereas the other standard editing commands are also available from the Edit menu.

4. Choose the Add Page command.

Another page is added to the tab control; and when you click on its tab, it moves to the forefront:


CA-Visual Objects - [Sales_Data in App WIndows of Order Entry]
File Edit View Tools Application Debug Window Help
Sales Data
Salespersons
Page
Tab Control Properties
TabControl1
HyperLabel
General
Styles
ExStyles

Property	Value
AutoSize	<Auto>
Visible	Yes
Generate Code	Yes
Current Tab: Caption	Page
Current Tab: Name	TabControl1_Page2
Width	180
Height	100

TabControl1 (39 , 73) (180 x 100)

Newpage

FocusofPropertieswindowchangestonewpage

Note: After adding a new page to the tab control, the Next Page, Previous Page, and Delete Page local menu commands become activated.

5. Replace the Current Tab:Caption property's default value of "Page" with **Sales Regions**, for example.

Editing a Page

6. Right-click on the Salespersons page.

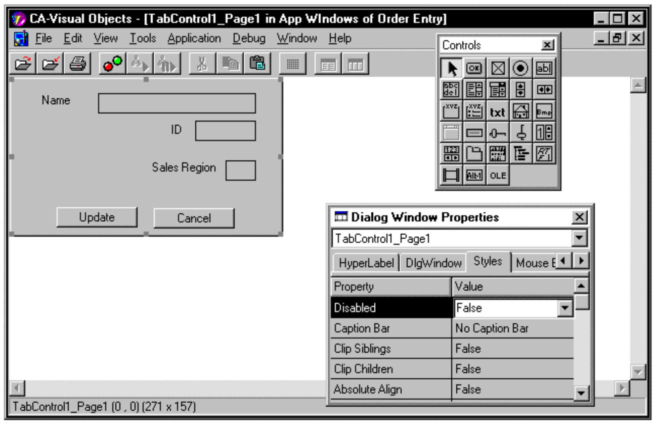
   The local pop-up menu reappears.

7. Select the Edit Page command from the local pop-up menu.

   A new window appears for "Page1," as well as the Dialog Window Properties window and the tool palette. This is because each page of a tab control is actually a modeless dialog window that you must define.

8. Add a few controls—such as single-line edit controls, fixed text controls, and push buttons—using the tool palette, and specify their properties.

For example, you might add controls for the salesperson's name, ID, and sales region, in addition to push buttons that allow the end user to update or cancel any data entered:


CA-Visual Objects - [TabControl1_Page1 in App WIndows of Order Entry]
File Edit View Tools Application Debug Window Help
Controls
Name
ID
Sales Region
Update
Cancel
Dialog Window Properties
HyperLabel DlgWindow Styles Mouse E

TabControl1_Page1

Property	Value
Disabled	False
Caption Bar	No Caption Bar
Clip Siblings	False
Clip Children	False
Absolute Align	False

TabControl1_Page1 (0 , 0) (271 x 157)

9. Save your changes and close out of the “Page1” window.

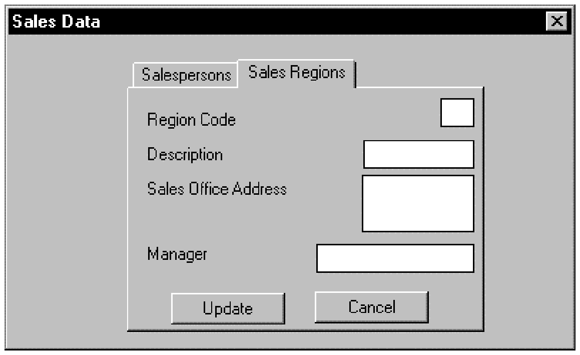
   You are returned to the Sales Data tab control.

10. Repeat steps 6 - 9 for the Sales Region page, adding controls for region code, description, etc. in the window for “Page2.”

11. Save your changes and recompile.

    Note: Separate form entities are created for each page of the tab control, as well as for its parent window.

The new Sales Data dialog window, when completely finished, should look something like this:



Linking and Embedding OLE Objects and Controls

Object linking and embedding (OLE) is not a new technology. It has been around before the birth of Windows, and was used mostly in Microsoft Office applications to create compound documents or to embed one or more foreign applications inside another application. With Microsoft OLE 2.0, a full 32-bit technology that is a central component of Windows, OLE is now an umbrella term that includes many subtechnologies—automation, data transfer, memory allocation, file management, and OLE custom controls (OCXs).

Note: CA-Visual Objects 2.7 is a full-fledged OLE client/server application, and supports all of OLE 2.0 technologies. For detailed information about creating ActiveX controls and OLE server applications, see the online help.

Basic Terms

Before showing you how to use OLE objects and controls in CA-Visual Objects, a brief discussion of some basic terms is in order.

Linking and Embedding

Linking and embedding objects into compound documents or container applications is the basic foundation of OLE. These two methods store items, which were created by one application, inside a document of another application. The application that created the object is called the *server application* and the application that stores the object is called the *container* or *client application*.

Embedding

Embedding is the more common of the two methods, and there are several ways of embedding an object into a container application. For example, you could create a spreadsheet in MS Excel, and then place it inside your CA-Visual Objects application by using the Window Editor’s Edit Paste or Paste Special menu command and standard drag-and-drop editing techniques. You could insert it using the Edit Insert OLE Object menu command. (The CA-Visual Objects Window Editor fully supports OLE 2.0 features.)

After the object is placed in the container application, it is in a *passive* state. An object will stay in this state until it is necessary to make modifications to it, in which case it will become *active* by some user action defined by the client. Activating the object launches the application used to create the object (for example, MS Excel).

The application can be launched in two ways, depending on how the client application has implemented activation support. The first way is for the client to start the server as a separate application in a separate window. The second way is for the client to become the server application for the duration of the object editing session. In this case, which is called *in-place activation*, the client will change itself into the server. This change encompasses menus, toolbars, status bars, and any palette windows.

In-place activation requires no extra user interaction; simply placing the cursor on the object will activate it. Thus, embedded objects are indistinguishable from the client application's native data. This model can be used when the overhead of activating the object is small.

Linking

An object can also be linked to the client application. A linked object is a representation of (or *pointer* to) the actual object which resides elsewhere (either in the same application or in a different application).

An object can be linked by choosing the Window Editor's Edit Paste Special menu command and selecting the Paste Link option in the Paste Special dialog box that appears. The Paste Link option creates a link to the source file for the embedded object, so that changes to the source file will be reflected in your application. The Insert Object dialog box also supports linking of objects (see Inserting an OLE Object below).

Editing a linked object is very similar to editing an embedded object. The only difference is that the data for a linked object remains in the application which created the object, whereas the data for an embedded object travels with the object to the client application. Additionally, editing a linked object is always done "out-of-place."

Controls and Control Containers

An OLE custom control is a special kind of embedded OLE 2.0 object that has an extended interface that lets it behave like a Windows control. OCXs are a set of extensions that turn simple OLE 2.0 containers and objects into more powerful tools.

The OLE 2.0 standard for compound documents meets many of the requirements for both controls and control containers, but not all of them. Creating a control involves some other issues, primarily those dealing with OLE automation. A control must expose its events, methods, and properties to a control container; and a control container must expose *ambient properties* and its own events to the control. Ambient properties are named characteristics or values of the container itself that generally apply to all controls in the container. Some examples of ambient properties are default colors, font, and whether the container is in *design mode* or *run mode.*

The difference between controls and simple OLE objects is that controls generally do not need a lot of user interface components, like toolbars and menus. However, they do have additional needs for event capture—such as focus and keystrokes.

Note: For an in-depth discussion of utilizing OLE objects and controls, refer to the "Object Linking and Embedding" chapter in the *Programmer's Guide*.

Inserting an OLE Object

The CA-Visual Objects Window Editor is both an OLE object and an OCX container. Consequently, it provides the following OLE features:

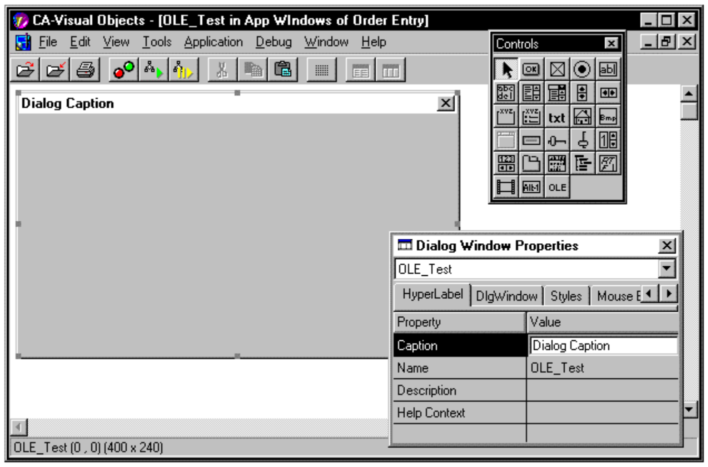
- Design and runtime modes
- Drag-and-drop placement of objects and controls
- Embedding and linking options
- In-place activation and editing of embedded objects
- Ability to access all the methods, properties, and events of an OLE custom control (OCX)

To insert an OLE object in your application, follow these basic steps:

1. Start the Window Editor.

   Note: You *must* have the OLE library in your application's search path in order to embed and link OLE objects and controls. See **Default Path Options** in "Working in the Desktop" for information about modifying your application's default search path.

   Notice that the OLE icon in the Window Editor's tool palette is now active:


CA-Visual Objects - [OLE_Test in App WIndows of Order Entry]
File Edit View Tools Application Debug Window Help
Controls
txt
OLE
Dialog Caption
Dialog Window Properties
HyperLabel DlgWindow Styles Mouse E

OLE_Test

Property	Value
Caption	Dialog Caption
Name	OLE_Test
Description	
Help Context	

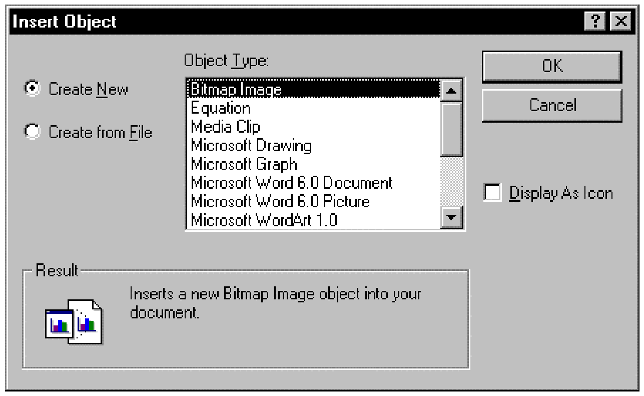
OLE_Test (0 , 0) (400 x 240)

2. Specify the new window's properties, such as its Caption property.

3. Drag-and-drop the OLE icon onto the form.

   Alternatively, choose the Insert OLE Object command from the Edit Menu.

   In the latter instance, the Insert Object dialog box appears:



Insert Object

(● Create New) ○ Create from File

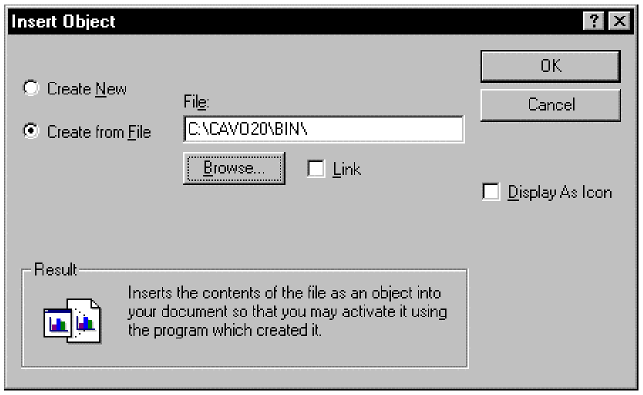
Object Type: Bitmap Image (selected), Equation, Media Clip, Microsoft Drawing, Microsoft Graph, Microsoft Word 6.0 Document, Microsoft Word 6.0 Picture, Microsoft WordArt 1.0

☐ Display As Icon

Result: Inserts a new Bitmap Image object into your document.

4. Select the Create from File radio button. (Select the Create New radio button and an object type if you wish to create a new object.)

   The Insert Object dialog box changes slightly when you choose the Create from File option:



Insert Object

○ Create New (● Create from File)

File: C:\CAVO20\BIN\

☐ Link

☐ Display As Icon

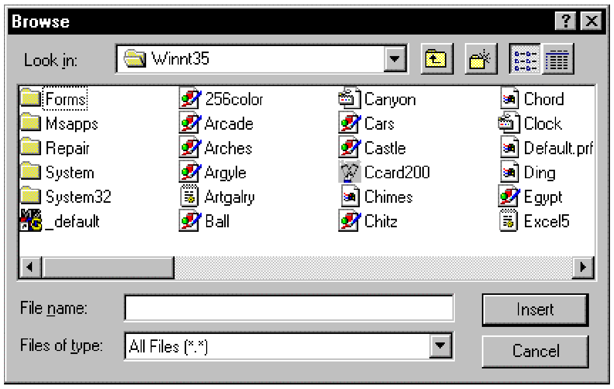
Result: Inserts the contents of the file as an object into your document so that you may activate it using the program which created it.

5. Optionally, select the Link option to link the object to the selected file so that any changes to the file will be reflected in your document.

6. Optionally, select the Display As Icon option if you want the embedded object to appear as an icon on your form.

7. Click the Browse push button.

A standard Browse dialog box appears:



Look in: Winnt35

Forms	256color	Canyon	Chord
Msapps	Arcade	Cars	Clock
Repair	Arches	Castle	Default.prf
System	Argyle	Ccard200	Ding
System32	Artgalry	Chimes	Egypt
_default	Ball	Chitz	Excel5

File name:

Files of type: All Files (*.*)

8. Select a file (for example, Cars.bmp from the WINNT35 folder).

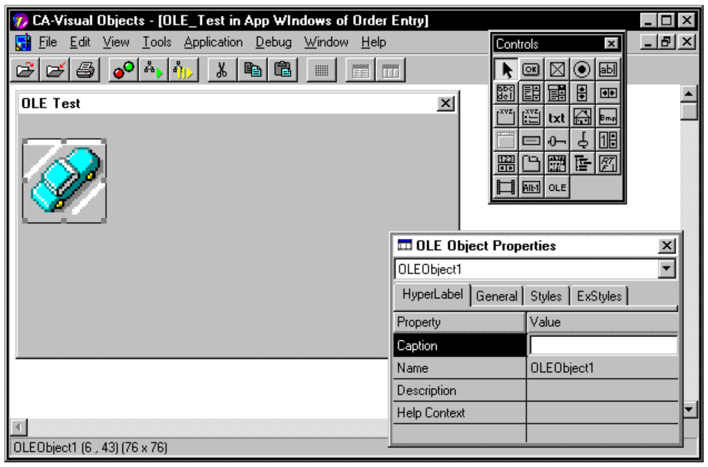
   Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory.

9. Click Open.

   You are returned to the Insert Object dialog box.

10. Click OK.

    You are returned to the Window Editor, and the OLE object is added your form:



OLEObject1

Property	Value
Caption	
Name	OLEObject1
Description	
Help Context	

OLEObject1 (6 , 43) (76 x 76)

11. Specify the OLE object's properties, such as Caption, AllowInPlace, and ActivateOnDblClk, using the OLE Object Properties window.

12. Click the Save toolbar button to save the form entity.

    Alternatively, choose the File Save menu command.

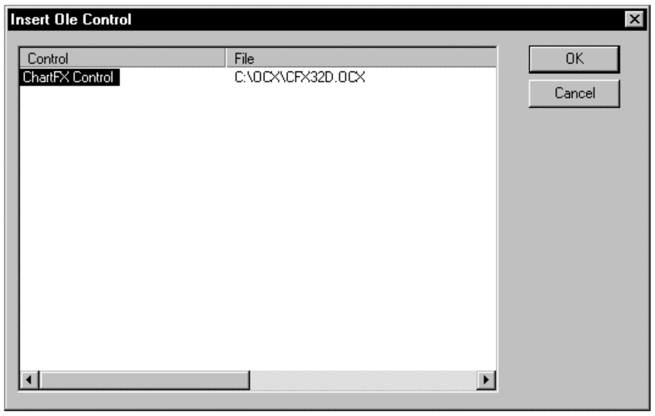
Afterwards, when the application is recompiled and up and running, you can activate the program that created the Clipart bitmap image object.

Inserting an OLE Control

To insert an OLE control (OCX) in your application instead of (or in addition to) an OLE object, follow these basic steps:

1. Restart the Window Editor, if closed.
2. Choose the Insert OLE Control command from the Edit Menu.

   The Insert OLE Control dialog box appears:

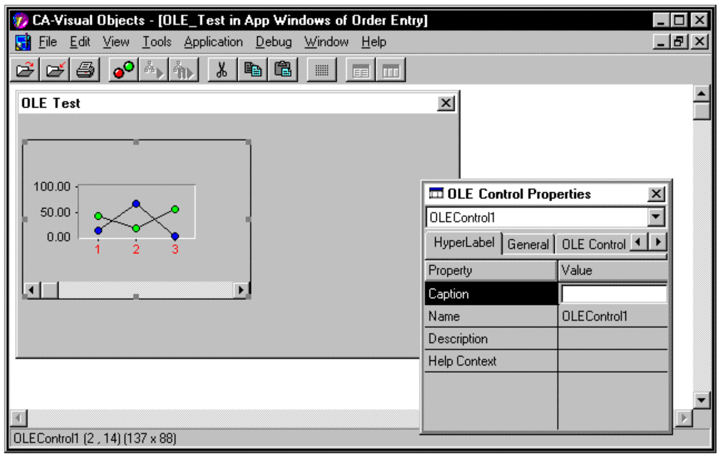


Control	File
ChartFX Control	C:\OCX\CFX32D.OCX

   This dialog box displays all of the OLE controls and associated files currently in the OLE database in the CA-Visual Objects registry. (See Setting Up OLE Controls for information about registering an OLE control).

3. Select a control (for example, ChartFXControl), and then click OK.

   The OLE control is added to your form:


CA-Visual Objects - [OLE_Test in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
OLE Test
100.00
50.00
0.00
1
2
3
OLE Control Properties
HyperLabel
General
OLE Control

OLEControl1

Property	Value
Caption	
Name	OLEControl1
Description	
Help Context	

OLEControl1 (2 , 14) (137 x 88)

4. Specify the OLE control's properties, such as Caption, Control Properties, and Event:LButtonDblClk, using the OLE Control Properties window.
5. Click the Save toolbar button to save the form entity.

   Alternatively, choose the File Save menu command.

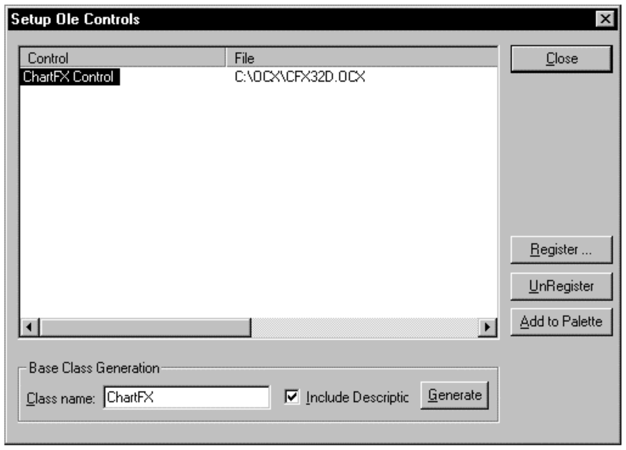
Setting Up OLE Controls

OLE controls must be prepared, or *set up*, for use in your applications. Preparation includes registering the controls and generating the CA-Visual Objects wrapper code for their exported properties and methods. You can also add OCXs as icons to the Window Editor's tool palette.

To set up an OLE control:

1. Restart the Window Editor, if closed.
2. Choose the Setup OLE Control command from the Tools Menu.

   The Setup OLE Controls dialog box appears:

   ChartFX Control C:\OCX\CFX32D.OCX

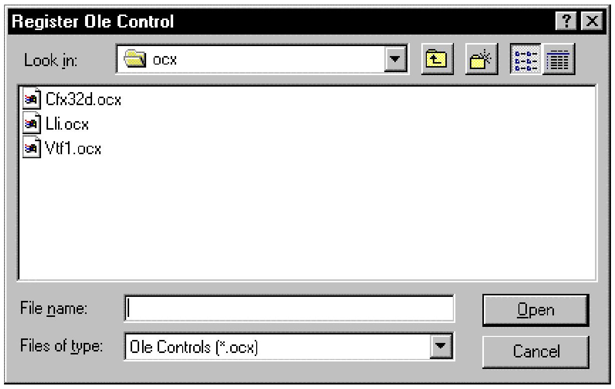
   Class name: ChartFX

   Note: In this version of CA-Visual Objects, this dialog box has two new options, Include Containing Objects and Remove from Palette. For detailed information, see the online help.

   This dialog box displays all of the OLE controls and associated files currently in the OLE database in the CA-Visual Objects registry.

3. Click the Register button to register a control that is not currently displayed in the Control/File list box.

   The Register OLE Control dialog box appears:

   Look in: ocx

   Cfx32d.ocx
   Lli.ocx
   Vtf1.ocx

   Files of type: Ole Controls (*.ocx)

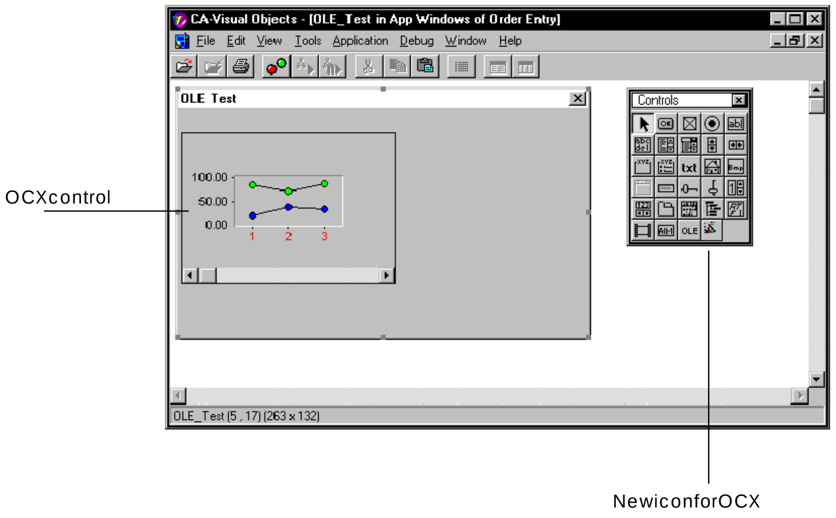
4. Select an .OCX file (for example, VTFL.OCX), and click Open.
5. Click OK in the Window Editor’s message box when informed that the OLE control is successfully registered.
6. Optionally, click the Generate button.

   This option is used to generate the base class source code for the OLE control just registered.
7. If you choose the Generate option, you can also click on the Include Description check box.

   This option adds a description for the OLE control entity in the Repository Explorer’s list view pane, as well as in the source code.
8. Select the Add to Palette option.

   This option, if selected, adds the specified OCX as an icon to the Window Editor’s tool palette.
9. Click Close.

   You are returned to the Window Editor. Notice that a new icon for the OCX has been added to the tool palette:


CA-Visual Objects - [OLE_Test in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
OLE Test
Controls
100.00
50.00
0.00
1 2 3
OLE_Test (5 , 17) (263 x 132)
OCXcontrol
NewiconforOCX

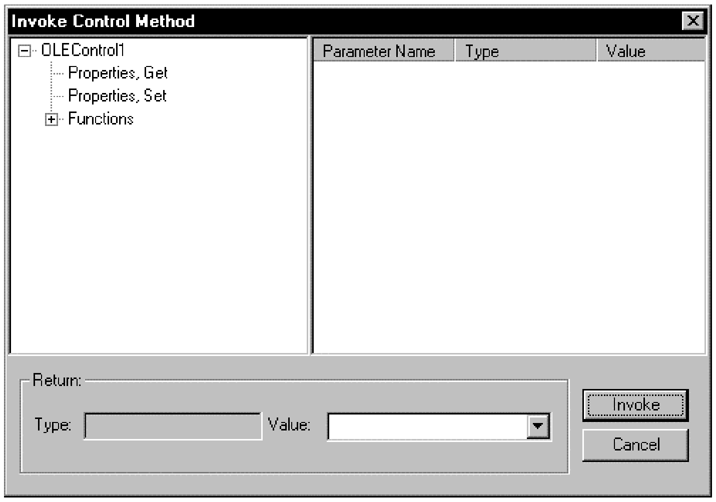
Note: You can also select a control and remove it from the registry by clicking on the UnRegister button in the Setup OLE Control dialog box.

Invoking OLE Control Methods

You can set, test, and view the initial properties and functions associated with an OLE control, including its physical appearance at runtime. To do so:

1. Choose the OLE Control Methods command from the Edit menu.

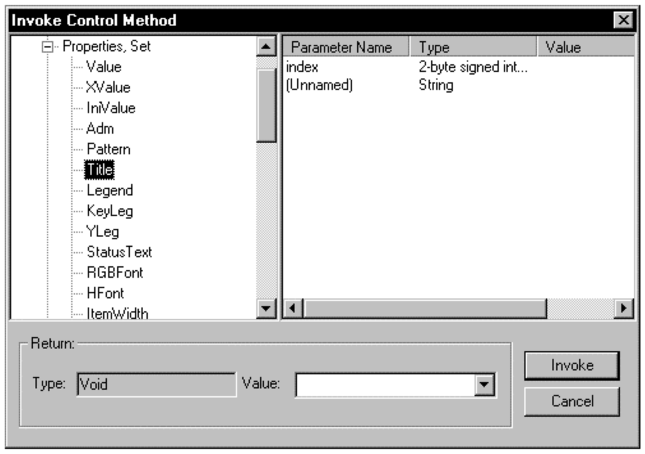
   The Invoke Control Method dialog box appears:



   This dialog box displays a list of Set properties, Get properties, and Functions for the selected control.

2. Expand a branch of the tree structure in the left pane of this dialog box (for example, Properties, Set), and select a property to be set and tested (for example, Title).

   The property's parameter names, data types, and values appear in the Parameter Name/Type/Value list view:



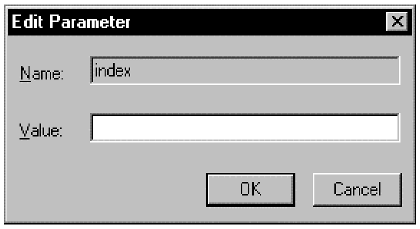
Parameter Name	Type	Value
index	2-byte signed int...	
(Unnamed)	String	

Return: Type: Void Value:

   Notice that the Return group box displays the return value for the specified method.

3. Double-click on the index parameter.

   The Edit Parameter dialog box appears:

   Edit Parameter

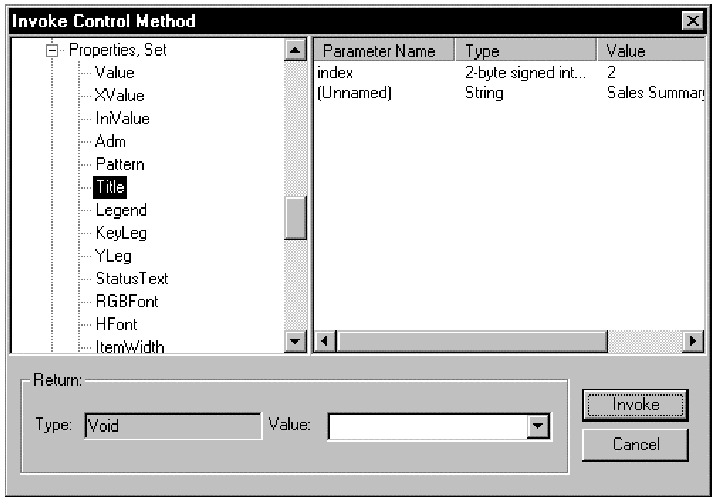
   Name: index

   Value:

4. Enter a value (for example, **2**), and then click OK.
5. Double-click on the (Unnamed) parameter.
6. The Edit Parameter dialog box appears again.

   Enter a value (for example, **Sales Summary**).
7. Click the Invoke button.

   Both values appear in the Value column of the list view:

   Invoke Control Method

   Properties, Set: Value, XValue, IniValue, Adm, Pattern, Title, Legend, KeyLeg, YLeg, StatusText, RGBFont, HFont, ItemWidth

Parameter Name	Type	Value
index	2-byte signed int...	2
(Unnamed)	String	Sales Summar

   Return:

   Type: Void

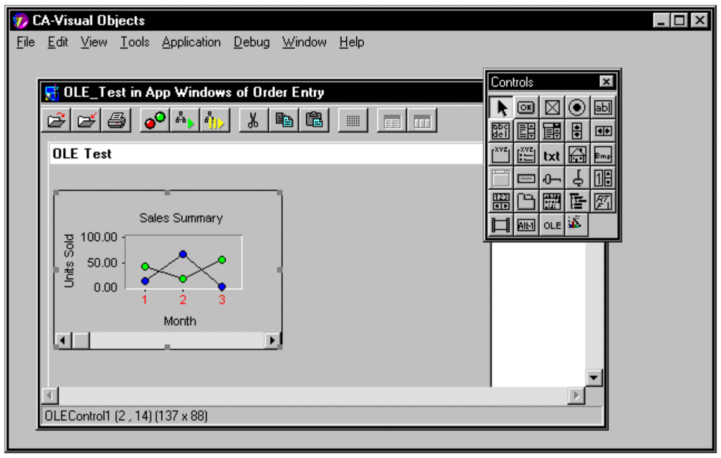
   Value:

8. Repeat steps 3–7, entering the following pairs of values:

   0, **Units Sold** and **3**, **Month**.

   Note: Remember to click the Invoke button after setting each parameter, keeping in mind that a property may have any number of parameters.
9. Click the dialog box's Close button in the upper-right corner.

The OLE control in your form now displays your test values:


CA-Visual Objects
File Edit View Tools Application Debug Window Help
OLE_Test in App Windows of Order Entry
OLE Test
Sales Summary
Units Sold
100.00
50.00
0.00
1
2
3
Month
Controls
txt
OLE
OLEControl1 (2 , 14) (137 x 88)

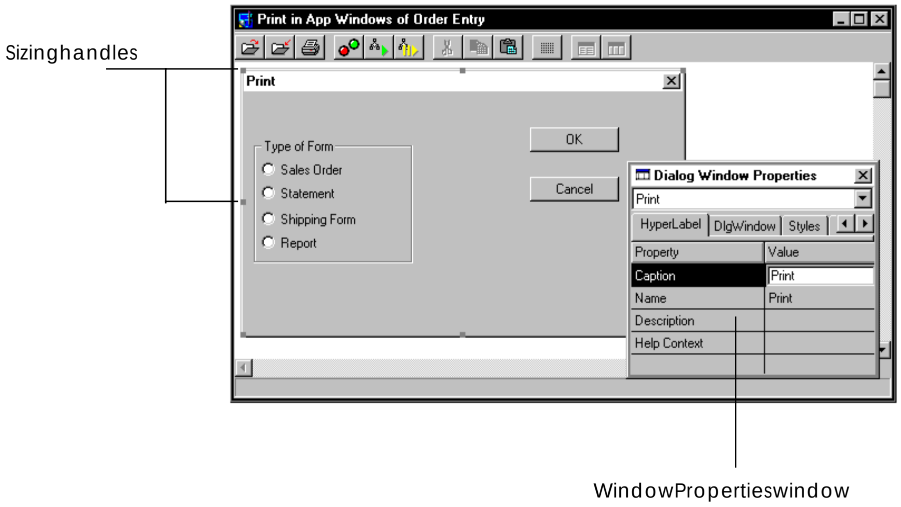
You can, of course, change your initial settings for an OCX control at any time. For example, you might want to resize it, change *x* and *y* values, edit the legend, change fonts, and so on.

Modifying a Window

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

Editing Window Properties

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



Property	Value
Caption	Print
Name	Print
Description	
Help Context	

Changing Window Properties

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

Sizing a Window

To change the size of a selected window:

1. Place the mouse pointer on one of its sizing 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 window to the desired size, and release the mouse button.

Tip: You can also use the Width and Height properties to specify the size of your window directly.

Moving a Window

You can also move a selected window to a new location in the Window Editor. This has no effect on the window itself, but it can make working with it more convenient. However, in the case of dialog windows, moving its position in the Window Editor will modify the position at which it displays at runtime.

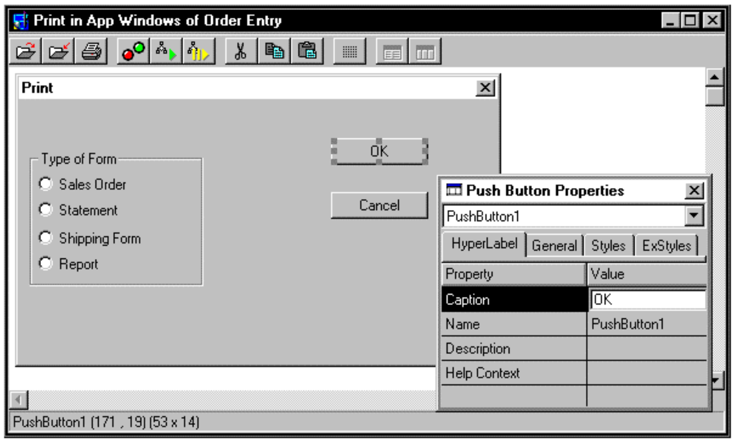
1. Place the mouse pointer anywhere on the edge of the window that is not currently occupied.

   The mouse pointer will change to a four-arrow pointer

2. Press the left mouse button and hold it down.
3. Drag the mouse to move the window to the desired location, and release the mouse button.

Editing Controls

To make any changes to a control, you must first select the control by clicking on it. When a control is selected, its perimeter is marked with several handles that are used for sizing, and the Properties window changes to show the control's properties:



Property	Value
Caption	OK
Name	PushButton1
Description	
Help Context	

PushButton1 (171 , 19) (53 x 14)

Tip: When you select a control on the window template, its name, coordinates, and size are displayed on the Window Editor status bar. For example, a push button named OK might display as "PushButton1 (171, 19) (53 X 14)," in which 171 and 19 are its respective row and column coordinates in pixels, and 53 and 14 are its respective width and height in pixels. This information is continuously updated when moving and sizing the control and can be very useful in determining if two controls line up properly or are the same size.

Changing Control Properties

To change any property associated with a selected control, click the property that you want to change in its Properties window and specify a new value.

Note: You *must* press the Enter key after modifying a control's property, otherwise that change will not adhere.

See Specifying Control Properties and Style Settings for complete information.

Selecting Controls

To select a single control, place the mouse pointer on the control, and click the left mouse button.

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 rectangle 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).

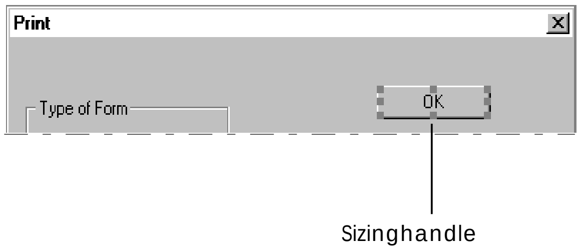
The last control selected has a different appearance; it has gray selection handles while the other controls have white selection handles. The last control selected plays a key role when you are using one of the Edit Arrange menu choices.

For example, if you have four controls, the first three 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. (See Arranging Size and Position of Controls for more information.)

Sizing a Control

Some controls, such as radio buttons, have a fixed size. Others, like list boxes and push buttons, do not so you can change their size. For example, to change the size of a push button:

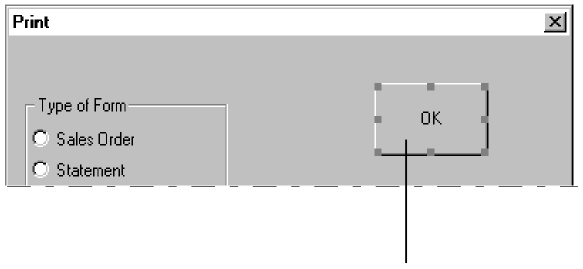
1. Place the mouse pointer on one of its handles. For example:



   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:



Tip: You can also use the Width and Height properties to specify the size of your control directly.

Moving a Control

To move a selected control (or controls) to a new location in the window:

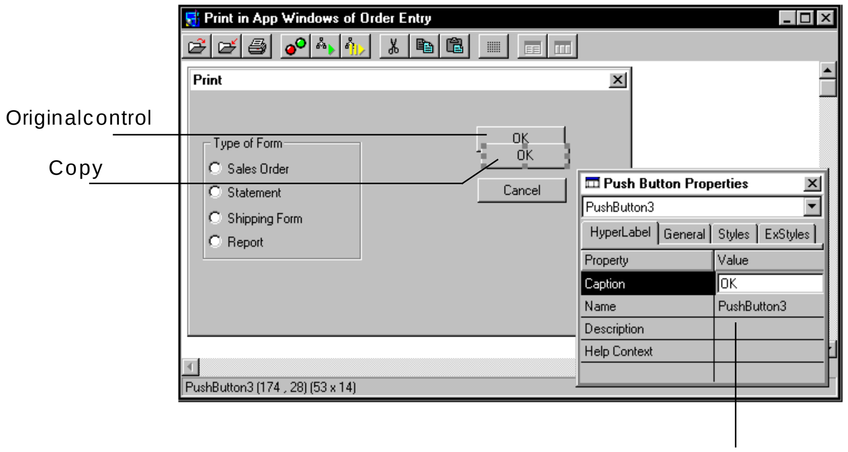
1. Place the mouse pointer anywhere on the selection.
2. Press the left mouse button and hold it down.
3. Drag the mouse to move the control(s) to the desired location, and release the mouse button.

Copying a Control

To copy a selected control (or controls) to a new location in the window:

1. Choose the Copy toolbar button.
2. Choose the Paste toolbar button.

   The new control(s) will be pasted on the window. For example:



PushButton3

Property	Value
Caption	OK
Name	PushButton3
Description	
Help Context	

Note: After copying one or more controls, you will want to move each one to its proper location and assign a new caption to it—the copy will have the same caption as the original, but a different name (for example, PushButton3).

Deleting a Control

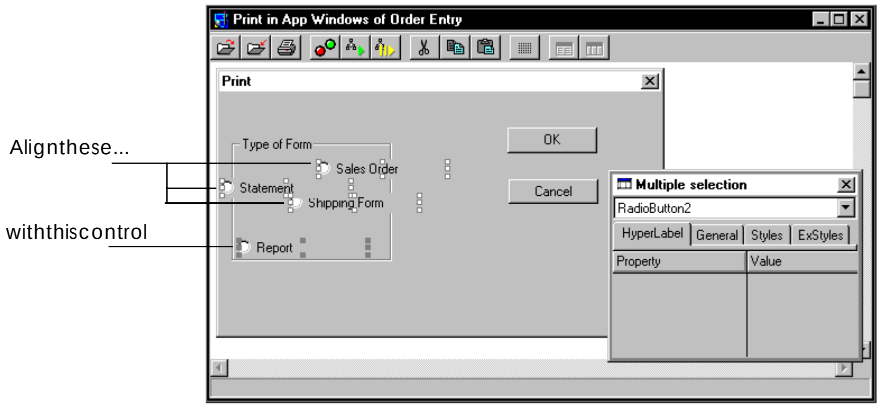
To delete a selected control (or controls) from the window, press Del or choose Edit Delete to remove the control(s).

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

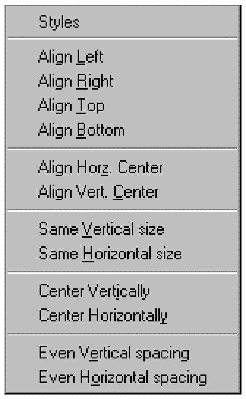
Arranging Size and Position of Controls

There are a number of useful commands on the Edit Arrange menu that will allow you to arrange several controls to have the same alignment, size, or spacing. First select several controls, making sure that the last control selected is the one that you do not wish to change. This is the control with the gray selection handles. Then select one of the Edit Arrange menu choices (or simply right click on one of the controls to see the same choices in a local pop-up menu).

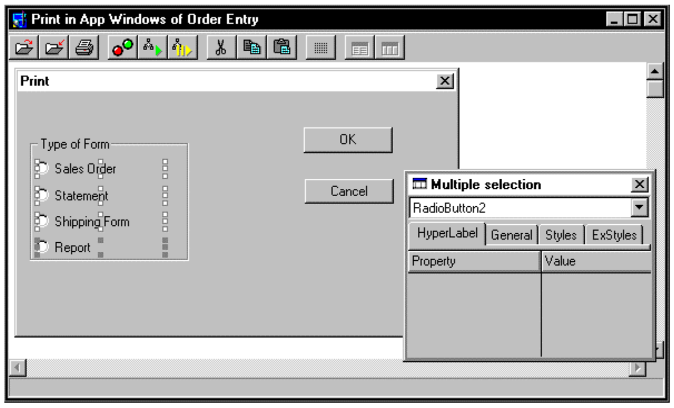
For example, suppose you wanted to modify the top three radio buttons to have the same left alignment and vertical size as the bottom radio button, and also have the same vertical spacing:



Right-click on one of the selected controls. A local pop-up menu appears:



Select the Align Left command and release the mouse. Repeat this with the Same Vertical Size command and finally select the Even Vertical Spacing command. These commands allow you to easily arrange controls on a form, like so:



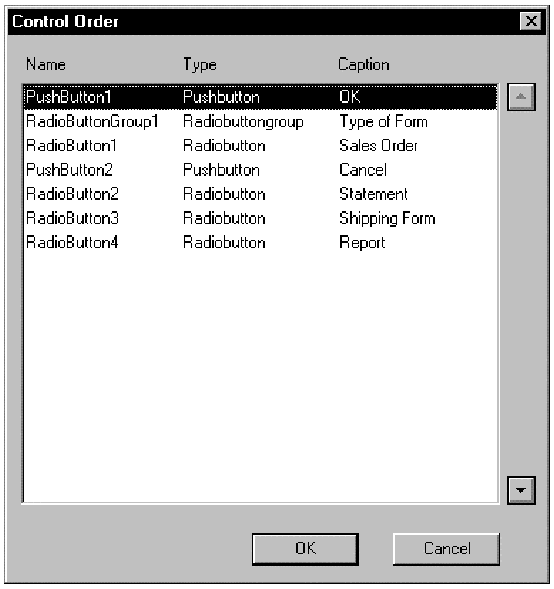
Changing Tab Order by Reordering Controls

The Edit Control Order menu command allows you to change the cursor tabbing order for a window's controls by reordering the controls in the source code. When you add controls to a window, the system automatically creates a tab order.

Note: The Tab Stop option must be selected in the appropriate Styles dialog box for each control; otherwise, the tab key cannot be used to select the control.

The initial order is based on the vertical and horizontal position of the controls. The control in the upper left-hand corner comes first, and subsequent controls are ordered based on a left-to-right, top-to-bottom progression.

For example, if you choose the Edit Control Order command for the sample Print dialog box, the Control Order dialog box appears:

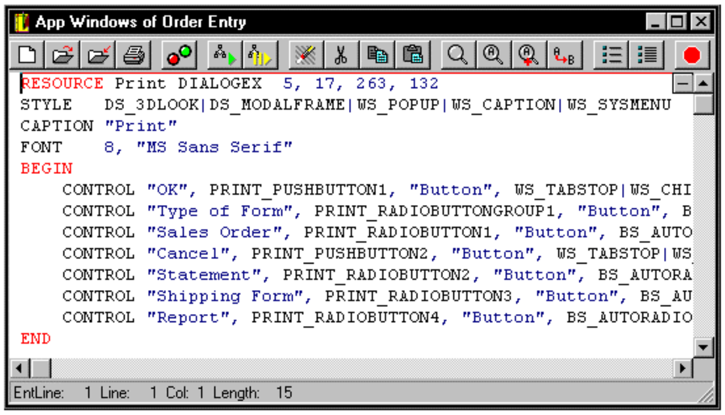


Name	Type	Caption
PushButton1	Pushbutton	OK
RadioButtonGroup1	Radiobuttongroup	Type of Form
RadioButton1	Radiobutton	Sales Order
PushButton2	Pushbutton	Cancel
RadioButton2	Radiobutton	Statement
RadioButton3	Radiobutton	Shipping Form
RadioButton4	Radiobutton	Report

Note: In CA-Visual Objects 2.7, this dialog box provides a new option, Use Mouse, which allows you to modify the tab order interactively. For more detailed information, see the online help.

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, you can access the module's source code by double-clicking on the Print resource entity in the Repository Explorer's list view pane (you must save the dialog window before any source code is generated by the Window Editor). The order in which controls are defined in the resource entity determines their tab stop order:



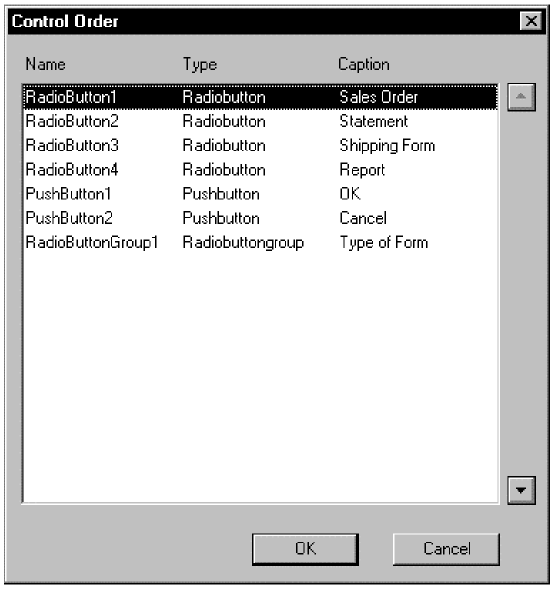
```
RESOURCE Print DIALOGEX  5, 17, 263, 132
STYLE   DS_3DLOOK|DS_MODALFRAME|WS_POPUP|WS_CAPTION|WS_SYSMENU
CAPTION "Print"
FONT    8, "MS Sans Serif"
BEGIN
    CONTROL "OK", PRINT_PUSHBUTTON1, "Button", WS_TABSTOP|WS_CHI
    CONTROL "Type of Form", PRINT_RADIOBUTTONGROUP1, "Button", B
    CONTROL "Sales Order", PRINT_RADIOBUTTON1, "Button", BS_AUTO
    CONTROL "Cancel", PRINT_PUSHBUTTON2, "Button", WS_TABSTOP|WS
    CONTROL "Statement", PRINT_RADIOBUTTON2, "Button", BS_AUTORA
    CONTROL "Shipping Form", PRINT_RADIOBUTTON3, "Button", BS_AU
    CONTROL "Report", PRINT_RADIOBUTTON4, "Button", BS_AUTORADIO
END
```

EntLine: 1 Line: 1 Col: 1 Length: 15

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, you would most likely want the cursor to move through the radio buttons in top-down order and then to the push buttons. To do this:

1. Select a control name.
2. Use the Up or Down arrow button to change its control order appropriately.
3. Repeat steps 1 and 2 until the Control Order dialog box reflects the desired order:

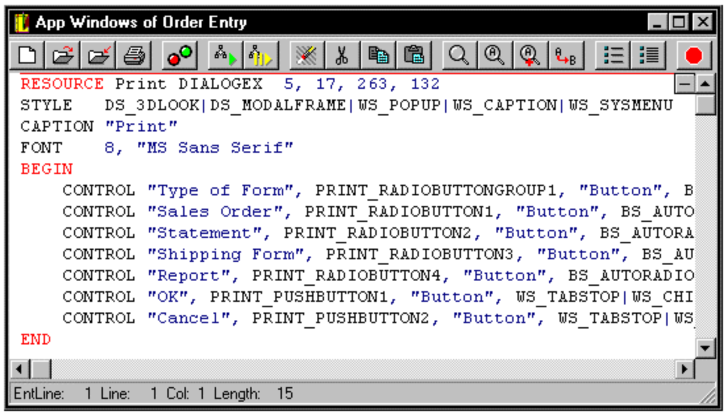


Name	Type	Caption
RadioButton1	Radiobutton	Sales Order
RadioButton2	Radiobutton	Statement
RadioButton3	Radiobutton	Shipping Form
RadioButton4	Radiobutton	Report
PushButton1	Pushbutton	OK
PushButton2	Pushbutton	Cancel
RadioButtonGroup1	Radiobuttongroup	Type of Form

4. Click OK.

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

After you save your changes in the Window Editor, the source code will reflect the reordered controls and tab stops:



```
RESOURCE Print DIALOGEX  5, 17, 263, 132
STYLE   DS_3DLOOK|DS_MODALFRAME|WS_POPUP|WS_CAPTION|WS_SYSMENU
CAPTION "Print"
FONT    8, "MS Sans Serif"
BEGIN
    CONTROL "Type of Form", PRINT_RADIOBUTTONGROUP1, "Button", B
    CONTROL "Sales Order", PRINT_RADIOBUTTON1, "Button", BS_AUTO
    CONTROL "Statement", PRINT_RADIOBUTTON2, "Button", BS_AUTORA
    CONTROL "Shipping Form", PRINT_RADIOBUTTON3, "Button", BS_AU
    CONTROL "Report", PRINT_RADIOBUTTON4, "Button", BS_AUTORADIO
    CONTROL "OK", PRINT_PUSHBUTTON1, "Button", WS_TABSTOP|WS_CHI
    CONTROL "Cancel", PRINT_PUSHBUTTON2, "Button", WS_TABSTOP|WS
END
```

Caution! *Microsoft Windows requires you to have at least one control present on any window in order to use the Tab key. If you push Tab in an empty window or a window containing uneditable controls, your application will freeze, requiring you to reboot your computer. This is a behavior of Windows that is not unique to CA-Visual Objects applications.*

Printing Windows

To print a screen shot of the window you are designing, click the Print toolbar button. Only a copy of the window form itself—and not the Window Editor containing it—is printed.

Using the Window in an Application

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

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

Associating a window with either of these events is easy. Simply use the form entity name in the ButtonClick event property of the push button or the Event Name property of the menu item. Then, when you select the push button or menu item, the window will be displayed. See **Chapter 5: Using the Menu** Editor for more information on associating a window with a menu selection.

Chapter 5

Using the Menu Editor

Most windows designed for use in a GUI application have an associated menu and / or toolbar, defining the available choices for the window and the actions associated with those choices. In a GUI application, all logic, navigation, and database operations depend on visual controls, so naturally menus make up a large part of the user interface.

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

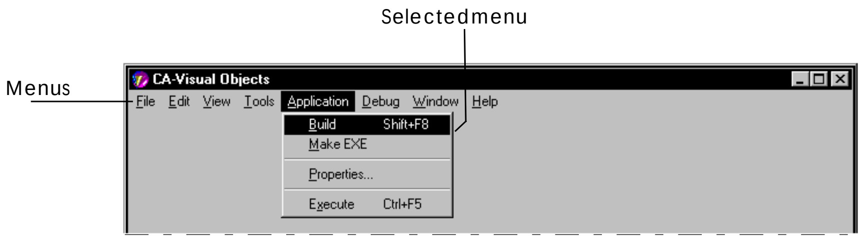
This chapter describes how to use the Menu Editor, including how to:

- Create a custom menu structure, complete with toolbar
- Add predefined, standard menus at the touch of a button
- Modify an existing menu structure
- Print using the Menu Editor
- Use the menu you have created in an application

Important! *The GUI Classes and System Classes libraries must be included in the search path of your application in order to use a menu generated by the Menu Editor. See **Setting the Search Path** in "Using the Repository Explorer" for more information.*

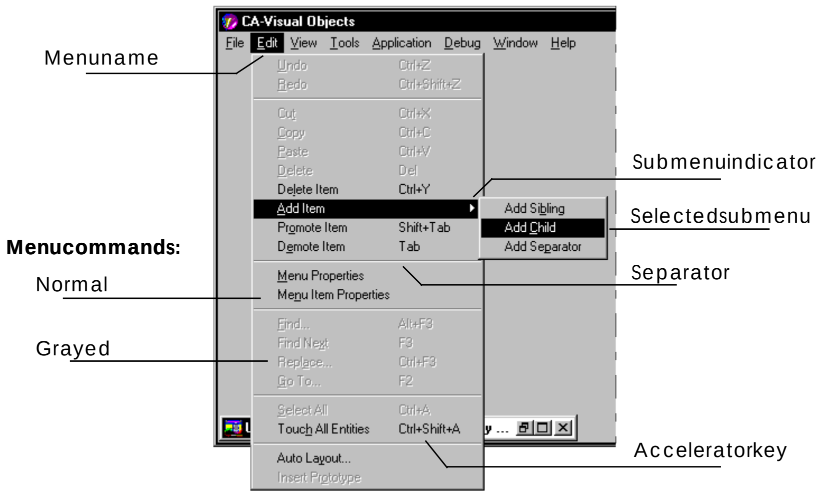
Menu Terms

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 CA-Visual Objects 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 in CA-Visual Objects are illustrated below:



Note: A toggle menu command, whose active state is indicated by a check mark ( ), is not shown in the above diagram.

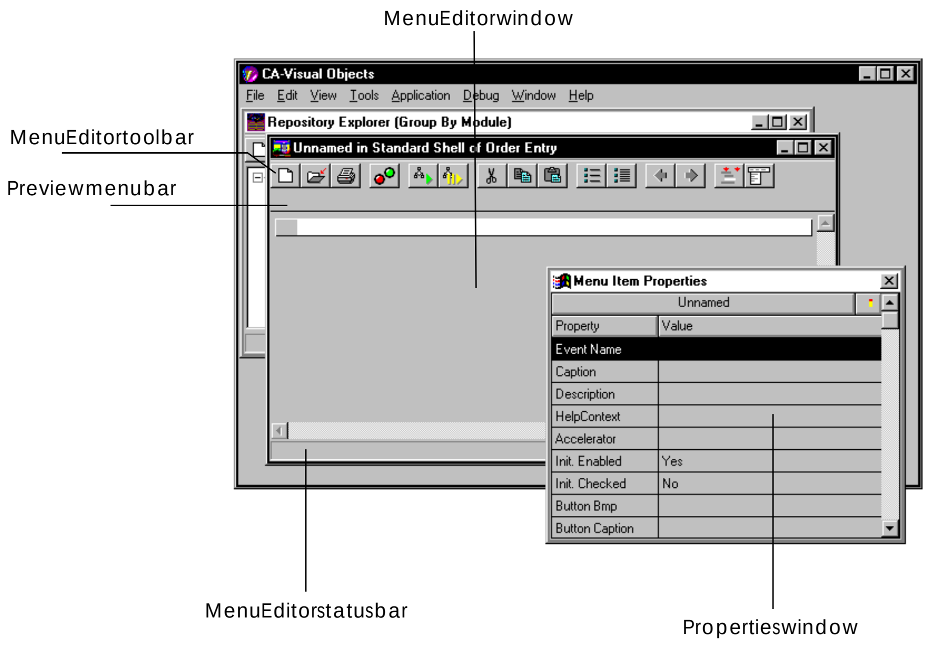
Tip: You can also use bitmaps as menus or menu items. For example, instead of the text "Help" for a Help menu, you might use a question mark bitmap in the menu bar. Refer to the Menu class in the online help system for more information about adding bitmaps to menus.

Workspace Overview

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

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

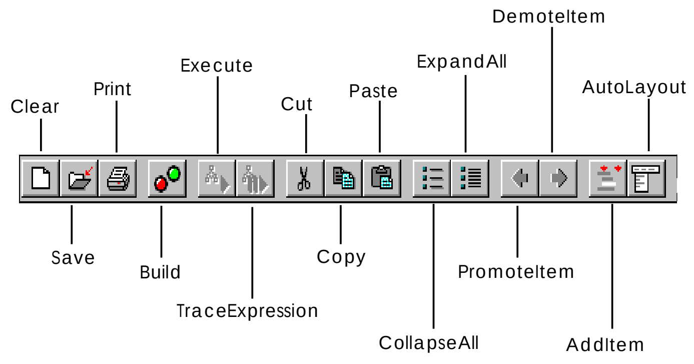
The Menu Editor has its own toolbar, status bar, an associated Properties window, and an area for previewing defined menus:


MenuEditorwindow
MenuEditortoolbar
Previewmenubar
CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Unnamed in Standard Shell of Order Entry
Menu Item Properties
Unnamed
MenuEditorstatusbar
Propertieswindow

Property	Value
Event Name	
Caption	
Description	
HelpContext	
Accelerator	
Init. Enabled	Yes
Init. Checked	No
Button Bmp	
Button Caption	

The Toolbar

The Menu Editor toolbar contains the following buttons:

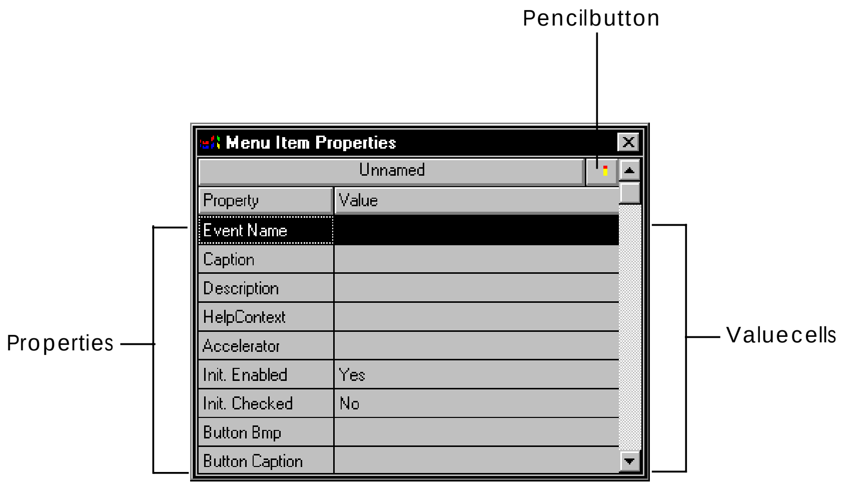


The Clear, Save, Print, Cut, Copy, Paste, Collapse All, Expand All, Promote Item, Demote Item, Add Item, and Auto Layout buttons are discussed in this chapter. See **Chapter 2: Working in the** Desktop for information about Build and Execute and **Chapter 11: Debugging Your Applications** for details on Trace Expression.

Tip: For a quick description of each toolbar button, look at the tooltip windows as the mouse pointer passes over the buttons.

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:



Property	Value
Event Name	
Caption	
Description	
HelpContext	
Accelerator	
Init. Enabled	Yes
Init. Checked	No
Button Bmp	
Button Caption	

The *Property* column lists all the 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, a keyword for use in a context-sensitive help system, or an initial state (such as checked or enabled).

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 use it, simply highlight a property by clicking on it, then use one of the following techniques for specifying its value:

- Click in the value cell and enter a new value by typing directly into a single-line edit control
- Choose a new value from a drop-down list by clicking on the down arrow button
- Fill in a corresponding dialog box

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 Show 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 just below the Menu Editor's menu bar. The preview menu bar is partially operational 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.

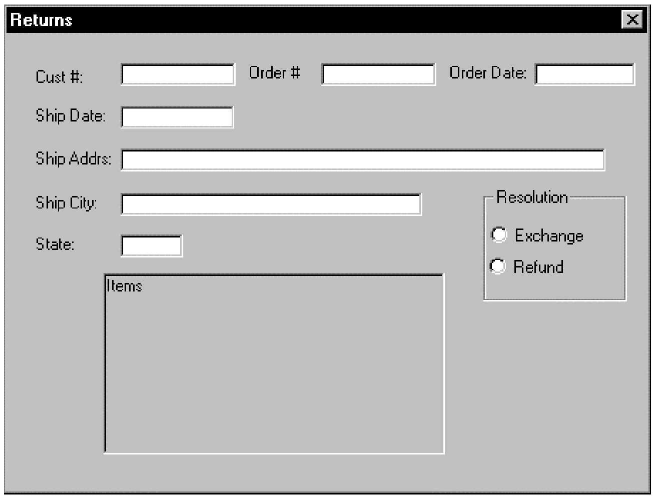
Defining a Menu

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, specifying a hierarchy among them, and specifying properties for the menu structure and its entries
- Create a predefined menu using the Auto Layout feature
- Modify an existing menu structure

Creating a Menu

Suppose you decide to extend the Order Entry sample application by adding a Returns data window, shown below, for resolving returned items. (See **Sub-Data Windows** in "Creating Data-Aware Windows" later in this guide for detailed information about creating the Returns data window.)



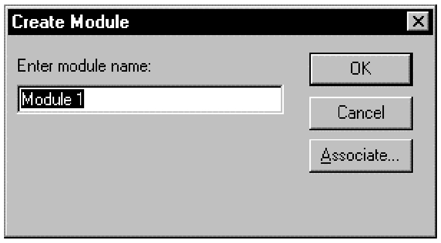
To create a menu for a window form (for example, the Returns data window), perform the following basic steps:

1. From within the Repository Explorer, select the specified application (for example, Order Entry) and module.

   For Order Entry, you will need to create a new module; otherwise, you can skip steps 2 and 3.

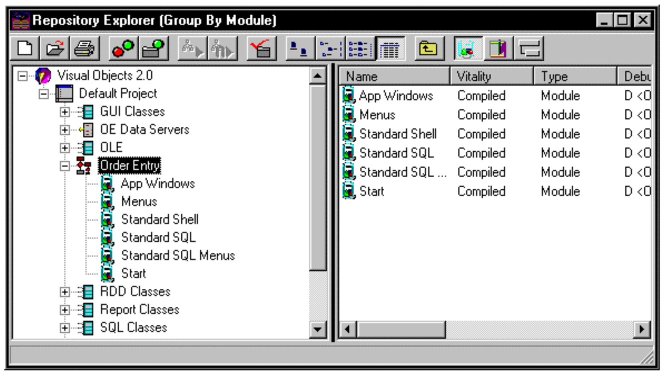
2. Click the New Module toolbar button.

   The Create Module dialog box appears:



3. Type **Menus** in the Enter Module name edit control, and choose OK.

The Menus module is added to Order Entry in the Repository Explorer's tree view pane:

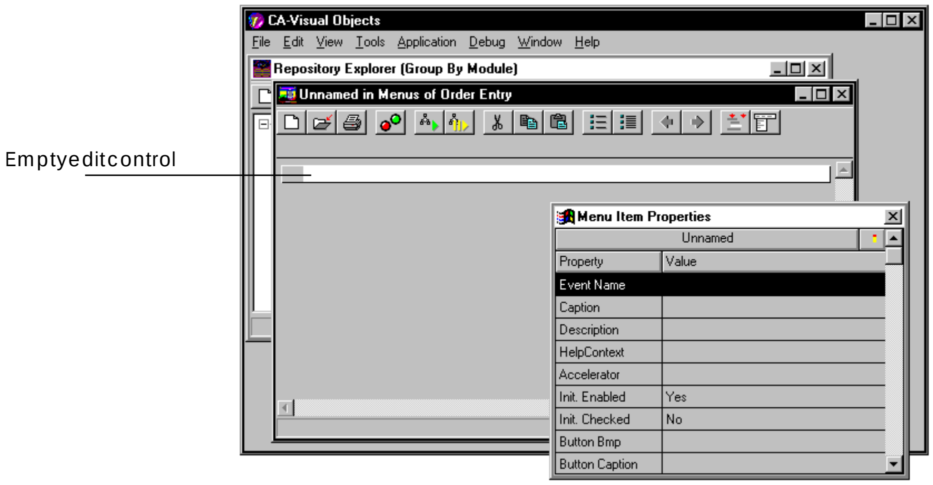


Name	Vitality	Type	Debu
App Windows	Compiled	Module	D <0
Menus	Compiled	Module	D <0
Standard Shell	Compiled	Module	D <0
Standard SQL	Compiled	Module	D <0
Standard SQL ...	Compiled	Module	D <0
Start	Compiled	Module	D <0

4. Highlight the Menus module in the tree structure, and then start the Menu Editor.

   Like all CA-Visual Objects tools, the Menu Editor is accessed using the Tools menu or the New Entity toolbar button.

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


Emptyeditcontrol
CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Unnamed in Menus of Order Entry
Menu Item Properties
Unnamed

Property	Value
Event Name	
Caption	
Description	
HelpContext	
Accelerator	
Init. Enabled	Yes
Init. Checked	No
Button Bmp	
Button Caption	

5. Specify a name and, optionally, a toolbar and other properties for the menu structure.

   Note: In this version of CA-Visual Objects, you can create flat toolbars with bands as separators, like those in the Internet Explorer and other Windows products. For detailed information, see the online help.

   See Specifying Menu Properties for more detailed information.

6. 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 detailed information.

7. Promote / demote any entry necessary to create the hierarchy.

   See Creating the Hierarchy for more information.

8. Optionally preview the menu bar and each of the menus that you have defined.

   See Previewing the Menu Bar for more information.

9. 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 details.

10. 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.

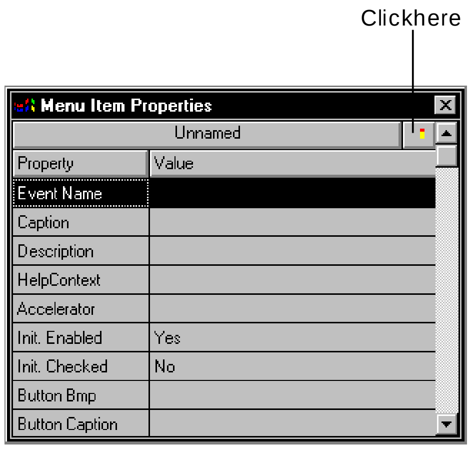
Specifying Menu Properties

Before you get started with the menu design for the Returns data window, you may want to go ahead and define a few properties for the menu structure, including its name and the specifications for its toolbar (if you are planning to have one).

Menu Name

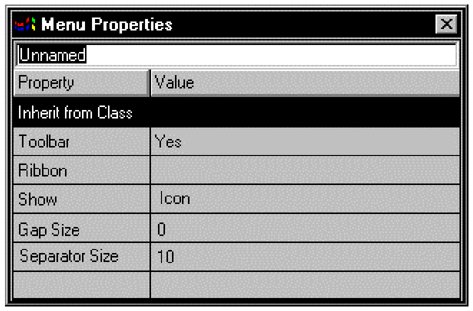
At a minimum, you are required to supply a name for the menu structure before you can save it. To do this:

1. Click on the Pencil button to the right of the 3-D bar titled "Unnamed" at the top of the Menu Item Properties window:



Property	Value
Event Name	
Caption	
Description	
HelpContext	
Accelerator	
Init. Enabled	Yes
Init. Checked	No
Button Bmp	
Button Caption	

The Menu Item Properties window becomes the Menu Properties window:



Property	Value
Inherit from Class	
Toolbar	Yes
Ribbon	
Show	Icon
Gap Size	0
Separator Size	10

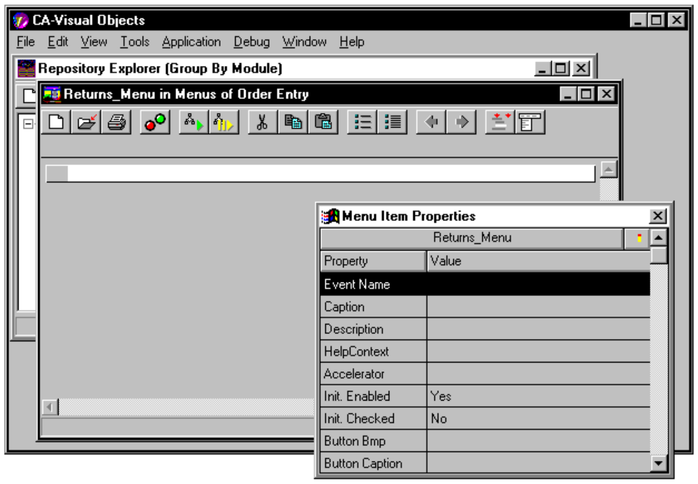
2. Enter a name in the edit control that appears (for example, **Returns Menu**).

   The name that you enter will be used in the source code generated by the Menu Editor to create a class entity, a binary menu entity, and any resource entities needed to create the menu structure, so it must not conflict with other entity names in your application.

   Note: Blank spaces within the name will be converted to underscores.

3. Press Enter.

   "Returns_Menu" now appears in the title bar of the Menu Editor, as well as in the 3-D bar at the top of the Menu Item Properties window:


CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Returns_Menu in Menus of Order Entry
Menu Item Properties
Returns_Menu

Property	Value
Event Name	
Caption	
Description	
HelpContext	
Accelerator	
Init. Enabled	Yes
Init. Checked	No
Button Bmp	
Button Caption	

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, described below, is optional.

Note: As mentioned earlier, you can create flat toolbars with bands as separators in this version of CA-Visual Objects. Consequently, there is a new property, Use Bands, for menus. Additionally, another property, Toolbar, has an additional value, Flat Toolbar. For detailed information about new and updated menu properties, see the online help.

Inherit from Class	Select the Menu class or subclass from which the specified menu will inherit its basic characteristics. Valid choices are: <Auto>, EMPTYSHELLMENU, SYSTEMMENU, STANDARDSHELLMENU, and all classes derived from the Menu class that are in the current search path. The default value is <Auto>, which means a menu generates an instance of the CA-Visual Objects Menu class. **Note:** The following properties cannot be changed if Toolbar is set to No.
Toolbar	Specify whether a toolbar will be associated with this menu structure. This is a Yes/No option that you change by selecting a value from a drop-down list. The default value of Yes indicates that a toolbar will be displayed.
Ribbon	If using a predefined ribbon for the toolbar, specify the name of the ribbon entity. (See **Chapter 10: Using the Image** Editor for detailed information about ribbons and other image object entities.)

Show	Choose how the toolbar should be displayed. The options are: Text, Icon, or Text and Icon. The default is Icon. See the Button Bmp property under Specifying Menu Item Properties later in this chapter for more information.
Gap Size	Enter the distance in points between individual buttons. The default is zero (0).
Separator Size	Enter the width in points between groups of buttons. The default is ten (10). See the Button Pos property under Specifying Menu Item Properties later in this chapter for more information.

These options specify whether you want to include a toolbar and, if so, what its general characteristics are. See the Button Bmp and Button Pos properties under Specifying Menu Item Properties later in this chapter for more information on how to create the actual toolbar buttons.

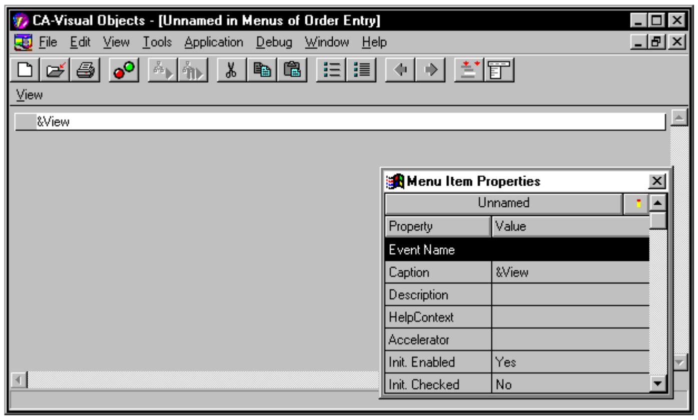
Adding Menus and Menu Items

The remaining Menu Editor features are demonstrated using a simple menu structure for the Returns data window. In it, a single View menu allows the user to view customer records, sales orders, inventory items, and shipping records, and to print several types of reports using a submenu.

Tip: Each menu and menu item in a Windows application typically features a single underlined letter that indicates how to select it with the keyboard. For example, the “F” in the CA-Visual Objects File menu and the “x” in the Exit command are 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 underlined with an ampersand (&).

To create the View menu for our sample application:

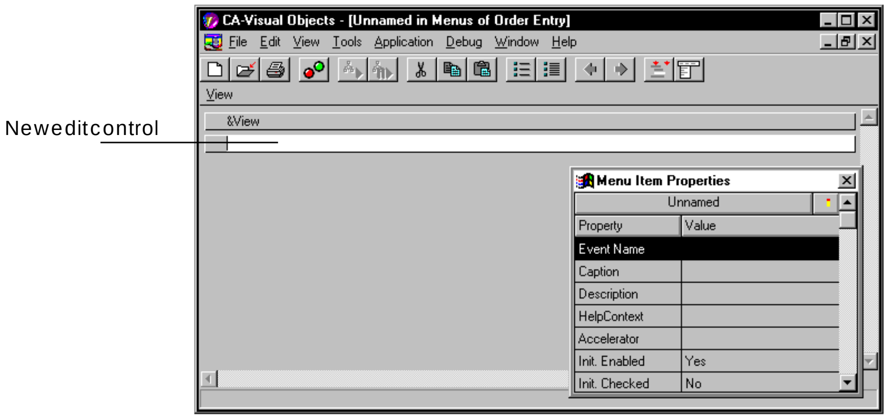
1. Click on the first empty edit control, and type **&View**:


CA-Visual Objects - [Unnamed in Menus of Order Entry]
File Edit View Tools Application Debug Window Help
View
&View
Menu Item Properties
Unnamed

Property	Value
Event Name	
Caption	&View
Description	
HelpContext	
Accelerator	
Init. Enabled	Yes
Init. Checked	No

2. Press Enter.

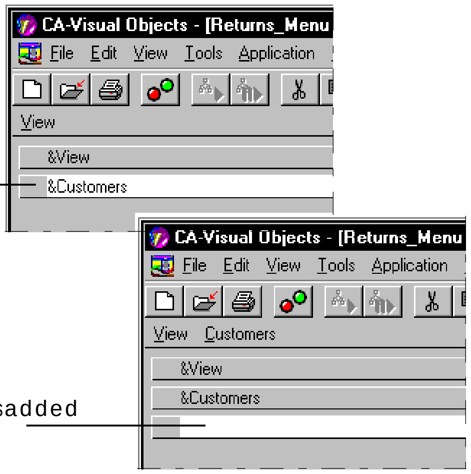
   After the Menu Item Properties window is updated using "&View" as the caption for this menu, a new edit control appears:


Neweditcontrol
CA-Visual Objects - [Unnamed in Menus of Order Entry]
File Edit View Tools Application Debug Window Help
View
&View
Menu Item Properties
Unnamed

Property	Value
Event Name	
Caption	
Description	
HelpContext	
Accelerator	
Init. Enabled	Yes
Init. Checked	No

3. Type the next entry (for example, **&Customers**), and press Enter.

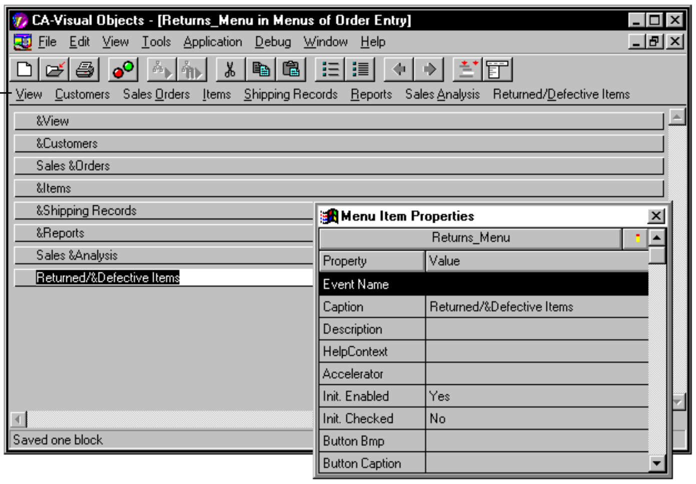
Again a new line appears:


Type next menu here and press Enter...
A new edit control is added
CA-Visual Objects - [Returns_Menu
File Edit View Tools Application
View
&View
&Customers
CA-Visual Objects - [Returns_Menu
File Edit View Tools Application
View Customers
&View
&Customers

4. Repeat step 3 six times, using the text **Sales &Orders**, **&Items**, **&Shipping Records**, **&Reports, Sales &Analysis**, and **Returned/&Defective Items**. (Do not press Enter after the last line.)

5. Click the Save toolbar button to save your work so far.

When you are finished, your menu structure should look something like this:


Preview menu bar
CA-Visual Objects - [Returns_Menu in Menus of Order Entry]
File Edit View Tools Application Debug Window Help
View Customers Sales Orders Items Shipping Records Reports Sales Analysis Returned/Defective Items
Menu Item Properties
Returns_Menu

&View
&Customers
Sales &Orders
&Items
&Shipping Records
&Reports
Sales &Analysis
Returned/&Defective Items

Saved one block

Property	Value
Event Name	
Caption	Returned/&Defective Items
Description	
HelpContext	
Accelerator	
Init. Enabled	Yes
Init. Checked	No
Button Bmp	
Button Caption	

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

Note: In this version of CA-Visual Objects, the preview menu bar is located just below the Window Editor’s menu bar.

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, View will act as the topmost menu, containing five menu items (four commands and a submenu). The submenu will contain two 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 Returns data window:

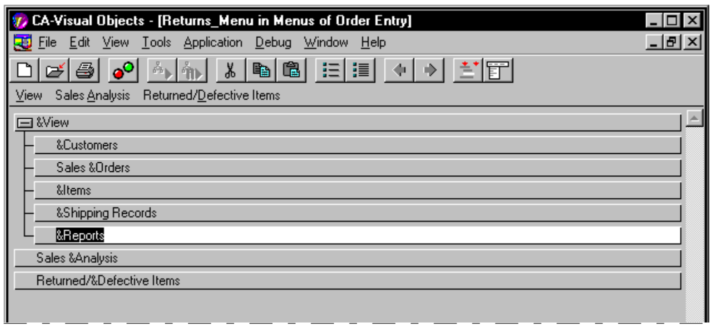
1. Select the Customers entry by clicking on it in the Menu Editor window, and click the Demote Item toolbar button.

   The Menu Editor immediately indents the Customers entry so that it is one level below View, making it a *child*, or menu item, of View. The Customers entry is also removed from the preview menu bar, which is continually updated as you work, providing immediate visual feedback.

   Note: As you can see, the Menu Editor represents the hierarchy of the menu in a tree-like structure similar to that in the Repository Explorer. The +/– button added to the left of the View entry is a Collapse/Expand toggle button; 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 four more times to demote the Sales Orders, Items, Shipping Records, and Reports entries.

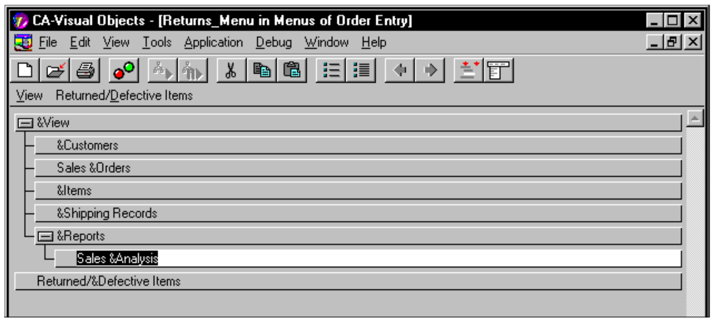
   All demoted entries are now menu items on the View menu:


CA-Visual Objects - [Returns_Menu in Menus of Order Entry]
File Edit View Tools Application Debug Window Help
View Sales Analysis Returned/Defective Items
&View
&Customers
Sales &Orders
&Items
&Shipping Records
&Reports
Sales &Analysis
Returned/&Defective Items

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

3. Place the cursor in the Sales Analysis entry and then click the Demote Item toolbar button *twice*.

   The menu structure is updated as follows:



CA-Visual Objects - [Returns_Menu in Menus of Order Entry]

File Edit View Tools Application Debug Window Help
View Returned/Defective Items

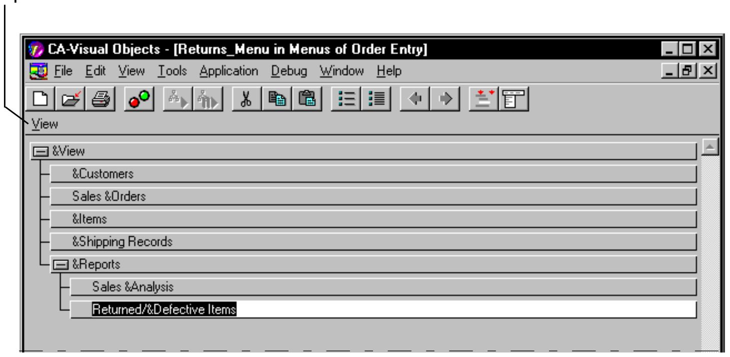
```
&View
  &Customers
  Sales &Orders
  &Items
  &Shipping Records
  &Reports
    Sales &Analysis
Returned/&Defective Items
```

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

4. Repeat step 3 one more time for the Returned / Defective Items entry.

   The menu structure now looks as follows:

Updatedpreviewmenubar



CA-Visual Objects - [Returns_Menu in Menus of Order Entry]

```
&View
  &Customers
  Sales &Orders
  &Items
  &Shipping Records
  &Reports
    Sales &Analysis
    Returned/&Defective Items
```

   Note that the preview menu bar now correctly displays only the View menu.

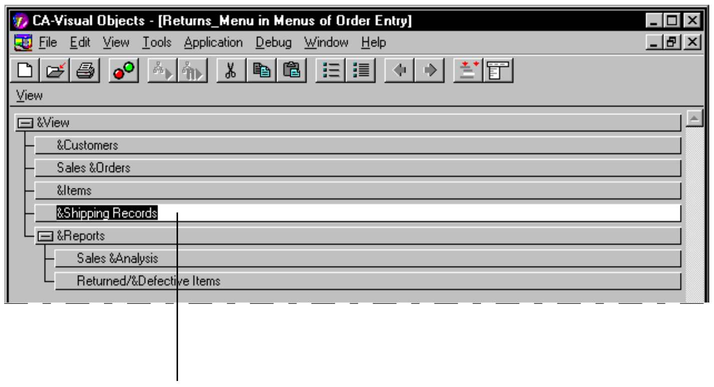
Tip: If necessary, move the Properties window out of the way by clicking on its title bar and dragging it to a new position.

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, to separate Reports from the other entries in our View menu:

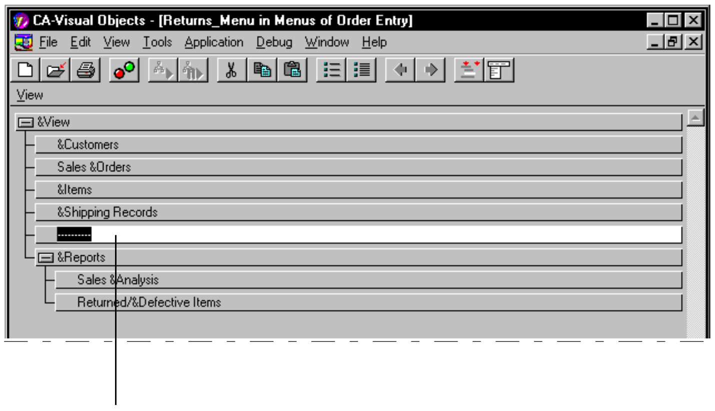
1. Click on the &Shipping Records entry:


CA-Visual Objects - [Returns_Menu in Menus of Order Entry]
File Edit View Tools Application Debug Window Help
View
&View
&Customers
Sales &Orders
&Items
&Shipping Records
&Reports
Sales &Analysis
Returned/&Defective Items

Clickhere...

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

   A new edit control with a predefined separator is added to the menu structure:


CA-Visual Objects - [Returns_Menu in Menus of Order Entry]
File Edit View Tools Application Debug Window Help
View
&View
&Customers
Sales &Orders
&Items
&Shipping Records

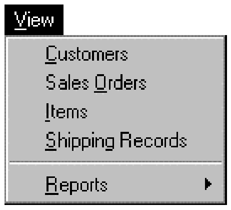
&Reports
Sales &Analysis
Returned/&Defective Items

Separator

3. Click the Save toolbar button to save the menu structure.

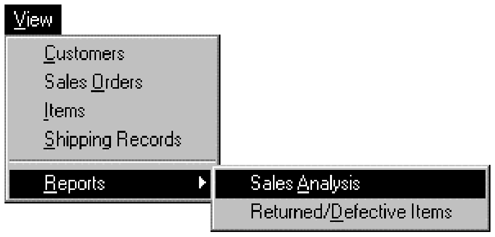
Previewing the Menu Bar

After completing the steps outlined in the previous section, you have noticed that the Menu Editor's preview menu bar has been reduced to just the View 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 View to display the following:



Note that the Menu Editor has added the underlining for the letters indicated using the "&" and the ▶ symbol to indicate a submenu for the Reports menu item.

If you then click on Reports, 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.

Note: Refer to the online help for detailed information about new and updated menu item properties.

Event Name

Enter the name of an event to be invoked when this menu item is selected. At runtime, the system will first look for a method with this name. If no method can be found, it will look for a Window or ReportQueue class. If none of these can be found, the menu selection will do nothing. (See "GUI Classes" in the *Programmer's Guide* for a complete discussion of predefined menu events.)

Note: Event Name cannot be defined for a menu bar item or a menu item that defines a submenu. Selecting this property in these cases will produce an error message.

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, an ampersand (to underline a letter and enable an equivalent key command), and the underscore character (_). With the exception of the ampersand, all characters will appear exactly as typed.

Note: Accelerator key name, check mark, and the ▶ symbol to indicate a submenu are not part of the caption. These indicators are added automatically by the system. If, on the other hand, you want to use the standard ellipsis (...) indicator for a menu item that opens a dialog box, you must include the ellipsis as part of the caption.

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.

Description

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

HelpContext

Enter a unique keyword for the entry that can be used to identify it in a context-sensitive help system. This property can consist of letters, digits, and the underscore character (_), but it cannot contain any other characters, including blanks or punctuation. (For more information on creating a help file using this property, refer to “GUI Classes” in the *Programmer’s Guide.*)

Accelerator

To define an *accelerator* (or shortcut key), press the corresponding keys for a modifier and the accelerator key from within this property’s Value column. For example, press the Shift plus the F8 key to specify **Shift+F8** as a menu item’s accelerator.

Note: To activate the edit control in the Value column, click the Accelerator property column first. Also, for this property to take effect, do *not* press Enter after entering the accelerator key. Instead, move the cursor and click outside the Value column.

At runtime, the accelerator 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 accelerator as a shortcut to selecting the menu and then choosing 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 by selecting a value from a drop-down list. The default value of Yes indicates enabled.

Note: 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.

Tip: The Menu class has methods, DisableItem() and EnableItem(), that you can use as part of the event name method to change the status of a menu item.

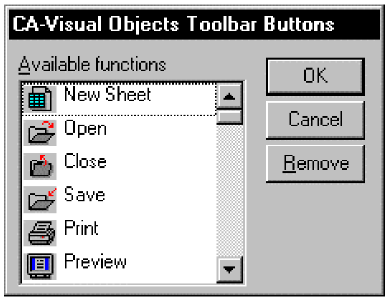
Init. Checked

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 by selecting a value from a drop-down list. The default value of No indicates that the entry will not be checked.

Tip: The Menu class has methods, CheckItem() and UnCheckItem(), that you can use as part of the event name method to change the status of a menu item.

Button Bmp

Define a toolbar button for an entry using the Toolbar Buttons dialog box:



The Available Functions list box allows you to choose an icon/text pair to add to the toolbar. To define a toolbar button, you are required to choose an item from this list box.

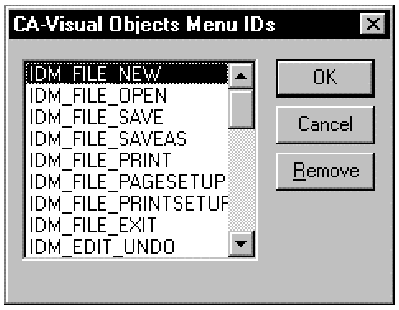
OK adds the selected button to the toolbar and closes the dialog box. The Remove button removes a previously defined button from the toolbar and closes the dialog box.

The Show property, described under Specifying Menu Properties earlier in this chapter, controls whether the text, icon, or both are displayed on the toolbar.

Button Caption	Enter a caption for the specified toolbar button. The default is the text that is paired with the selected icon in the Available Functions list box of the Toolbar Buttons dialog box.
Button ToolTip	Enter descriptive text for the specified button's tooltip control. Tooltip controls are small pop-up windows that display relevant text when the mouse moves over the toolbar button. The default is the predefined text that is associated with the selected icon in the Available Functions list box of the Toolbar Buttons dialog box.
Button Pos	Enter a number indicating the position of the button on the toolbar. When you choose a Button Bmp, this property is automatically filled in using the next available position on the toolbar. You can either leave it as is, or start a new group of buttons by entering a number that is one greater than the value displayed. **Note:** The Gap Size and Separator Size properties, described under Specifying Menu Properties earlier in this chapter, control how far apart toolbar buttons and groups are displayed.

ID

Choosing this property activates the Menu IDs dialog box, shown below, from which you can choose one of several internally-defined resource IDs for a menu item:



This property allows you to assign the same ID to the same menu items in different menu structures, thereby reducing the Windows resources necessary to display the menu.

If you do not specify an ID using this property, the Menu Editor will generate a unique ID number and name for it.

Note: Unless the Toolbar property described under Specifying Menu Properties earlier in this chapter is Yes, no toolbar will be displayed with the menu.

Generating Code

When you save a menu and its toolbar (optional), the Menu Editor automatically generates:

- A binary menu entity

  You can double-click on this entity in the Repository Explorer's list view pane to begin editing it with the Menu Editor.
- A menu subclass using the name of the menu
- Other default source code that can be modified

  This includes resources for the menu and accelerators, and a subclass for accelerators.

Note: In this version of CA-Visual Objects, the Menu Editor does not generate local variables if not required. This is a behind-the-scenes time saver.

Adding Predefined Menus

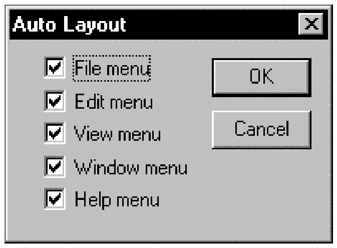
The Menu Editor allows you to quickly add standard, predefined File, Edit, View, Window, and Help menus. Each menu has default menu items and corresponding toolbar buttons.

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.

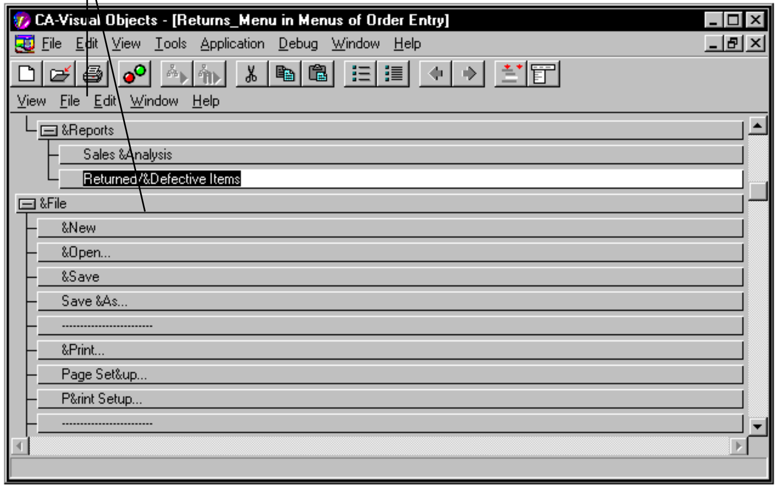
   The Menu Editor Auto Layout dialog box appears with all menus selected:

2. Make sure that all menus you want to add are checked.
3. Choose OK.

For example, if the File, Edit, Window, and Help standard menus are added to the end of the Returns_Menu created earlier for the Order Entry sample application, the result would be:

Standardmenusaddedtopreviewmenubarandmenustructure


CA-Visual Objects - [Returns_Menu in Menus of Order Entry]
File Edit View Tools Application Debug Window Help
View File Edit Window Help
&Reports
Sales &Analysis
Returned/&Defective Items
&File
&New
&Open...
&Save
Save &As...
&Print...
Page Set&up...
P&rint Setup...

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 add 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 other than 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 Properties window will change to show the properties for the menu structure, and you can change any property you want. See Specifying Menu Properties.

Editing Menus and 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.

Changing 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.

Adding a Sibling

To add an entry at the *sibling* (or same) level after the currently selected entry:

1. Select the Add Item toolbar button.

   Alternatively, choose the Edit Add Item command and select Add 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.

Adding a Child

To add an entry at the *child* (or submenu) level after the currently selected entry:

1. Choose the Edit Add Item command and select Add 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.

Deleting

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

Use the Print toolbar button to print copies of your menu designs.

Using the Menu in an Application

Menus that you create using the Menu Editor must be attached to a window form of some type. The easiest way to make this connection (and to create the form itself) is using the Window Editor. There, the menu name is a property of the form that you can choose from a drop-down list. See Chapter 5: Using the Menu Editor and **Chapter 8: Creating Data-Aware Windows** for more information.

Tip: The Standard Application contains two predefined binary menu entities: EmptyShellMenu and StandardShellMenu. These are located in a separate module, and you can easily customize them using the Menu Editor to meet the specifications of your application.

Chapter

6

Using the Source Code Editor

The Source Code Editor provides a powerful environment for writing and editing code, drawing much of its strength from its close integration with the CA-Visual Objects *repository*. It can even complete a partially entered statement with the parameters defined in the prototype of the function or method. (This a very useful feature if you have forgotten the correct syntax!)

This chapter describes how to use the Source Code Editor, including how to:

- Create new entities
- Modify selected entities within a module
- Modify all entities within a module
- Import source code from a text file for editing
- Use some of its special editing features, such as inserting prototypes, finding matching tokens, and presetting breakpoints for debugging sessions
- Print copies of your source code

Workspace Overview

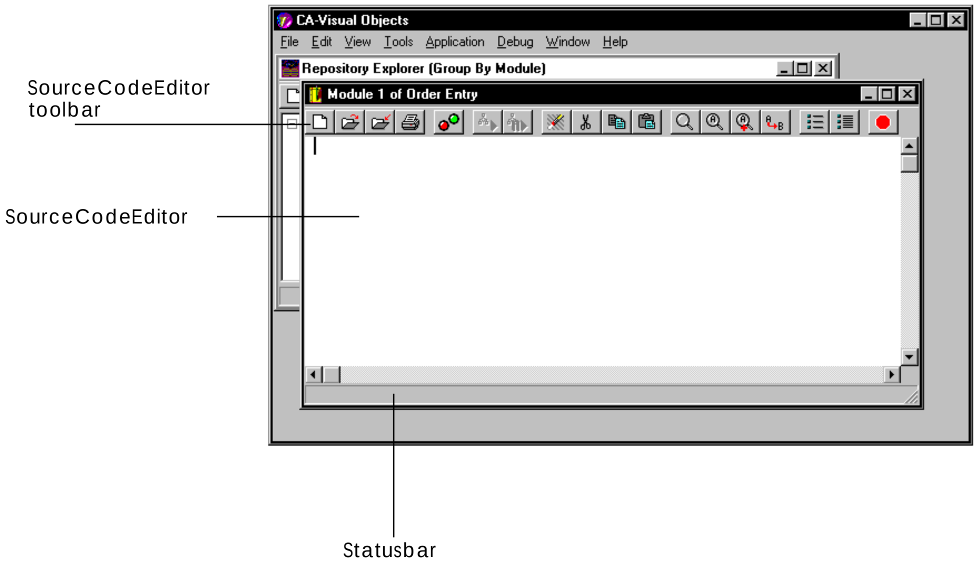
The Source Code Editor is the primary workspace in the IDE for viewing, creating, and modifying source code entities, such as functions, classes, methods, and resources. When you are in the Source Code Editor, you can:

- Perform standard editing features such as cut, copy, paste, delete, search for text, replace text, undo, and redo using standard Windows techniques for any entity currently loaded
- Create new entities
- Go immediately to a specific entity for editing
- Insert a prototype for incomplete statements
- View a source code entity’s definition while editing and jump to its prototype in the appropriate class library
- Import and export text-based files
- Print the source code for all entities currently loaded

- Save or discard the changes that you have made

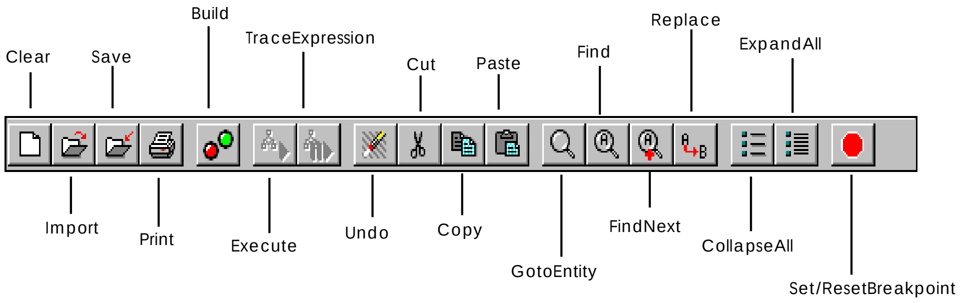
Note: As mentioned earlier in Setting System-Wide Options in the "Working in the Desktop" chapter, there are several new options available for the Source Code Editor: Use Spaces for Tabs, Show Parameter Tips, and Activate IDE on Break. Additionally, the Automatic Method Insertion option is now enabled by default. For more detailed information, see the online help.

The Source Code Editor has its own toolbar and status bar. When first loaded for a new source code entity in a new module, an empty Source Code Editor window is displayed:



The Toolbar

The Source Code Editor toolbar contains the following buttons:



All of the above buttons except Build, Execute, and Trace Expression are described in this chapter. See **Chapter 2: Working in the** Desktop for information about Build and Execute and **Chapter 11: Debugging Your Applications** for details on Trace Expression.

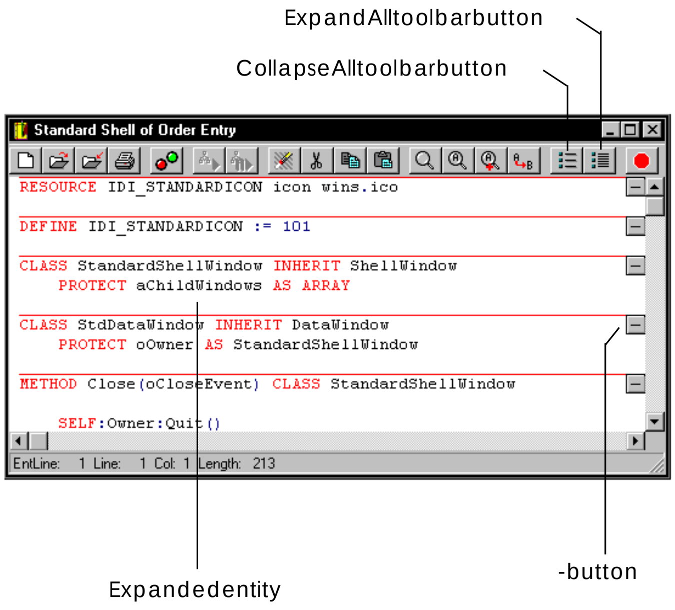
Tip: For a quick description of any toolbar button, simply point to it—a tooltip window with descriptive text pops up right next to the button.

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 an associated external source file. It also indicates whether the contents of the editor have been changed or not.

Collapsing/Expanding

When you are using the Source Code Editor, you will notice that each entity displayed in the editor has a – button to the right of its declaration statement as illustrated below:

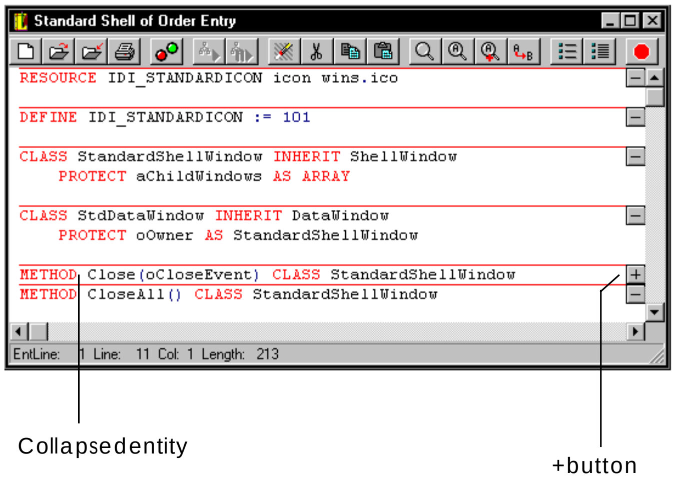


```
RESOURCE IDI_STANDARDICON icon wins.ico

DEFINE IDI_STANDARDICON := 101

CLASS StandardShellWindow INHERIT ShellWindow
    PROTECT aChildWindows AS ARRAY

CLASS StdDataWindow INHERIT DataWindow
    PROTECT oOwner AS StandardShellWindow

METHOD Close(oCloseEvent) CLASS StandardShellWindow

    SELF:Owner:Quit()
```

```
EntLine:   1 Line:   1 Col: 1 Length: 213
```

By default, each entity is displayed in its entirety. Clicking on the – button collapses the entity, so that only its declaration statement can be seen, and changes the – button to a + button.

For example, the Close() method entity in the Standard Shell module of Order Entry below is collapsed:


Standard Shell of Order Entry
EntLine: 1 Line: 11 Col: 1 Length: 213
Collapsed entity
+ button

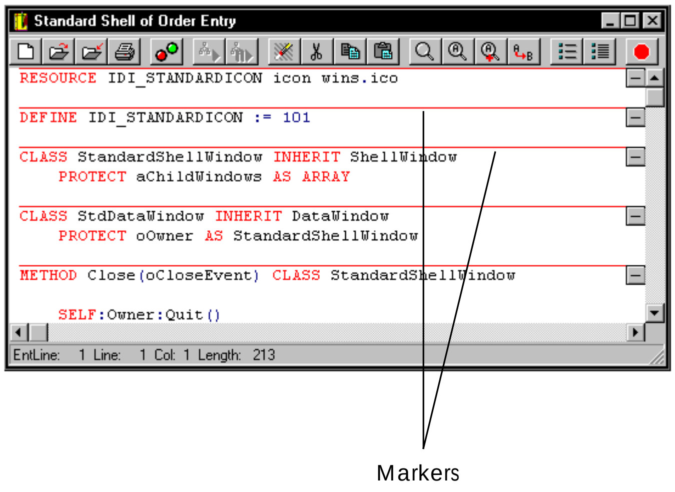
```
RESOURCE IDI_STANDARDICON icon wins.ico

DEFINE IDI_STANDARDICON := 101

CLASS StandardShellWindow INHERIT ShellWindow
    PROTECT aChildWindows AS ARRAY

CLASS StdDataWindow INHERIT DataWindow
    PROTECT oOwner AS StandardShellWindow

METHOD Close(oCloseEvent) CLASS StandardShellWindow
METHOD CloseAll() CLASS StandardShellWindow
```

The + button indicates that the entity is collapsed. Clicking on it expands the entity, showing all of its 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 contents of the Source Code Editor at once.

Markers

If you use the Source Code Editor to edit several entities at once, you will notice another feature—the markers displayed between entities to visually delimit one from the next:


Standard Shell of Order Entry
EntLine: 1 Line: 1 Col: 1 Length: 213
Markers

```
RESOURCE IDI_STANDARDICON icon wins.ico

DEFINE IDI_STANDARDICON := 101

CLASS StandardShellWindow INHERIT ShellWindow
    PROTECT aChildWindows AS ARRAY

CLASS StdDataWindow INHERIT DataWindow
    PROTECT oOwner AS StandardShellWindow

METHOD Close(oCloseEvent) CLASS StandardShellWindow

    SELF:Owner:Quit()
```

You can toggle these markers on or off by alternately checking and unchecking the Entity Markers command on the View menu. The +/- buttons for collapsing and expanding 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, literals, and comments, for example, are all displayed in different colors for immediate visual feedback, based on the colors you selected when setting up your system. See **Setting Editor Options** in "Working in the Desktop" earlier in this guide for detailed information about the System Settings dialog box and the color, auto indentation, keyword casing, case synchronization, and other options for the Source Code Editor.

Accessing the Source Code Editor

Now that you have a general overview of what you can do with the Source Code Editor, you are ready to use it to create and edit source code entities. In this section, you will learn how to:

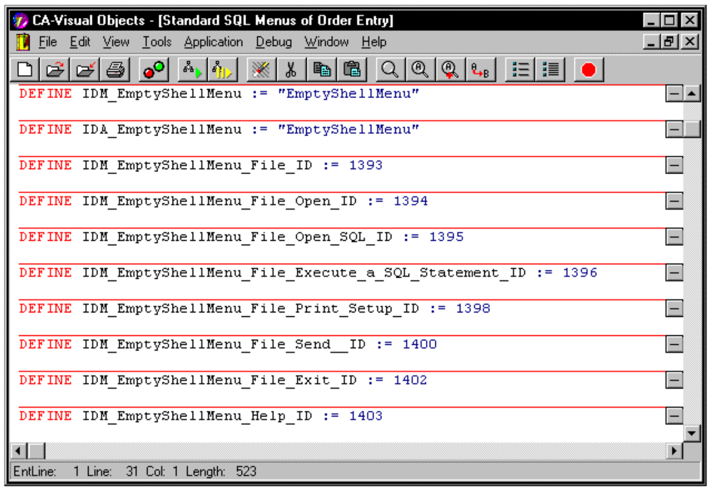
- Create one or more new source code entities
- Load individual entities for editing
- Load all entities defined in a module for editing
- Import a text file for editing

Creating New Entities

To create a new source code entity, you need to perform the following basic steps:

1. In the Repository Explorer's tree view pane, select the desired application and module (for example, Standard SQL Menus in Order Entry).
2. Right-click on the Standard SQL Menus module and select Edit All Source in Module from the pop-up menu.

The Source Code Editor appears:

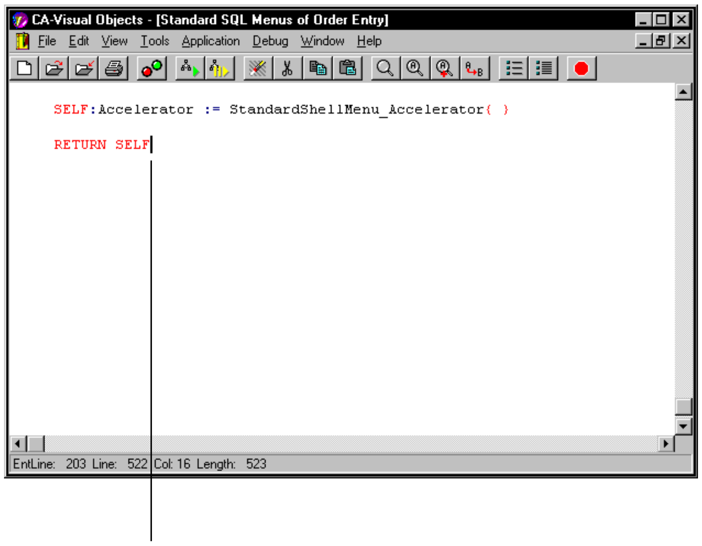

CA-Visual Objects - [Standard SQL Menus of Order Entry]
File Edit View Tools Application Debug Window Help

```
DEFINE IDM_EmptyShellMenu := "EmptyShellMenu"
DEFINE IDA_EmptyShellMenu := "EmptyShellMenu"
DEFINE IDM_EmptyShellMenu_File_ID := 1393
DEFINE IDM_EmptyShellMenu_File_Open_ID := 1394
DEFINE IDM_EmptyShellMenu_File_Open_SQL_ID := 1395
DEFINE IDM_EmptyShellMenu_File_Execute_a_SQL_Statement_ID := 1396
DEFINE IDM_EmptyShellMenu_File_Print_Setup_ID := 1398
DEFINE IDM_EmptyShellMenu_File_Send__ID := 1400
DEFINE IDM_EmptyShellMenu_File_Exit_ID := 1402
DEFINE IDM_EmptyShellMenu_Help_ID := 1403
```

EntLine: 1 Line: 31 Col: 1 Length: 523

Notice that the Source Code Editor is not empty in this instance, as this is one of the Standard Application’s predefined modules.

3. Scroll to the bottom of the window and place the cursor after the last entry:


CA-Visual Objects - [Standard SQL Menus of Order Entry]
File Edit View Tools Application Debug Window Help

```
    SELF:Accelerator := StandardShellMenu_Accelerator( )

    RETURN SELF
```

EntLine: 203 Line: 522 Col: 16 Length: 523

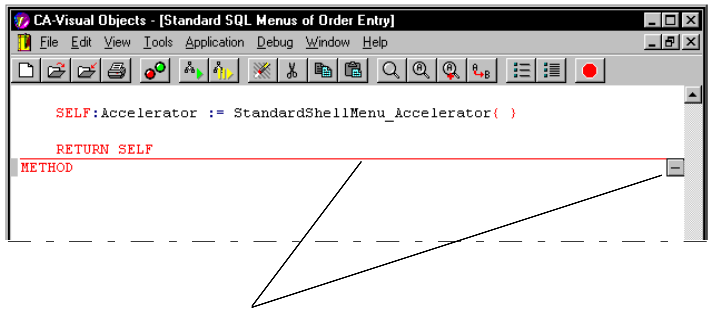
Place cursor here...

4. Press Enter.

   The cursor moves to a new line.

5. Begin typing the source code for the new entity, starting with its declaration statement. For example, enter **Method**.

CA-Visual Objects automatically adds a marker and a
- button to indicate the new entity:


CA-Visual Objects - [Standard SQL Menus of Order Entry]
File Edit View Tools Application Debug Window Help

```
    SELF:Accelerator := StandardShellMenu_Accelerator( )

    RETURN SELF
METHOD
```

Markerand-buttonaddedautomatically

Notice also that CA-Visual Objects recognizes the word “Method” as a keyword as you type! It changes its case (if you selected the Case Synchronization option and the Upper Case option for keywords when setting up your system) and color codes it. (See **Setting Editor Options** in “Working in the Desktop” for more information about case synchronization and color coding of syntactic elements.)

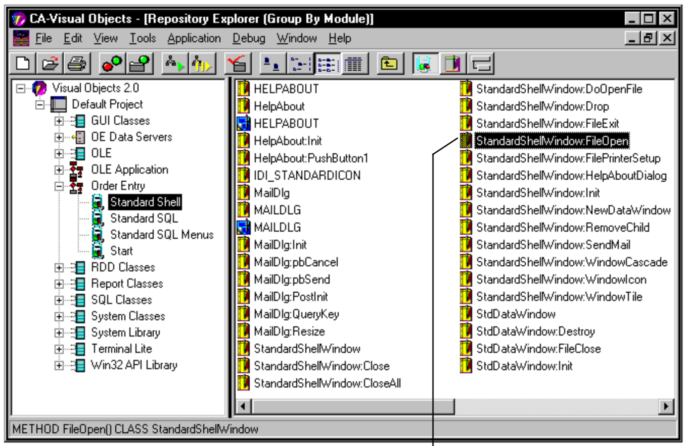
6. At the end of each line, press Enter to start a new line.

7. Click the Save toolbar button to save the new entity.

Loading Single Entities

There are three ways to edit existing entities using the Source Code Editor. The first is to choose individual entities from the Repository Explorer's list view pane. To do this:

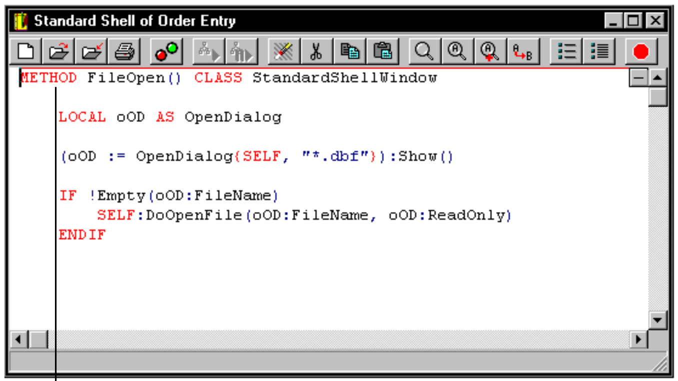
1. Start the Source Code Editor by double-clicking on the entity that you want to edit (for example, FileOpen() in the Standard Shell module of Order Entry):



METHOD FileOpen() CLASS StandardShellWindow

Double-clickhere...

The Source Code Editor appears as follows:



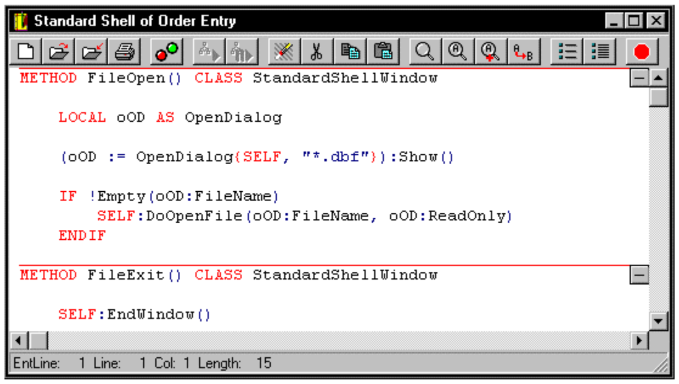
```
METHOD FileOpen() CLASS StandardShellWindow

    LOCAL oOD AS OpenDialog

    (oOD := OpenDialog(SELF, "*.dbf")):Show()

    IF !Empty(oOD:FileName)
        SELF:DoOpenFile(oOD:FileName, oOD:ReadOnly)
    ENDIF
```

Singleentity

Note: Double-clicking on an entity in a module that is associated with an external file loads all source code for that module rather than just the source code for the current entity. See Creating Modules in "Using the Repository Explorer" for more information on working with external modules.

2. To load another entity from the same module, minimize the Source Code Editor and then double-click on another entity in the Repository Explorer (for example, FileExit()).

   Its contents are added to the Source Code Editor when it reappears:


Standard Shell of Order Entry

```
METHOD FileOpen() CLASS StandardShellWindow

    LOCAL oOD AS OpenDialog

    (oOD := OpenDialog(SELF, "*.dbf")):Show()

    IF !Empty(oOD:FileName)
        SELF:DoOpenFile(oOD:FileName, oOD:ReadOnly)
    ENDIF

METHOD FileExit() CLASS StandardShellWindow

    SELF:EndWindow()
```

EntLine: 1 Line: 1 Col: 1 Length: 15

   Note: Double-clicking on a non-source code entity, such as a window or menu, will invoke the appropriate editor for that entity type rather than the Source Code Editor. Double-clicking on an entity in another module will start a new copy of the Source Code Editor for that module.

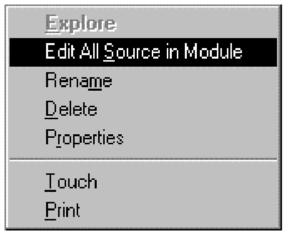
3. Repeat step 2 to load as many entities as you like from the same module.

Loading All Entities

The second method for editing existing entities in the Source Code Editor is to load them all at once using the Edit All Source in Module menu command. For example:

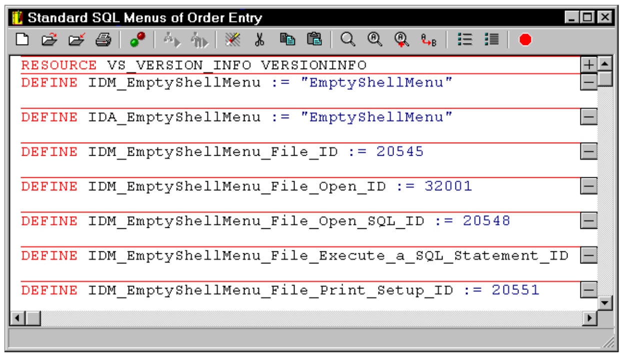
1. Right-click on the Standard SQL Menus module in the Order Entry sample application.

   A local pop-up menu appears:



2. Select the Edit All Source in Module command from the local pop-up menu.

The following code appears:



```
RESOURCE VS_VERSION_INFO VERSIONINFO
DEFINE IDM_EmptyShellMenu := "EmptyShellMenu"

DEFINE IDA_EmptyShellMenu := "EmptyShellMenu"

DEFINE IDM_EmptyShellMenu_File_ID := 20545

DEFINE IDM_EmptyShellMenu_File_Open_ID := 32001

DEFINE IDM_EmptyShellMenu_File_Open_SQL_ID := 20548

DEFINE IDM_EmptyShellMenu_File_Execute_a_SQL_Statement_ID

DEFINE IDM_EmptyShellMenu_File_Print_Setup_ID := 20551
```

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.

Importing a File

Lastly, another alternative is to import source code from a text file while in the Source Code Editor. To do this, use either the File Import toolbar button or the File Import menu command and choose the file you want to import. Its contents will be added to the current contents of the Source Code Editor.

Tip: You can also export the current contents of the Source Code Editor as a text file. For more information on importing and exporting, see **Chapter 12: Importing and Exporting** Applications.

Editing and Saving

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 entering declaration statements and source code for them as described earlier under Creating New Entities.
2. Edit it using standard Windows editing techniques.

   See Editing Source Code later in this chapter for 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 Repository Explorer's List View pane for the appropriate module.

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 all of the standard editing features that you have come to expect. For example, there are toolbar buttons for cut, copy, paste, undo, find, find next, and replace. All of these operations are performed using standard Windows techniques, like drag-and-drop, and are, therefore, not described in this section.

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 the online help system for more information.

The following editing features, however, are unique to the Source Code Editor and are described in this section:

- Deleting individual lines of code
- Inserting a new line
- Going directly to an entity
- Filling in prototypes
- Matching tokens
- Presetting breakpoints

Note: In CA-Visual Objects 2.7, there are a number of very useful aids for editing source code, like automatic insertion of methods, parameter tips, and bookmarks. Additionally, the Source Code Editor's Find dialog box displays its search history, making it easier to repeat a recent search for specified text. For more detailed information, see the the online help.

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 Edit Delete Line menu command).

You can also delete several contiguous lines of code at once by selecting them and pressing the Del key (or choosing the Edit Delete menu command).

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.

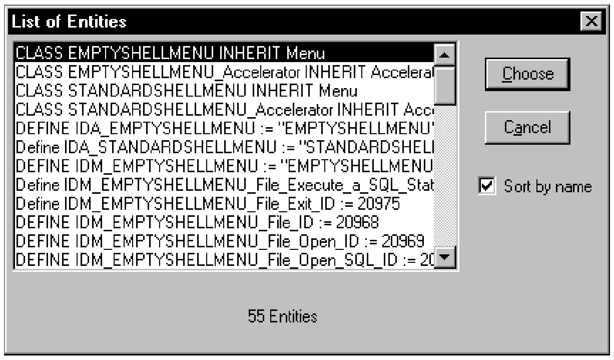
Going Directly to an Entity

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.

   Alternatively, choose the Go To command from the Edit menu.

   The List of Entities dialog box appears, showing the total number of entities currently loaded and their names:

   ```
   CLASS EMPTYSHELLMENU INHERIT Menu
   CLASS EMPTYSHELLMENU_Accelerator INHERIT Accelera
   CLASS STANDARDSHELLMENU INHERIT Menu
   CLASS STANDARDSHELLMENU_Accelerator INHERIT Acc
   DEFINE IDA_EMPTYSHELLMENU := "EMPTYSHELLMENU"
   Define IDA_STANDARDSHELLMENU := "STANDARDSHELL
   DEFINE IDM_EMPTYSHELLMENU := "EMPTYSHELLMENU
   Define IDM_EMPTYSHELLMENU_File_Execute_a_SQL_Stat
   Define IDM_EMPTYSHELLMENU_File_Exit_ID := 20975
   DEFINE IDM_EMPTYSHELLMENU_File_ID := 20968
   DEFINE IDM_EMPTYSHELLMENU_File_Open_ID := 20969
   DEFINE IDM_EMPTYSHELLMENU_File_Open_SQL_ID := 20
   ```

   [✓] Sort by name

   55 Entities

   This example displays all of the entities defined for the Standard SQL Menus module in Order Entry.

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 List View pane of the Repository Explorer.

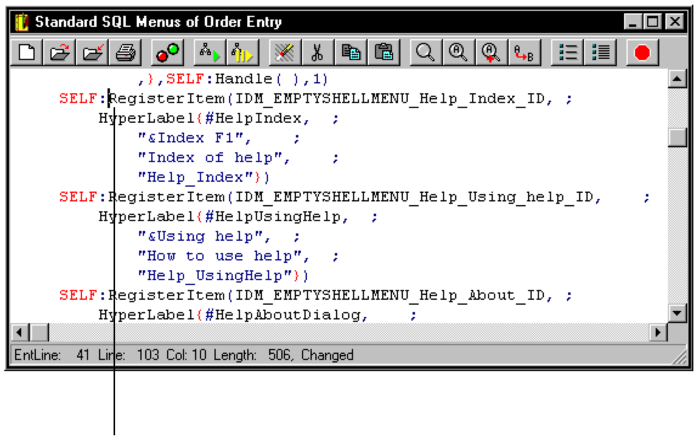
3. Highlight the entity you want to view, and select the Choose button.

   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.

Viewing Prototypes

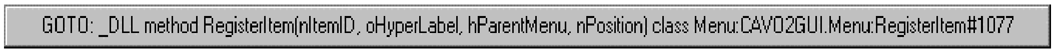
CA-Visual Objects close integration with the repository allows you to view a source code entity's definition. Simply right-click on the specified entity (for example, RegisterItem() in Standard SQL Menus):



```
            ,},SELF:Handle( ),1)
    SELF:RegisterItem(IDM_EMPTYSHELLMENU_Help_Index_ID, ;
        HyperLabel{#HelpIndex,  ;
            "&Index F1",    ;
            "Index of help",    ;
            "Help_Index"})
    SELF:RegisterItem(IDM_EMPTYSHELLMENU_Help_Using_help_ID,    ;
        HyperLabel{#HelpUsingHelp,  ;
            "&Using help",  ;
            "How to use help",  ;
            "Help_UsingHelp"})
    SELF:RegisterItem(IDM_EMPTYSHELLMENU_Help_About_ID, ;
        HyperLabel{#HelpAboutDialog,    ;
```

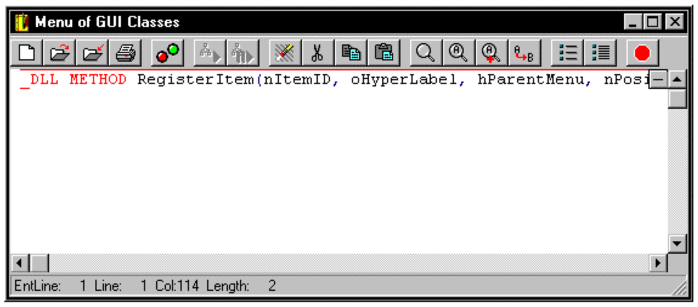
EntLine: 41 Line: 103 Col: 10 Length: 506, Changed

The method's definition displays in a local GOTO pop-up menu:



GOTO: _DLL method RegisterItem(nItemID, oHyperLabel, hParentMenu, nPosition) class Menu:CAVO2GUI.Menu:RegisterItem#1077

If you now click on the pop-up menu (or alternatively, press F11), you will jump to the prototypical code in the relevant class library (and owner application, if appropriate):



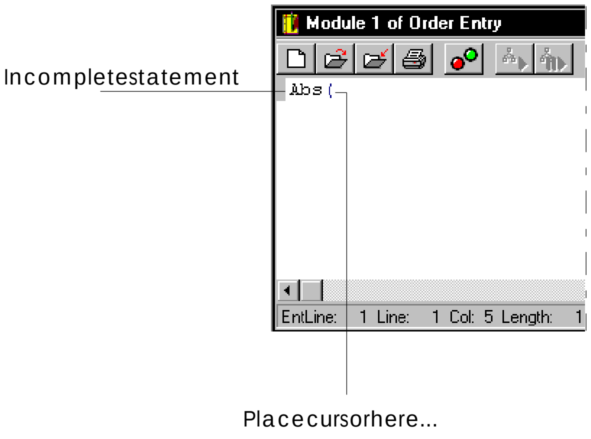
```
 _DLL METHOD RegisterItem(nItemID, oHyperLabel, hParentMenu, nPosi
```

EntLine: 1 Line: 1 Col:114 Length: 2

Filling in Prototypes

The Source Code Editor's close integration with the repository also allows you to easily insert a prototype for incomplete statements, such as functions and methods. To do this:

1. Type the name of the function or method followed by an open parenthesis—for example, **Abs(** on line 1 in the sample code below:



```
Abs(
```

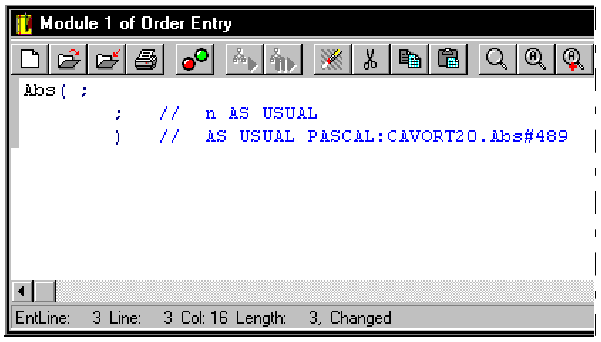
```
EntLine:   1 Line:   1 Col:  5 Length:   1
```

2. Place the cursor just to the right of the open parenthesis.
3. Choose the Insert Prototype command from the Edit menu or, alternatively, click the right mouse button.

   A local pop-up menu appears:

4. Select the Expand Prototype command from the local pop-up menu.

   The Source Code Editor inserts comment lines showing the prototype for all valid parameters, commas to separate the arguments, and a closing parenthesis:



```
Abs( ;
        ;   //  n AS USUAL
        )   //  AS USUAL PASCAL:CAVORT20.Abs#489
```

```
EntLine:   3 Line:   3 Col: 16 Length:   3, Changed
```

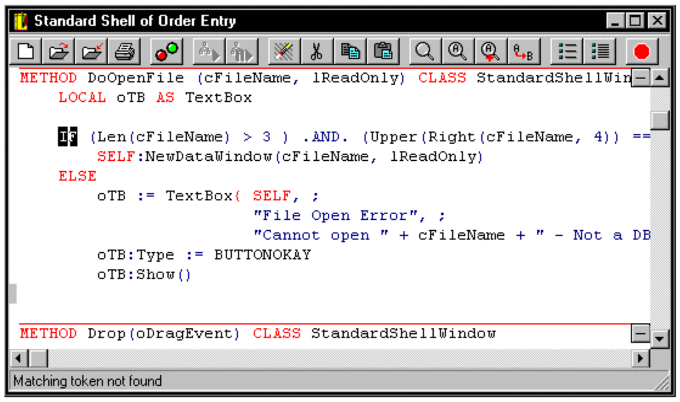
5. Edit the prototype using valid expressions for the function or method.
6. Click the Save toolbar button to save your changes.

Finding Matching Tokens

The Source Code Editor also enables you to search easily for missing tokens in your source code, using the Edit menu's Find Matching Token command. This is a great aid especially when coding nested statements. To use this feature:

1. Place the cursor just before or inside the token (for example, **IF** in the DoOpenFile() method in the Standard Shell module of Order Entry).
2. Choose the Edit Find Matching Token menu command.

   The Source Code Editor indicates that a matching token was not found:



```
METHOD DoOpenFile (cFileName, lReadOnly) CLASS StandardShellWin
    LOCAL oTB AS TextBox

    IF (Len(cFileName) > 3 ) .AND. (Upper(Right(cFileName, 4)) ==
        SELF:NewDataWindow(cFileName, lReadOnly)
    ELSE
        oTB := TextBox( SELF, ;
                        "File Open Error", ;
                        "Cannot open " + cFileName + " - Not a DB
        oTB:Type := BUTTONOKAY
        oTB:Show()

METHOD Drop(oDragEvent) CLASS StandardShellWindow
```

Matching token not found

3. Enter the missing token (for example, **ENDIF**).

   Note: The ENDIF token in the DoOpenFile() method in this instance of the Standard Shell module was deleted only for the purposes of illustration.

4. Click the Save toolbar button to save your changes.

Presetting Breakpoints

CA-Visual Objects allows you to *preset* breakpoints in your source code as you develop or expand an application. Breakpoints allow you to stop running an application at a predetermined line of code where you anticipate a possible error in logic.

For example, perhaps you are expanding the functionality of an existing application that has compiled successfully in the past. As you add new methods to its modules, you can also preset breakpoints within these modules using the Set/Reset Breakpoint toolbar button. When you rebuild the modules or the application later on, you can quickly step through the code at these points as you debug the module or application.

Note: The preset breakpoints remain within the source code until you remove them.

Refer to **Chapter 11: Debugging Your Applications** for detailed information about using breakpoints and correcting errors using the Debugger.

Printing Source Code

Use the Print toolbar button to print all of the source code in the current Source Code Editor window.

Chapter

7

Defining Data Servers and Field Specifications

One of the primary tasks of any GUI database application is to enter, modify, view, and utilize the information stored in databases. This is facilitated by the use of ancillary information, like filters and index files in the Xbase model and WHERE and ORDER BY clauses in the SQL model.

CA-Visual Objects provides a set of editors—the *DB Server Editor* and the *SQL Editor*—that let you create and modify *data servers*. A data server is a high-level, abstract entity designed to give you a consistent object-oriented interface for your database. The DB Server Editor creates data servers based on the traditional Xbase model of a database file, while the SQL Editor creates data servers based on the SQL model of a table or view.

This chapter explains how to create and modify data servers, providing instructions on how to accomplish these tasks in both the DB Server Editor and the SQL Editor. You will also learn how to define field specifications using the FieldSpec Editor.

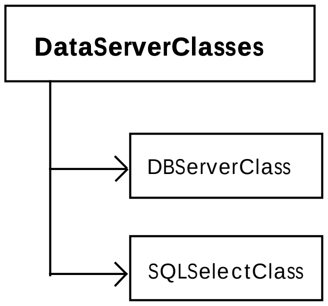
Note: Many of the properties that you define for a data server and its field specifications are designed to be used by data windows that you design and link to the data server. See **Chapter 4: Using the Window** Editor for more information on creating data windows and how to link field specification properties to data controls.

What Is a Data Server?

A data server is a high-level, abstract entity designed to give you a consistent, object-oriented interface for your database. It is like an abstract definition of a database, acting as a database describer, defining its file (or table) name, its fields (or columns), the order in which it is accessed, and so on.

A Single, Compatible Protocol

Using data servers in your CA-Visual Objects applications allows you to access both Xbase and SQL databases using a single, compatible protocol:



Although SQL and Xbase databases use different logic, the SQLSelect class provides an interface compliant with DBServer and other data server classes, allowing an application to operate the same way regardless of the kind of database it uses.

Such compliance is accomplished via a common set of methods (like Append(), Commit(), and Zap()) and a consistent way of referring to fields as properties using ACCESS and ASSIGN methods. These methods make it easy for you to manage different data servers with the same code, regardless of their type. In addition, with these methods, all the capabilities of the traditional Xbase approach are provided, but have been enhanced to fit the event-driven, multitasking nature of GUI applications.

Note: 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-Visual Objects ensures that these changes are reflected in all appropriate places, such as a window that is associated with that data server.

Tip: You may want to refer to the "Data Server Classes" chapter in the *Programmer's Guide* for an overview of the various built-in data server classes. You may also want to refer to the entries for the DBServer, FieldSpec, HyperLabel, and SQLSelect classes in the online help system to learn more about the methods and properties defined for these classes.

What Is a Field Specification?

A field specification is a unique, independent entity in the CA-Visual Objects architecture. It allows you to specify and store a wide variety of properties for an item in your application, all in a single, manageable location. (Note that it is derived from the FieldSpec class.)

Typically, a field specification is "associated" with a field in a data server. Some of the properties you might define for such a field specification are type, length, picture clause, validation checks, etc. In fact, when you create a data server in the DB Server Editor or the SQL Editor, the system automatically generates a field specification for every field in that server.

However, field specifications do not need to be associated with data server fields. For example, when designing a form in the Window Editor, you might add an edit control for which you want to specify a picture clause, a required flag, and a validation check. Even though the control is not linked to a data server field, you can attach a field specification to it to define these properties.

Field specifications, therefore, are useful in a variety of ways. And, because of the nature of repository-based development, field specifications can easily be selected and reused throughout the system.

For example, multiple data servers can access the same property values for common fields: If you create a Salary field specification, you can simply reuse its properties when creating 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 specifications, therefore, saves you both time and resources. Another advantage is that any changes made to a field specification are automatically propagated to all the appropriate places. For example, if you add extra validation to the AcctNum field specification to require that its second digit must be zero or 5, then all data servers that use that field specification would perform that validation.

What Is a Hyperlabel?

A *hyperlabel* (derived from the HyperLabel class) stores information about another object in an application. Using hyperlabels enables your application to display appropriate messages and help topics for the end user about the various objects in the application.

CA-Visual Objects automatically creates a hyperlabel for each data server you create in the DB Server and the SQL Editors. In fact, many of the properties you can set for a data server—like its Name, Caption, Description, and Help Context—are part of its associated hyperlabel.

Similarly, CA-Visual Objects creates a hyperlabel for each data server field or column included, and several of the field properties (Name, Caption, Description, and Help Context) are actually part of its associated hyperlabel. All other field properties are associated with the field specification, which also has a hyperlabel.

The Data Server Editors

With both the DB Server and the SQL Editors, you can import an existing database structure and generate a default set of field specifications that you can optionally modify. This is probably the easiest way to create a data server for an existing database.

The DB Server Editor also lets you generate a database file (and index files) from the data server definition. Thus, you can design a data server "from scratch" and create new files, or you can import existing files and make modifications to them, such as adding a new field to the database file structure, changing a key expression, or creating an additional index file. When you select the File Export menu command, the DB Server Editor will generate the new database and index files.

Warning! *The File Export command overwrites existing files. When working with an existing database file, you should make a backup of the file before using this command.*

When you save the data server, CA-Visual Objects automatically generates object-oriented code within the current module using the appropriate class—DBServer or SQLSelect. (The system also generates object-oriented code for field spec entities based on the FieldSpec class.) The data server is an abstract entity—it is not written to disk as a database file or table, although you can generate files from within the DB Server Editor, as mentioned above.

Using the DB Server Editor

In this section, you will learn how to:

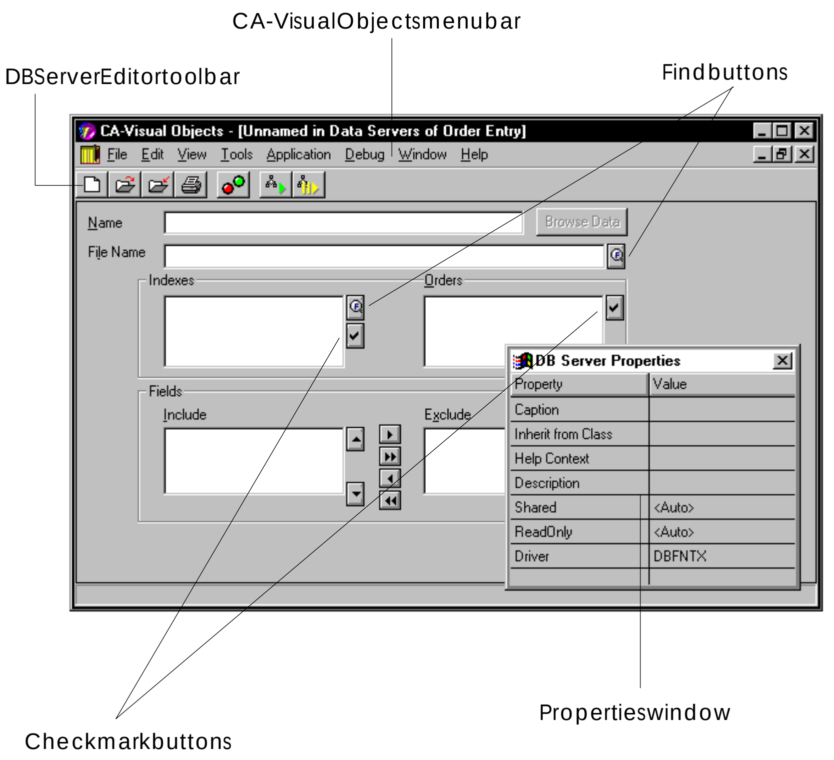
- Define a data server based on the traditional Xbase model of a .DBF file
- Add new fields to a data server and specify properties for them
- Add index files to a data server and specify properties for them
- Modify a data server by editing its properties, fields, and index files
- Import the structure of an existing database file and associated index files
- Create a database file and index files by exporting a data server definition

Important! *The following libraries must be included in the search path defined for your application in order to use a data server created using the DB Server Editor: RDD Classes and System Classes. (For more information on setting a search path, see* ***Setting the Search Path*** *in "Using the Repository Explorer.")*

The DB Server Editor

Similar to the Window or Menu Editor, the DB Server Editor has its own toolbar and an associated Properties window, in addition to the menu commands available on the CA-Visual Objects menu bar.

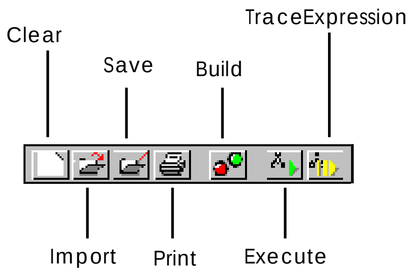
When first loaded for a new data server entity, the DB Server Editor looks as follows:



Property	Value
Caption	
Inherit from Class	
Help Context	
Description	
Shared	<Auto>
ReadOnly	<Auto>
Driver	DBFNTX

The Toolbar

The DB Server Editor toolbar contains the following buttons:

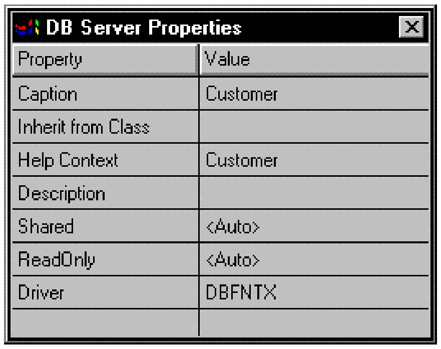


All of the buttons except Build, Execute, and Trace Expression are discussed in this chapter. See Chapter 2: Working in the Desktop for information about Build and Execute and "Debugging Your Applications" for details on Trace Expression.

Tip: For a quick description of any toolbar button, simply point to it—a tooltip window with descriptive text pops up right next to the button.

The Properties Window

When the DB Server Editor is launched, a modeless, *floating*, Properties window, shown below, is automatically opened. Initially, this window allows you to specify properties for the current data server.



Property	Value
Caption	Customer
Inherit from Class	
Help Context	Customer
Description	
Shared	<Auto>
ReadOnly	<Auto>
Driver	DBFNTX

Note: Refer to the online help for detailed information about new and updated data server properties.

When a new data server is opened, the *Property* column lists all properties that can be specified for the new data server; the value cell to the right of each property is used to enter a value for the property. For example, you can specify the replaceable database driver (RDD) to associate with the current data server. (See Specifying Data Server Properties for complete details.)

As you develop the data server, this window takes on different roles depending on the current focus of the editor. When creating or modifying the *fields* of the data server, it changes to the FieldSpec Properties window; when creating or modifying *indexes*, it changes to the Index Properties window.

See Specifying Data Server Properties, Specifying Field Properties, and Specifying Index File and Order Properties later in this chapter for more information about this window.

Note: The Properties window always remains open until explicitly closed (using the system menu or the Show Properties Window menu command) or until its owner, the DB Server Editor window, is closed. If explicitly closed, reopen it at any time using the Show Properties Window command on 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.

Other Options

The remainder of the options in the DB Server Editor workspace allow you to specify various properties for the current data server, such as a descriptive data server name and a database file name. See the Defining a Data Server in the DB Server Editor section next for details.

Defining a Data Server in the DB Server Editor

Defining a data server in the DB Server Editor requires you to:

- Create the data server and specify its properties
- Add fields to it and specify their properties
- Optionally set indexes and specify their properties

Importing a Data Server

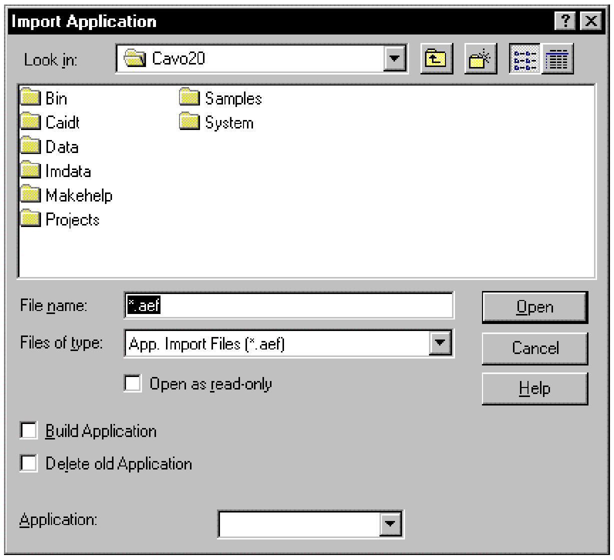
In the *Getting Started* guide, you were shown how to access the predefined data server for the CUSTOMER.DBF file by importing the OE Data Servers library. The Customer data server was created for you using the DB Server Editor, and it resides in a separate library on disk as an external file.

Note: The applications and libraries you create in CA-Visual Objects can be stored external to the repository as Application Export Format (.AEF) files using the File Export menu command. When desired, you can import them back into the repository using the File Import menu command. See **Chapter 12: Importing and Exporting** Applications for more detailed information.

If you have not already imported this library and loaded the Customer data server in the DB Server Editor, the steps are repeated here:

1. Click on Default Project in the Repository Explorer.
2. Choose the File Import menu command.

   The Import Application dialog box appears:

   File name: *.aef

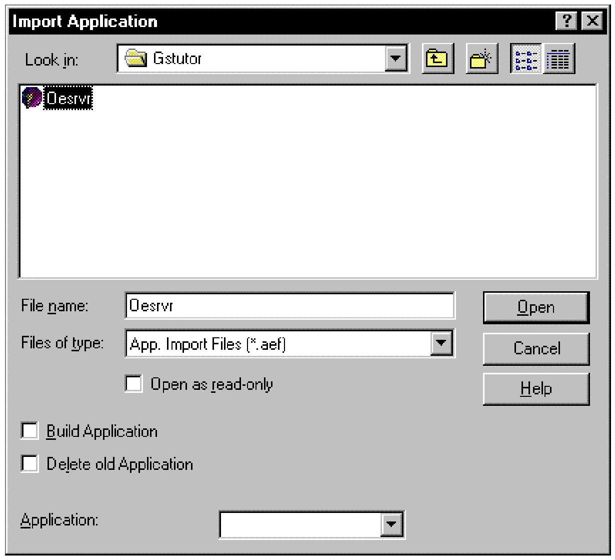
   Files of type: App. Import Files (*.aef)

3. Double-click on the SAMPLES folder in the CAVO27 directory, and then double-click on the GSTUTOR folder that appears.

Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory.

4. Select Oesrvr in the GSTUTOR folder.

   Oesrvr is added to the File Name edit control:



Look in: Gstutor

Oesrvr

File name: Oesrvr

Files of type: App. Import Files (*.aef)

☐ Open as read-only

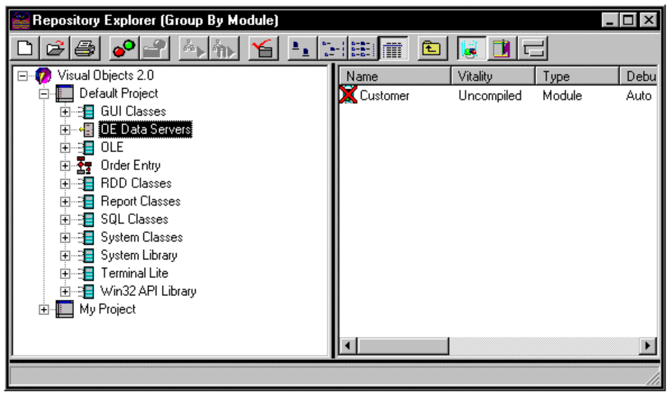
☐ Build Application

☐ Delete old Application

Application:

5. Click Open.

   The OE Data Servers library is added to the Repository Explorer's tree structure:



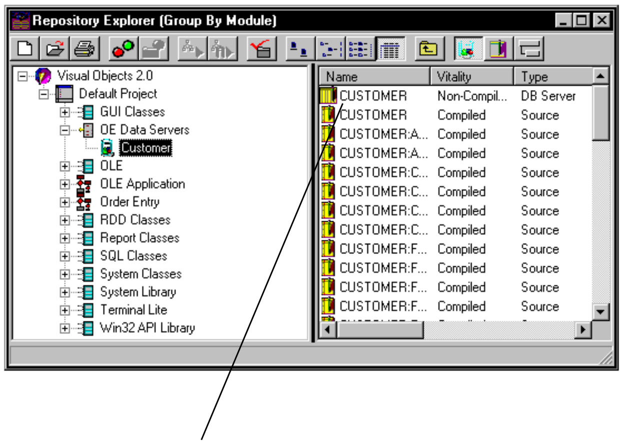
- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
    - OLE
    - Order Entry
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library
    - Terminal Lite
    - Win32 API Library
  - My Project

Name	Vitality	Type	Debu
Customer	Uncompiled	Module	Auto

6. Compile the OE Data Servers library by clicking on the Build toolbar button.

7. Now expand OE Data Servers to display its Customer module, and then double-click on the Customer module to display its entities in the Repository Explorer's list view pane.

8. Double-click on the Customer DB server entity:

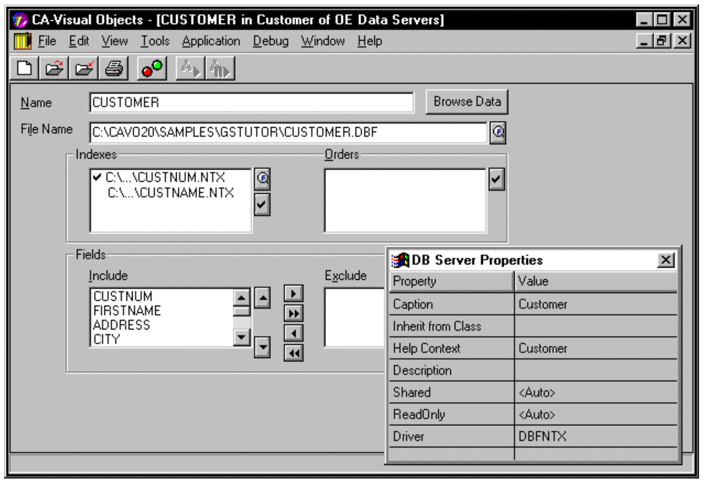


- Visual Objects 2.0
  - Default Project
    - GUI Classes
    - OE Data Servers
      - Customer
    - OLE
    - OLE Application
    - Order Entry
    - RDD Classes
    - Report Classes
    - SQL Classes
    - System Classes
    - System Library
    - Terminal Lite
    - Win32 API Library

Name	Vitality	Type
CUSTOMER	Non-Compil...	DB Server
CUSTOMER	Compiled	Source
CUSTOMER:A...	Compiled	Source
CUSTOMER:A...	Compiled	Source
CUSTOMER:C...	Compiled	Source
CUSTOMER:C...	Compiled	Source
CUSTOMER:C...	Compiled	Source
CUSTOMER:C...	Compiled	Source
CUSTOMER:F...	Compiled	Source
CUSTOMER:F...	Compiled	Source
CUSTOMER:F...	Compiled	Source
CUSTOMER:F...	Compiled	Source

Double-clickhere

The Customer data server is loaded in the DB Server Editor:


CA-Visual Objects - [CUSTOMER in Customer of OE Data Servers]
File Edit View Tools Application Debug Window Help
Browse Data

Name: CUSTOMER

File Name: C:\CAVO20\SAMPLES\GSTUTOR\CUSTOMER.DBF

Indexes:
✓ C:\...\CUSTNUM.NTX
C:\...\CUSTNAME.NTX

Orders:

Fields

Include:
CUSTNUM
FIRSTNAME
ADDRESS
CITY

Exclude:

DB Server Properties

Property	Value
Caption	Customer
Inherit from Class	
Help Context	Customer
Description	
Shared	<Auto>
ReadOnly	<Auto>
Driver	DBFNTX

Notice that this predefined data server has two indexes, CUSTNUM.NTX and CUSTNAME.NTX, defined to it and that the check mark to the left of CUSTNUM.NTX indicates that this index file contains the controlling order. Also notice that all of the fields in the Customer data server are listed in the Include list box in the Fields group box. (All of these components and the Properties window are discussed in detail in the next section, Creating a New Data Server.)

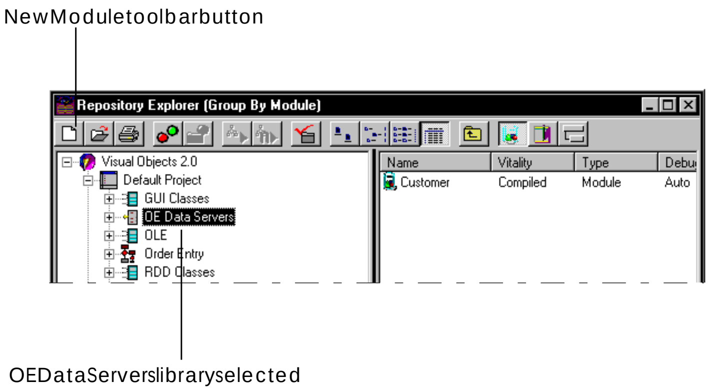
9. Click the Close button to exit the DB Server Editor.

Creating a New Data Server

To create a new data server—in this instance, the Orders data server for the Order Entry sample application—you need to perform the following steps:

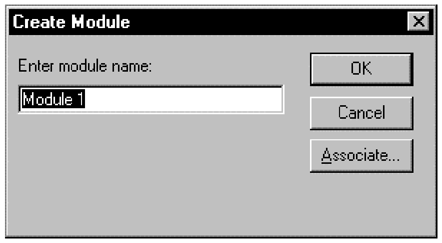
Accessing the OE Data Servers Library

1. Select the OE Data Servers library in the Repository Explorer's tree view pane, and then click on the New Module toolbar button:



Name	Vitality	Type	Debu
Customer	Compiled	Module	Auto

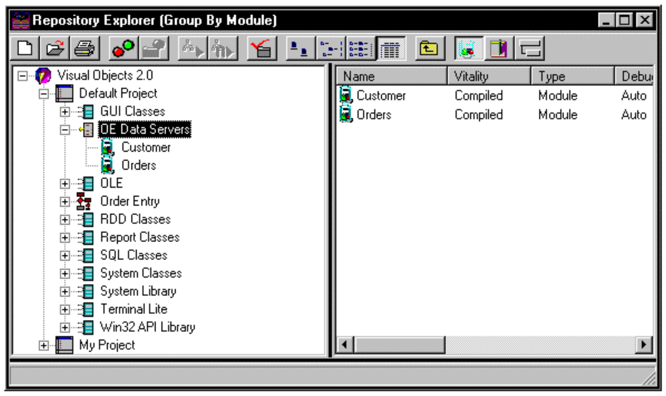
The Create Module dialog box appears:



2. Type **Orders** in the Enter Module Name edit control.

3. Click OK.

The new module is added to the Repository Explorer's tree structure:



Name	Vitality	Type	Debu
Customer	Compiled	Module	Auto
Orders	Compiled	Module	Auto

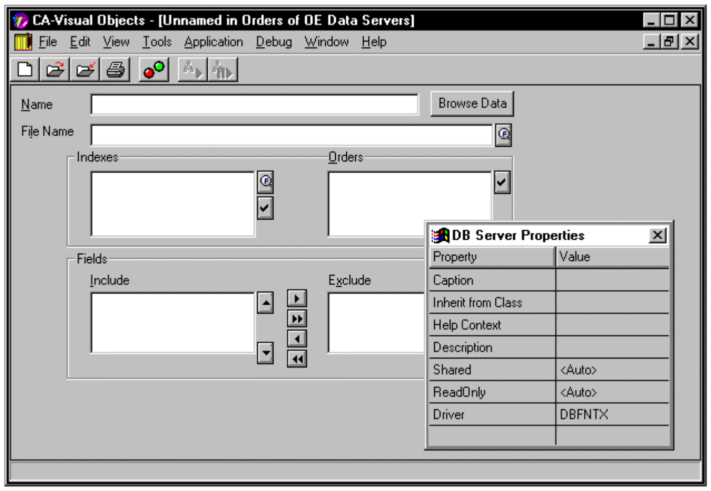
Note: To create an Xbase-style data server without accessing a data servers library, you would skip steps 1 through 3 and begin by starting the DB Server Editor (next step).

Starting the DB Server Editor

4. From within a specified module (in this instance, the Orders module of OE Data Servers), select the DB Server Editor command from the Tools menu.

   Alternatively, click the New Entity toolbar button and then select the DB Server Editor command from the local pop-up menu.

   The DB Server Editor appears:


CA-Visual Objects - [Unnamed in Orders of OE Data Servers]
File Edit View Tools Application Debug Window Help
Name
Browse Data
File Name
Indexes
Orders
Fields
Include
Exclude
DB Server Properties

Property	Value
Caption	
Inherit from Class	
Help Context	
Description	
Shared	<Auto>
ReadOnly	<Auto>
Driver	DBFNTX

5. In the Name edit control, type the name of the data server. This information is included in the data server's automatically generated hyperlabel.

   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.

   Note: Blanks within the name will be changed to underscores when you save the data server.

   This name is used to create the DB server entity and to subclass the data server in the code generated later by the DB Server Editor. For example:

```
CLASS Customer_Order INHERIT DBServer
```

   It is also the name used to refer to this data server in your program code. For example:

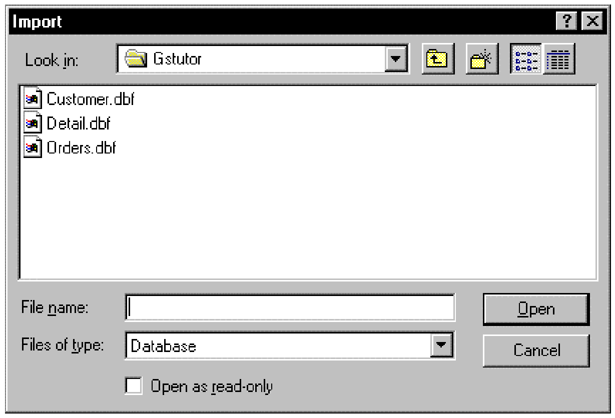
```
oDBCust := Customer_Order{}
```

Importing a Database File

6. Alternatively, you can import an existing database file, and then add new fields to the data server or make modifications to the existing fields.

   To do so, click on the Find button to the right of the File Name edit control (or click on the Import toolbar button).

A standard Import dialog box appears:



Look in: Gstutor

Customer.dbf
Detail.dbf
Orders.dbf

File name:

Files of type: Database

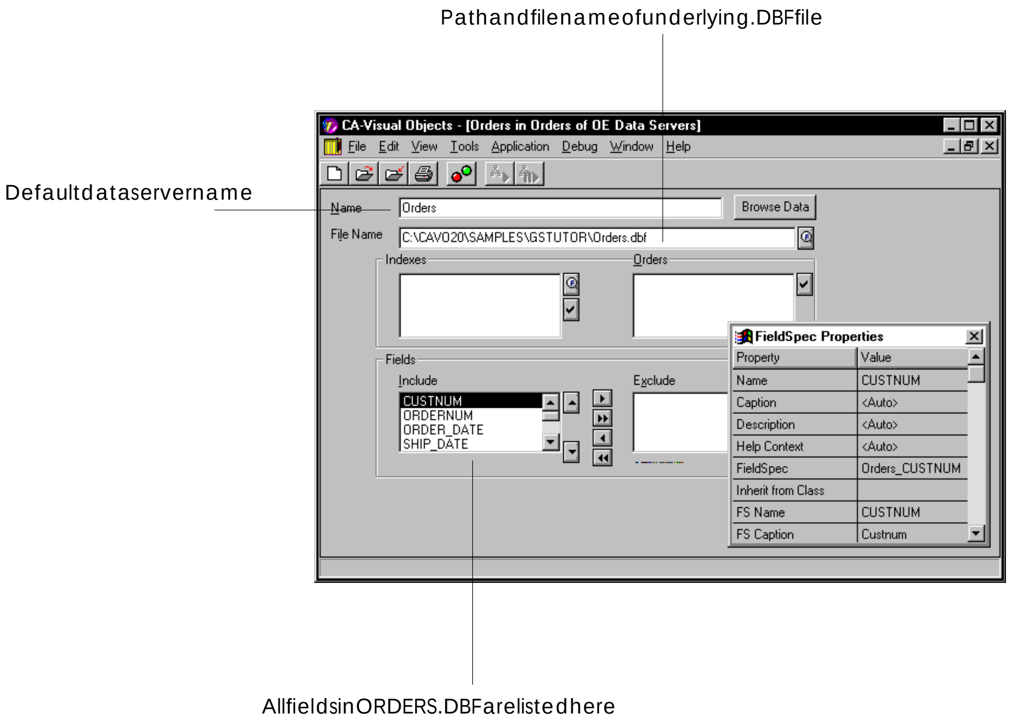
☐ Open as read-only

7. Select ORDERS.DBF in the GSTUTOR folder.

   Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory.

8. Click Open.

   The ORDERS.DBF file is loaded in the DB Server Editor:



CA-Visual Objects - [Orders in Orders of OE Data Servers]

Name: Orders

File Name: C:\CAVO20\SAMPLES\GSTUTOR\Orders.dbf

Include:
CUSTNUM
ORDERNUM
ORDER_DATE
SHIP_DATE

FieldSpec Properties

Property	Value
Name	CUSTNUM
Caption	<Auto>
Description	<Auto>
Help Context	<Auto>
FieldSpec	Orders_CUSTNUM
Inherit from Class	
FS Name	CUSTNUM
FS Caption	Custnum

For the Orders data server, accept the default name.

9. In the File Name edit control, type the name of an existing database file, including an optional drive, directory, and extension, with which this data server is to be associated.

Alternatively, use the default path and file name of an imported .DBF file.

Note: To create a data server and generate a *new* database file, enter a new file name. You can generate the corresponding .DBF file (and optionally index files) later, using the File Export menu command.

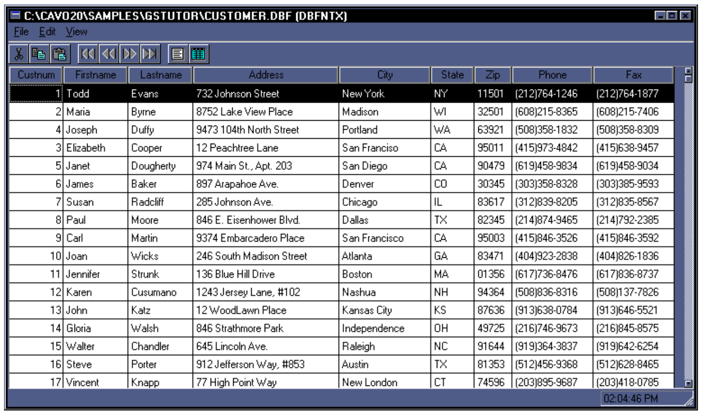
For the Orders data server, accept the default path and file name, C:\CAVO27\SAMPLES\GSTUTOR\ORDERS.DBF.

10. Optionally set additional data server properties using the DB Server Properties window.

    See the next section, Specifying Data Server Properties, for details.

11. Define one or more fields for the data server.

    You can click the Browse Data button to browse an existing database as an aid in determining which fields you want to include and/or exclude from your data server. For example:



Custnum	Firstname	Lastname	Address	City	State	Zip	Phone	Fax
1	Todd	Evans	732 Johnson Street	New York	NY	11501	(212)764-1246	(212)764-1877
2	Maria	Byrne	8752 Lake View Place	Madison	WI	32501	(608)215-8365	(608)215-7406
4	Joseph	Duffy	9473 104th North Street	Portland	WA	63921	(508)358-1832	(508)358-8309
3	Elizabeth	Cooper	12 Peachtree Lane	San Franciso	CA	95011	(415)973-4842	(415)638-9457
5	Janet	Dougherty	974 Main St., Apt. 203	San Diego	CA	90479	(619)458-9834	(619)458-9034
6	James	Baker	897 Arapahoe Ave.	Denver	CO	30345	(303)358-8328	(303)385-9593
7	Susan	Radcliff	285 Johnson Ave.	Chicago	IL	83617	(312)839-8205	(312)835-8567
8	Paul	Moore	846 E. Eisenhower Blvd.	Dallas	TX	82345	(214)874-9465	(214)792-2385
9	Carl	Martin	9374 Embarcadero Place	San Francisco	CA	95003	(415)846-3526	(415)846-3592
10	Joan	Wicks	246 South Madison Street	Atlanta	GA	83471	(404)923-2838	(404)826-1836
11	Jennifer	Strunk	136 Blue Hill Drive	Boston	MA	01356	(617)736-8476	(617)836-8737
12	Karen	Cusumano	1243 Jersey Lane, #102	Nashua	NH	94364	(508)836-8316	(508)137-7826
13	John	Katz	12 WoodLawn Place	Kansas City	KS	87636	(913)638-0784	(913)646-5521
14	Gloria	Walsh	846 Strathmore Park	Independence	OH	49725	(216)746-9673	(216)845-8575
15	Walter	Chandler	645 Lincoln Ave.	Raleigh	NC	91644	(919)364-3837	(919)642-6254
16	Steve	Porter	912 Jefferson Way, #853	Austin	TX	81353	(512)456-9368	(512)628-8465
17	Vincent	Knapp	77 High Point Way	New London	CT	74596	(203)895-9687	(203)418-0785

    See Adding Fields later in this chapter for more detailed information.

12. Optionally define one or more index files for the data server.

    See Adding Index Files later in this chapter for more detailed information.

13. Choose the Save toolbar button to save the data server.

    See Generating Code later in this chapter for more detailed information.

    Note: This step can be repeated whenever you make changes to the data server and want to save your work without closing the DB Server Editor.

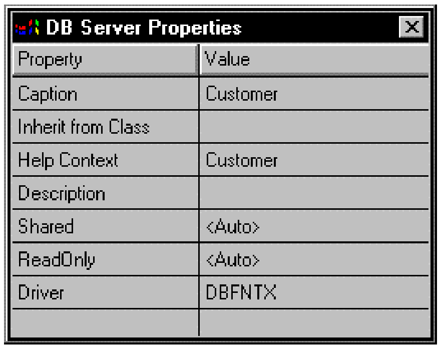
14. If you are using a data server library, you must include the library in the application's search path. For example, add the OE Data Servers library now to Order Entry's search path, so that the sample application has access to both the Customer and Orders data servers.

    See **Setting the Search Path** in "Using the Repository Explorer" for modifying an application's path and other properties.

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.

Specifying Data Server Properties

You can define additional properties for a data server using the DB Server Properties window, a two-column list box with properties listed on the left and the corresponding current value for each property on the right:



Property	Value
Caption	Customer
Inherit from Class	
Help Context	Customer
Description	
Shared	<Auto>
ReadOnly	<Auto>
Driver	DBFNTX

Note: In this version of CA-Visual Objects, there are several new data server properties, including PreInit Action, PostInit Action, and No Access / Assign. Refer to the online help for detailed information about new and updated data server properties.

Caption
Optionally enter the caption text to be associated with this data server. This text is included in the data server's hyperlabel and may be used subsequently in a window's title bar, for example.

Inherit from Class
Select the DBServer class or subclass from which the data server will inherit its basic characteristics. Valid choices are <Auto> and all classes derived from the DBServer class that are in the current search path. The default setting is <Auto>.

Help Context
Optionally enter a unique keyword for the data server that can be used to identify it in a context-sensitive help system. This property can consist of letters, digits, and the underscore character (_), but it cannot contain any other characters, including blanks or punctuation. This information is included in the data server's automatically generated hyperlabel. (For more information on creating a help file, refer to the "GUI Classes" chapter in the *Programmer's Guide.*)

Description
Optionally enter the descriptive text to be associated with this data server. This text is included in the data server's hyperlabel, and may be used subsequently in a window's status bar.

Shared	Specify whether the data server will open the associated database in shared (Yes) or exclusive (No) mode. The default is <Auto>, which enables you to waive an explicit setting and let the open mode be determined by the application’s SetExclusive() flag. See the online help system for more information about the SetExclusive() function.
Read Only	Choose whether the data server will open the associated database file as a read-only (Yes) or read-write (No). The default is <Auto>, which means the system default setting, read-write, is used as the file open mode.
Driver	Choose the name of the database driver to be used with this data server. The drop-down list displays the RDDs currently installed on your system; the default is the DBFNTX driver.

Adding Fields

Techniques

Once a data server’s properties are specified, you can continue defining the data server by adding fields to it. When creating fields, there are a number of techniques you can use, such as:

- Creating a new field, specifying all required properties and, optionally, additional properties

  The DB Server Editor generates a default field specification for the new field.
- Creating a new field and associating it with an existing field specification

  Instead of specifying properties for the new field, you can use those of the associated field specification.
- Creating a new field, associating it with an existing field specification, and then customizing the properties automatically imported with the field specification

These three techniques are described in greater detail in the following sections.

Note that no matter which technique you use, there are several *required* properties that must be specified. The steps described below focus on the requirements; additional properties are summarized in the next section, Specifying Field Properties.

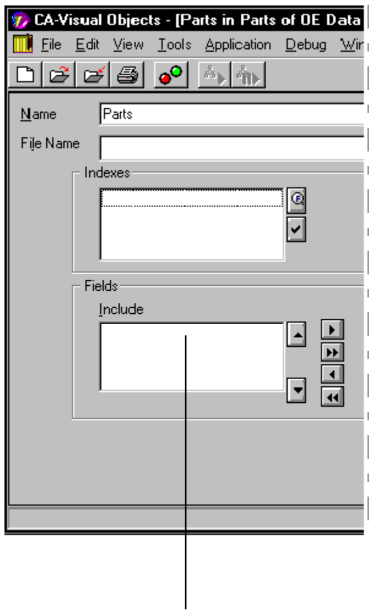
Creating a New Field

To add a new field to a data server and fully specify all required properties:

1. In the Include list box, click on any entry.

   If there are no fields included in the data server, the DB Server Editor indicates the selected, empty entry with a dotted gray line.

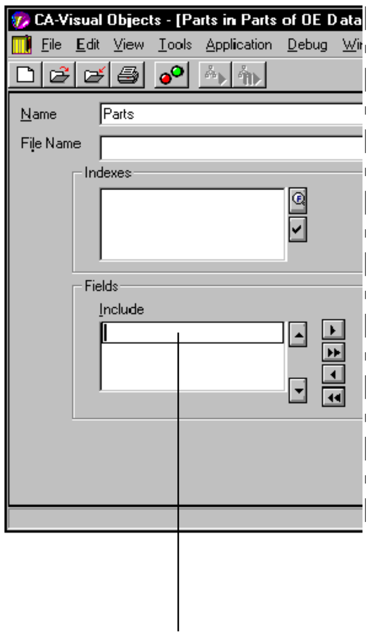
   For example, if you were creating a new data server called Parts for the OE Data Servers library, the Include list box would be empty initially. To begin to define the first field, you would click in the Include list box:



Clickhereforanewfieldentry

2. Choose the Add Item command from the Edit menu.

   The DB Server Editor opens an edit control for the new entry:

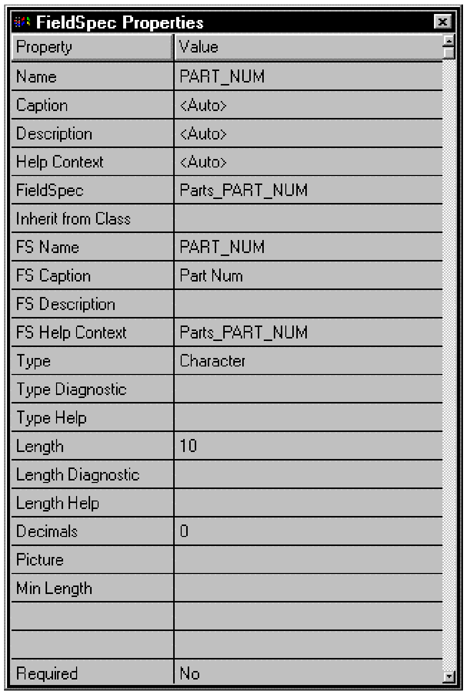

CA-Visual Objects - [Parts in Parts of OE Data
File Edit View Tools Application Debug Wi
Name
Parts
File Name
Indexes
Fields
Include

Openeditcontrol

3. Type the name of the field (for example, **PART_NUM**).

4. Press Enter.

   Note that as soon as you press Enter, the DB Server Editor automatically fills in the FieldSpec Properties window with default values for some of the properties:

Property	Value
Name	PART_NUM
Caption	<Auto>
Description	<Auto>
Help Context	<Auto>
FieldSpec	Parts_PART_NUM
Inherit from Class	
FS Name	PART_NUM
FS Caption	Part Num
FS Description	
FS Help Context	Parts_PART_NUM
Type	Character
Type Diagnostic	
Type Help	
Length	10
Length Diagnostic	
Length Help	
Decimals	0
Picture	
Min Length	
Required	No

   Note: Refer to the online help for detailed information about new and updated field properties.

   These default values can be changed at any time. Most of these properties are optional and are explained in greater detail in the Specifying Field Properties section below. However, there are two required properties—Type and Length—that you may want to change.

   Note: This field-specific Properties window is almost identical to the FieldSpec Editor you will learn about later in this chapter.

5. Scroll down to the Type property and double-click on its value cell.

   Select a field type from the combo box that appears. Valid choices are: Character, Numeric, Date, Logic, or Memo.

6. Use the Length property to specify the length of the new field.

Using a Field Specification

To create a new field that uses *all* of the properties of an existing field specification:

1. Add a new field according to steps 1-3 above.
2. Double-click on the FieldSpec property's value cell, and choose the desired field specification from the combo box that appears.

   The DB Server Editor fills in the FieldSpec Properties window with the values assigned to the selected field specification.

Customizing a Field Specification

To create a new field that uses only some of the properties of an existing field specification, follow the steps outlined in the previous section, Using a Field Specification. Then, simply customize any of the properties pulled in with the selected field specification.

Important! *If you are customizing an existing field specification, you may want to use a different name for the modified version and leave the existing field specification intact. To do this, type a new FieldSpec name in the FieldSpec Properties window before saving the customized field specification.*

No matter which technique you used to create the field, at this point all requirements have been met and the new field is complete. You can specify additional properties for this field by referring to the Specifying Field Properties section below.

You can also define additional fields by repeating the steps in one of the three preceding sections. Each subsequent field is added to the end of the list and will be included in the database file structure if you issue a File Export menu command. See the Editing Fields section for details on repositioning the order of the fields in the Include list box.

Specifying Field Properties

The FieldSpec Properties window allows you to specify the following properties for fields. Unless indicated otherwise, these properties are optional.

Note: In CA-Visual Objects 2.7, there are many new field properties, including Length Diagnostic, Length Help, Min Length Diagnostic, Min Length Help, Range Diagnostic, and Range Help. Refer to the online help for detailed information about new and updated field properties.

Name

The name of the currently selected field. (This is a required property.)

If you newly created this field, this property contains the name you entered when creating the field. If this field was imported, this property contains the name of the field as it was defined in the DBF file structure (see Importing Database and Index Files later in this chapter).

	Field names must be 1 to 10 characters in length, the first character cannot be a number, and the rest can only contain letters, numeric digits, and the underscore character. If you later use the File Export menu command to create a .DBF file from this data server definition, this name will be used to create this field.
Caption	The text used to label the field. (For example, if you use Auto Layout in the Window Editor for this server, the system will use the caption to create the fixed text control for this field.) This property is initially set to <Auto>, which means that, by default, the system will use the value in the FS Caption property for this field's caption. You should enter a value in the Caption property only if you wish to override the value defined in FS Caption without changing it. If you do not care about modifying the value held in FS Caption, simply enter the desired text there. **Note:** Refer to **Specifying Control Properties and Style** Settings in the "Using the Window Editor" chapter for an example and accompanying diagram of the hierarchical nature of the Caption, Description, and Help Context properties.
Description	The text displayed in a window's status bar when this field has focus in your application (for example, in a data window). (See the Data-Aware Windows section in "Using the Window Editor" for information on creating data windows.) This property is initially set to <Auto>, which means that, by default, the system will use the value in the FS Description property for this field's description. You should enter a value in the Description property only if you wish to override the value defined in FS Description without changing it. If you do not care about modifying the value held in FS Description, simply enter the desired text there.
Help Context	The unique keyword to be associated with this field for use in a context-sensitive help system. See the Help Context entry in Specifying Data Server Properties earlier in this chapter for specifics. This property is initially set to <Auto>, which means that, by default, the system will use the value in the FS Help Context property for this field's help keyword. You should enter a value in the Help Context property only if you wish to override the value defined in FS Help Context without changing it. If you do not care about modifying the value held in FS Help Context, simply enter the desired keyword there.
FieldSpec	The name of the field spec entity associated with this field, if any. If you designate a field specification, then all of the properties listed below will be saved to that entity. All entities must have unique names to be stored in the repository. Therefore, the entity name generally includes the name of the server as well as the field.

By default, the system creates a unique field specification for every field you create or import (using the pattern <*DataServerName*>_<*FieldName*>). When you save a data server, this name is used to create the field spec entity and to subclass the field specification in the code generated by the DB Server Editor. For example:

```
CLASS ZipCodeSpec INHERIT FieldSpec
```

If desired, you can use this property to associate a different, existing field specification, instead of the default, unique one automatically associated by the system. To do this, click on this property, click on the Down arrow button that appears, and then choose the desired field specification from the displayed list.

For example, if you created an LNAME field in your Customer data server, it would be associated with a field specification of CUSTOMERS_LNAME by default. Assuming you had a LASTNAME field specification you preferred to use for this field, you could import it here by choosing it from the drop-down list box.

Inherit from Class

Select the FieldSpec class or subclass from which the field will inherit its basic characteristics. Valid choices are <Auto> and all classes derived from the FieldSpec class that are in the current search path. The default setting is <Auto>.

FS Name

The name of the field described in the field specification. (All entities must have unique names to be stored in the repository.)

Initially, for newly created or imported fields, this property contains the same value as the Name property.

FS Caption

The text used to label the field. (For example, if you use Auto Layout in the Window Editor for this server, the system will use the caption to create the fixed text control for this field.)

By default, the value in this property will be used for this field’s caption, unless you enter a value in the Caption property (in which case the value in that property will be used instead).

Initially, for newly created or imported fields, this property contains a mixed-case version of the value in the Name property.

Note: Refer to **Specifying Control Properties and Style** Settings in the “Using the Window Editor” chapter for an example and accompanying diagram of the defaulting behavior of the FS Caption, FS Description, and FS Help Context properties.

FS Description

The text displayed in a window’s status bar when this field has focus in your application (for example, in a data window).

By default, the value in this property will be used for this field’s description, unless you enter a value in the Description property (in which case the value in that property will be used instead).

FS Help Context	The unique keyword to be associated with this field in the specified field specification. This keyword is intended for use in a context-sensitive help system. See the Help Context entry in Specifying Data Server Properties earlier in this chapter for specifics. By default, the value in this property will be used for this field's help keyword, unless you enter a value in the Help Context property (in which case the value in that property will be used instead). For newly created or imported fields, the default keyword for this property is created using the pattern <*DataServerName*>_<*FieldName*>.
Type	The data type of the field. You can choose Character, Numeric, Logical, Date, OLE, or Memo. (This is a required property). If you newly created this field, the type is initially set to Character. If this field was imported, this property contains the type of the field as defined in the .DBF file structure.
Type Diagnostic	The message that should appear in the status bar when an error occurs related to this field's type.
Type Help	The unique keyword for use in a context-sensitive help system that identifies the help message for the Type property. See the Help Context entry in Specifying Data Server Properties earlier in this chapter for specifics.
Length	The length of the field. (This is a required property). If you newly created this field, the length is initially set to 10. If this field was imported, this property contains the length of the field as defined in the DBF file structure. **Note:** For new fields, the different types you might choose have different default lengths associated with them. They are as follows: Character = 10, Numeric = 12 with 2 decimals, Date = 8, Logic = 1, OLE=10, and Memo = 10.
Length Diagnostic	The message that should appear when an error occurs related to this field's length.
Length Help	The unique keyword for use in a context-sensitive help system that identifies the help message for the Length property. See the Help Context entry in Specifying Data Server Properties earlier in this chapter for specifics.

Decimals	The number of decimal places to be used for this field if it is numeric. If you newly created this field and you specified a type of Numeric, this property is initially set to 2. If this field was imported as a Numeric, this property contains the decimal of the field as defined in the DBF file structure. **Note:** For all other types, this property is unused (set to 0).
Picture	The picture clause to be used to format this field. This is a standard Xbase picture clause, such as "@!" or "999-99-9999". Note that the quotation marks are optional.
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. By default, this property is set to No.
Required Diagnostic	The message that should appear when an error occurs related to a required field.
Required Help	The unique keyword for use in a context-sensitive help system that identifies the help message for the Required property. See the Help Context entry in Specifying Data Server Properties earlier in this chapter for specifics.
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 code block that takes the field name as an argument. The code block must return a logical value that must evaluate to TRUE before the data entered will be accepted in the field. You might, for example, use a table lookup function to determine if a key value is valid, as in {|KeyField| ValKeys->DBSeek(KeyField)}.
Validation Diagnostic	The message that should appear when an error occurs related to the validation rule.
Validation Help	The unique keyword for use in a context-sensitive help system that identifies the help message for the Validation property. See the Help Context entry in Specifying Data Server Properties earlier in this chapter for specifics.

Note: Field properties are inherited when you create a data window designed to work with the data server. Then, when the data server is accessed via the data window at runtime, the appropriate description is displayed in the window's status bar and all data validation (such as type, range, and picture) is performed automatically as data is entered in the individual data controls. See **Chapter 8: Creating Data-Aware Windows** for information about linking data servers to a data window and linking associated field specifications to data controls in the window.

Adding Index Files

CA-Visual Objects allows you to create and set multiple index files for data servers created with the DB Server Editor. Both the single-order and multi-order index file paradigms are supported. Index files provide a means of effectively using and maintaining the information stored in a database by allowing you to logically order the records according to a key value. The use of index files greatly speeds data access time when searching for a key value.

Creating a New Index File

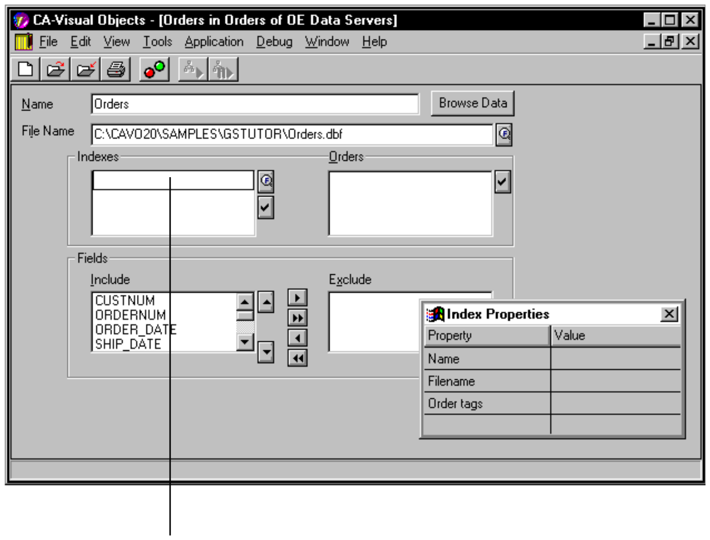
The process for creating an index file is similar to the process for adding a new field:

1. In the Indexes list box, click on any entry.

   The Properties window changes so that it relates to indexes. If there are no index files included in the data server, which is the case for the Orders data server, the DB Server Editor indicates the selected, empty entry with a dotted gray line.

2. Choose the Add Item command from the Edit menu.

   The DB Server Editor opens an edit control for the new entry:


CA-Visual Objects - [Orders in Orders of OE Data Servers]
File Edit View Tools Application Debug Window Help
Browse Data
Indexes
Orders
Fields
Include
Exclude
Index Properties

Name: Orders

File Name: C:\CAVO20\SAMPLES\GSTUTOR\Orders.dbf

Include: CUSTNUM, ORDERNUM, ORDER_DATE, SHIP_DATE

Property	Value
Name	
Filename	
Order tags	

Typehere(orusetheFindbutton)

Notice that the Properties window has changed to the Index Properties window.

3. Type the name of a new index file (for example, **ORDCUST.NTX**) and press Enter.
4. Repeat steps 2 and 3, entering **ORDERNUM.NTX**.

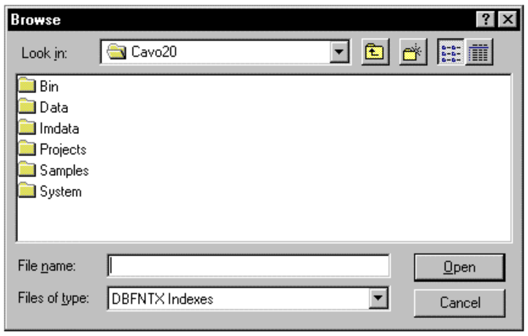
   As soon as you press Enter, the DB Server Editor automatically fills in the Index Properties window and adds the indexes to the Orders list box.

Importing Index Files

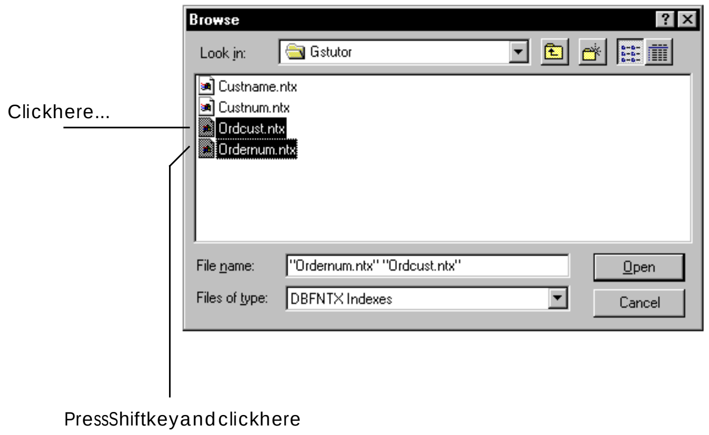
However, you can just as easily import the index files associated with ORDERS.DBF. To do this:

1. Click the Find button to the right of the Indexes list box.

   A standard Browse dialog box for indexes appears:

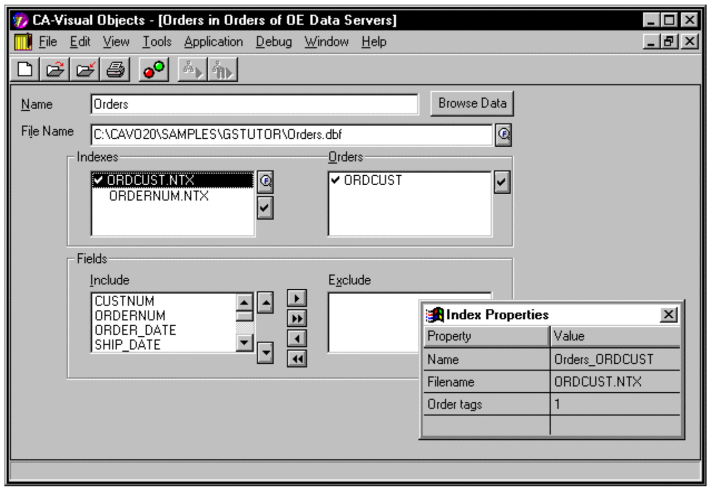


2. Choose ORDCUST.NTX and ORDERNUM.NTX from the GSTUTOR folder:



3. Click Open.

The DB Server Editor automatically fills in the Index Properties window and adds the indexes to the Orders list box:


CA-Visual Objects - [Orders in Orders of OE Data Servers]
File Edit View Tools Application Debug Window Help
Browse Data
Indexes
Orders
Fields
Include
Exclude
Index Properties

Name: Orders

File Name: C:\CAVO20\SAMPLES\GSTUTOR\Orders.dbf

Indexes: ✓ ORDCUST.NTX, ORDERNUM.NTX

Orders: ✓ ORDCUST

Include: CUSTNUM, ORDERNUM, ORDER_DATE, SHIP_DATE

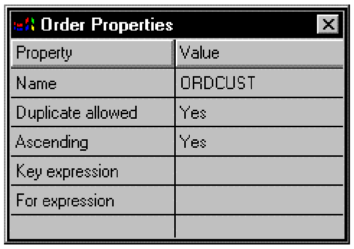
Property	Value
Name	Orders_ORDCUST
Filename	ORDCUST.NTX
Order tags	1

Notice that the Index Properties window displays just three properties: Name, Filename, and Order Tags. The last property, Order Tags, indicates the number of orders within the selected index. In this case, we defined the actual index file as an .NTX file; therefore, the number of Order Tags is 1, and only a single order is listed in the Orders list box.

Also notice the check marks (√) next to ORDCUST.NTX name in the Indexes list box and next to the ORDCUST order in the Orders list box. These check marks define the logical order in which the database file is processed, and this is called the *controlling order*.

4. Now click on ORDCUST in the Orders list box.

   The Properties window changes, displaying default values for some of the order-related properties:



Property	Value
Name	ORDCUST
Duplicate allowed	Yes
Ascending	Yes
Key expression	
For expression	

5. Enter the key expression for the order (for example, **CustNum**).
6. Press Enter.
7. Close the DB Server Editor and save your changes.

At this point all requirements have been met and the index file specification is complete. You can specify additional properties for this order by referring to the Specifying Index File and Order Properties section below. You can also add additional index files by repeating steps 1–6. Each subsequent index file is added to the end of the list.

Adding an Order

When working with multi-order index files, the process for creating the first order is the same as described above. To add subsequent orders to a multi-order index file:

1. In the Orders list box, click on any entry. Then, choose the Edit Add Item menu command.

   The DB Server Editor opens an edit control for a new entry.

2. Type the name of the order (for example, **ORDNUM**) and press Enter.
3. Enter the key expression for the order (for example, **OrdNum**).

Note: Whether an index file supports single or multiple orders per file depends on the RDD, which you determine using the data server's Driver property. Unless the driver you have specified supports multiple orders, you will not be able to define more than one order. See Specifying Data Server Properties earlier in this chapter for more information.

Specifying Index File and Order Properties

Index files can have the following properties in the DB Server Editor:

Name	A unique name for the index, such as ORDERS_ORDCUST.
Filename	The path (drive, directory, and file name) of the actual index file. This file name will be used, for example, by the File Export or File Export Index menu commands.
Order Tags	Displays the total number of orders (or tags) defined in an index file. This is a read-only field.

An index file's orders can have the following properties in the DB Server Editor:

Name	A unique name for the order, such as ORDCUST.
Duplicate Allowed	Indicates whether the order will have unique key values. The default is Yes, indicating that all records will be included in the order. If you change it to No, only records with unique key values will be included in the order.
Ascending	Indicates whether the key values in the order will be sorted as ascending or descending. The default is Yes, indicating ascending order. If you change it to No, the key values will be sorted in descending order.

Key Expression	The key expression on which the order is based. Its resultant key value determines the logical position of the associated record in the database when this is the controlling order (for example, CustNum).
For Expression	A condition on which to base this order (for example, Amount_Due > 1000.00). By default, none is specified and all records are included in the order. If you specify a For Expression, only records that meet the condition are included in the order.

Generating Corresponding Files

After creating a new data server according to the instructions in this section, the next logical step is to generate the associated database and index files. To generate all associated files at once, choose the File Export menu command. To generate only one index file, highlight it in the Indexes list box and choose the File Export Index menu command. (For more information, see Exporting Database and Index Files later in this chapter.)

Generating Code

When you save a data server, CA-Visual Objects automatically generates:

- A DB server entity for the data server

  You can double-click on this entity in the Repository Explorer's list view pane to begin editing it with the DB Server Editor.

- A DBServer subclass using the data server name
- A field spec entity for each field specification defined for the data server

  You can double-click on any field spec entity in the Repository Explorer's list view pane to begin editing it with the FieldSpec Editor.

- A FieldSpec subclass for each field specification in the data server
- Other default source code that can be modified

Note: Code generation in the DB Server Editor is now template-based. For complete details, refer to the *CA-Visual Objects 2.7 Software Development Kit*.

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 or index files associated with it. You can also change the data server name and its associated database file. Certain changes that you make may be reflected in the associated index files and database file structures if you issue a File Export menu command, all of which are explained in the following sections.

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.

Other Properties

To change any other property associated with a data server:

1. Click on the Name or File Name edit controls.

   The Properties window changes so that it relates to the data server.

2. Click on the property that you want to change in the Properties window and specify the new value.

See Specifying Data Server Properties for more information.

Editing Fields

If there are fields in the Include list box, you can change their properties, change their order, delete them from the list, or exclude them from use by the data server.

Changing Field Properties

To change any property associated with a field, follow these basic steps:

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 FieldSpec Properties window and specify the new value.

Order Entry Examples

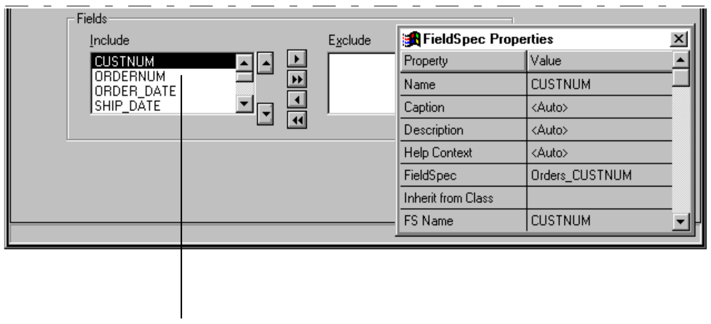
For example, if you did not follow the tutorial in the *Getting Started* guide, you should modify some of the field properties now for the Order Entry sample application.

First, you will reuse field specifications already defined for the Customer data server in the new data server, Orders, instead of defining them over again:

1. Double-click on the Orders DB server entity in the Repository Explorer's list view pane.

   The DB Server Editor appears.

2. Select the CUSTNUM field in the Include list box:

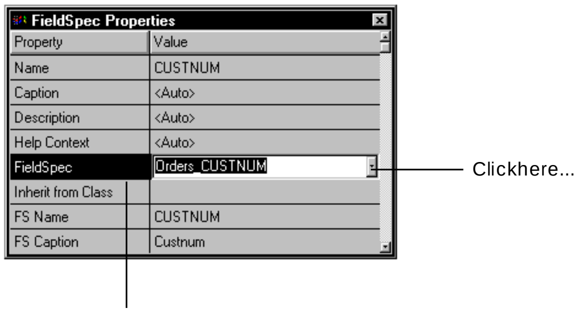


Property	Value
Name	CUSTNUM
Caption	<Auto>
Description	<Auto>
Help Context	<Auto>
FieldSpec	Orders_CUSTNUM
Inherit from Class	
FS Name	CUSTNUM

CUSTNUMselectedinOrdersdataserver

   As this field is identical to the CUSTNUM field in the Customer data server, you can take advantage of some of its field property definitions.

3. Click on the FieldSpec property in the FieldSpec Properties window:



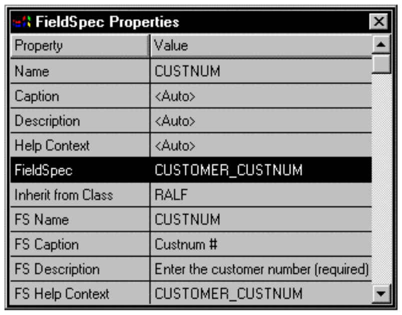
Property	Value
Name	CUSTNUM
Caption	<Auto>
Description	<Auto>
Help Context	<Auto>
FieldSpec	Orders_CUSTNUM
Inherit from Class	
FS Name	CUSTNUM
FS Caption	Custnum

FieldSpecpropertyselected

4. Click on the Down arrow that appears in its value cell, and select CUSTOMER_CUSTNUM from the list of all field specifications defined to the OE Data Servers library.

   In this step, you are importing properties from the field specification defined for the CUSTNUM field in the Customer data server and simply reusing them in the Orders data server.

For example, the FS description property in this data server is updated with the text defined in the Customer data server (this property was blank before):



FieldSpec Properties

Property	Value
Name	CUSTNUM
Caption	<Auto>
Description	<Auto>
Help Context	<Auto>
FieldSpec	CUSTOMER_CUSTNUM
Inherit from Class	RALF
FS Name	CUSTNUM
FS Caption	Custnum #
FS Description	Enter the customer number (required)
FS Help Context	CUSTOMER_CUSTNUM

The same is true for the Required and Validation properties.

5. Repeat the above steps to associate the SHIP_STATE field with the CUSTOMER_STATE field specification and the SHIP_ZIP field with the CUSTOMER_ZIP field specification.

6. Click on the Save button in the DB Server Editor toolbar to save your work so far.

Now you need to modify some properties for several other fields in the Orders data server:

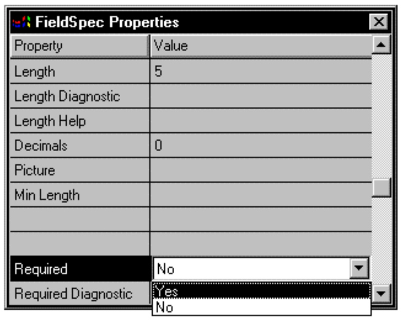
1. First, select ORDERNUM in the Include list box.

   The DB Server Editor highlights the field and displays the appropriate FieldSpec Properties window.

2. Click on the FS Caption property in the FieldSpec Properties window, replace the current contents of its value cell with **Order #**, and press Enter.

3. Click on the FS Description property, type **Enter the order number (required)** in its value cell, and press Enter.

4. Select the Required property, click on the Down arrow that appears in its value cell, and choose Yes from the list box:



FieldSpec Properties

Property	Value
Length	5
Length Diagnostic	
Length Help	
Decimals	0
Picture	
Min Length	
Required	No
Required Diagnostic	Yes / No

   Changing this property to Yes will require the end user to type a value for this field in the Order Entry application.

5. Click on the Required Diagnostic property just below, type **You must enter an order number**, and press Enter.

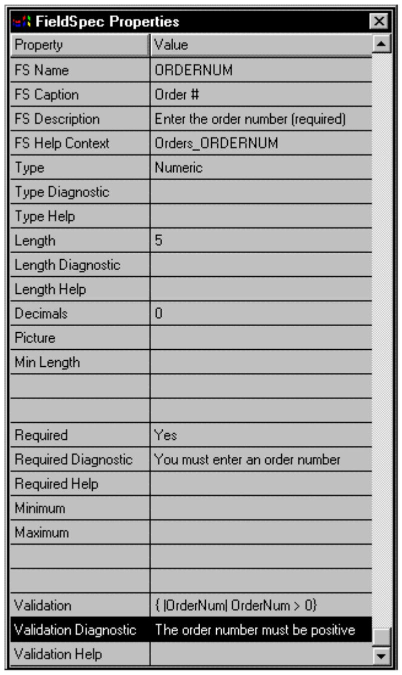
If the end user attempts to skip the OrderNum field, this message will display.

6. Scroll down to the Validation property, type **{ |OrderNum| OrderNum > 0}** in its value cell, and press Enter.

   This validation rule—in the form of a code block—will require the end user to enter a positive number. (See the "Code Blocks" chapter in the *Programmer's Guide* for more information.)

7. Click on the Validation Diagnostic property just below, type **The order number must be positive**, and press Enter.

   Your FieldSpec Properties window for ORDERNUM should now look something like this:



FieldSpec Properties

Property	Value
FS Name	ORDERNUM
FS Caption	Order #
FS Description	Enter the order number (required)
FS Help Context	Orders_ORDERNUM
Type	Numeric
Type Diagnostic	
Type Help	
Length	5
Length Diagnostic	
Length Help	
Decimals	0
Picture	
Min Length	
Required	Yes
Required Diagnostic	You must enter an order number
Required Help	
Minimum	
Maximum	
Validation	{ |OrderNum| OrderNum > 0}
Validation Diagnostic	The order number must be positive
Validation Help	

8. Click on the Save toolbar button.

Similarly, make the following changes to the ORDERPRICE field if you have not done so already:

1. Select the ORDERPRICE field in the Include list box.
2. Click on the FS Caption property, edit the current contents of its value cell to read **Order Price**, and press Enter.
3. Click on the FS Description property, type **Enter the order price** in its value cell, and press Enter.

4. Scroll down to the Picture property, click on its value cell, type **$$$$$$$.99**, and press Enter.

   The specified picture clause will cause the value to display with leading dollar signs.

5. Click on the Save toolbar button.

See Specifying Field Properties for more information about each available property.

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 window for this data server. (See **Chapter 8: Creating Data-Aware Windows** for information about the Window Editor's Auto Layout feature.) This Include list box order also determines the order in which the fields are created if you generate a .DBF file using the File Export menu command.

Deleting a Field

To delete a field from the Include list box:

1. Click on the field name in the Include list box.
2. Choose the Edit Delete Item menu command.

   Alternatively, press Esc and then press Del.

Excluding a Field

Deleting the field from the Include list box, however, is not necessary. You can simply exclude the field from use by this 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.

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. As a shortcut, you can move all fields from one list to the other by clicking the appropriate double arrow button.

Fields that are in the Exclude list box will be inaccessible when using this data server. A reference to an excluded field in your application will result in a runtime error.

Note: Changes that you make to the Name, Type, Length, and Decimals field properties, as well as deletions from the Include list box, will be reflected in the database file structure if you issue a File Export menu command. Excluded fields, however, will not.

Editing Index Files and Orders

You can change the properties of an index file, delete an index file from the Indexes list box, change the controlling order, change the properties of an order, and delete an order from an index file. This section describes each of these processes.

Changing Index File and Order Properties

To change any property associated with an index file or one of its orders:

1. Click on the index file name in the Indexes list box.

   The DB Server Editor highlights the file 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 Index File and Order Properties for more information.

Changing the Controlling Order

As stated earlier in this chapter, CA-Visual Objects allows you to set multiple index files for a single data server and, depending on the RDD, several orders can be defined for a single index file. All index files that are set will be opened automatically whenever you access the associated database file via this data server, and all orders in all of these index files will be properly maintained as changes to the database file are made.

However, there can only be one controlling order (indicated with a √ in the Indexes and Orders list boxes). If you want, you can change the logical order in which the database file is processed by changing the controlling order. To do this:

1. In the Indexes list box, select the index file that defines the order you want to make the controlling order (either click on it with the mouse or scroll to it with the Direction keys).

2. Click on the check mark button next to the Indexes list box.

   The system will place a √ next to that index.

3. If the index defines only one order, it will automatically become the controlling order. If the index has multiple orders, the first order in the index will be the default controlling order. Optionally highlight a different order in the Orders list box.

4. Click on the check mark button to the right of the Orders list box.

   The system will place a √ next to the new controlling order.

Deleting an Index File

To delete a file from the Indexes list box so that it will no longer be used by this data server:

1. Click on the index file name in the Indexes list box.
2. Choose the Edit Delete Item menu command.

   Alternatively, press Esc and then press Del.

Deleting an Order

To delete a file from the Indexes list box so that it will no longer be used by this data server:

You can remove individual orders from a multi-order index file:

1. Click on the order name in the Orders list box.
2. Choose the Edit Delete Item menu command.

   Alternatively, press Esc and then press Del.

Note: Changes that you make to orders, including Key Expression, For Expression, deletions, and additions will be reflected in the index file if you issue a File Export or File Export Index menu command. Deleting an index file from the Indexes list box, however, does not delete it from the disk, it simply prevents this data server from accessing the index file.

Importing Database and Index Files

So far, you have learned how to create data servers, fields, and indexes using the DB Server Editor, with the idea in mind that you could later generate the corresponding database and index files based on the properties defined in the data server. Greater emphasis has been given, however, to *importing* predefined data servers from a library, database files, and index files; importing saves you valuable time.

Database Files

For example, after importing an existing database file, all of the fields in the database structure are included in the data server that you are defining. You are free to include and exclude any of these predefined fields, modify them, and add new fields (as shown earlier in Adding Fields and Editing Fields).

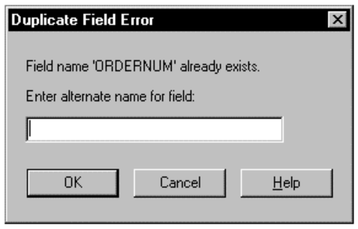
Furthermore, you can even import fields from another database file to add to the current data server. For example, you could add the fields in DETAIL.DBF to the Orders data server:

1. Double-click on the Orders DB server entity in the list view pane of the Repository Explorer to start the DB Server Editor.
2. Click the Find button associated with the File Name edit control.

   The Import dialog box appears.

3. Select DETAIL.DBF from the GSTUTOR folder in the SAMPLES directory.

   The Duplicate Field Error dialog box appears:



Duplicate Field Error

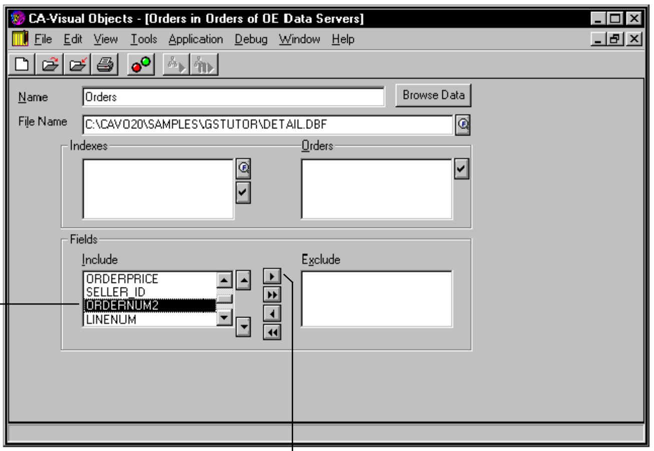
Field name 'ORDERNUM' already exists.

Enter alternate name for field:

4. Enter an alternate name for the indicated field (for example, **ORDERNUM2**).
5. Click OK.

   All of the fields in DETAIL.DBF (including the renamed ORDERNUM field) are added to the Orders data server.

6. Highlight ORDERNUM2 in the Include list box, and then click the Right arrow button:



CA-Visual Objects - [Orders in Orders of OE Data Servers]

Name: Orders

File Name: C:\CAVO20\SAMPLES\GSTUTOR\DETAIL.DBF

Include: ORDERPRICE, SELLER_ID, ORDERNUM2, LINENUM

   The duplicate field is added to the Exclude list box and becomes inaccessible when using this data server.

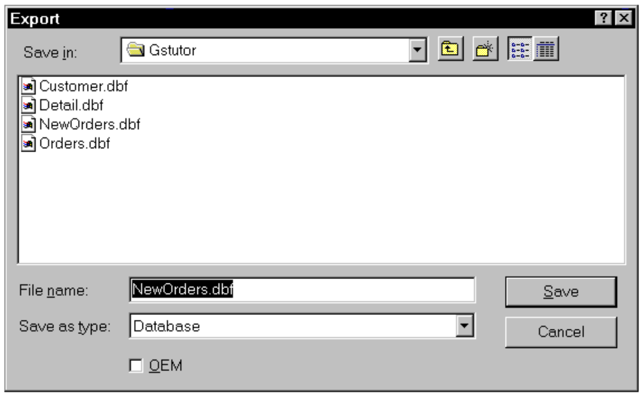
7. Click Save to save all of your changes, and then click the Close button.
8. Click the Build toolbar button to rebuild the data server.

Importing Index Files

Just as importing an existing database file is faster than creating one, so is importing an index file. After importing an existing index file, all of its orders are added to the data server that you are defining. You are free to make modifications to the index file itself or to any of its orders (as shown previously in Editing Index Files and Orders).

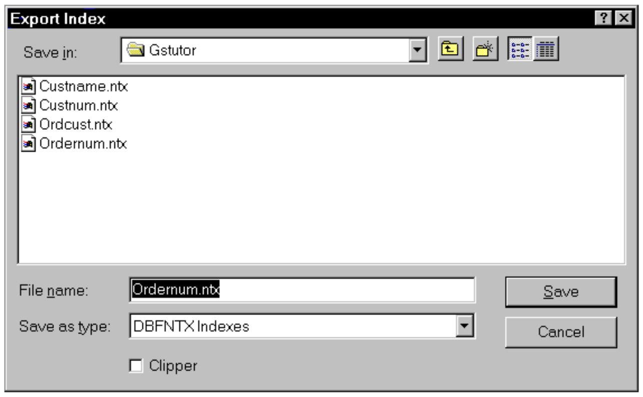
Exporting Database and Index Files

As mentioned earlier, you can create and modify database file structures and index files, and then export them while in the DB Server Editor. Choose the File Export menu command to create a .DBF file structure with no associated records, using the Export dialog box. For example:



Important! *Before exporting any file, you should make a backup of it; if you export without changing the name of the .DBF file, you will lose all records in the file.*

You can also choose the File Export Index menu command to export a single index file at a time, using the Export Index dialog box shown here:



Note: The implications of how changes to the data server, field specification, and index/order properties will affect the resulting exported files have already been discussed in detail throughout this section. For detailed information about the Export and Export Index dialog boxes, see the online help.

Using the SQL Editor

The SQL Editor is used for creating and editing SQL data servers. In this section, you will learn how to:

- Define a data server for a table or view in an existing SQL database
- Specify a WHERE clause and an ORDER BY clause
- Specify additional properties for the fields
- Modify an SQL data server

In order to connect to a data source using the SQL Editor, the appropriate ODBC engine and driver must be installed on your computer, and the data source must be properly configured.

CA-Visual Objects comes packaged with the ODBC kernel and several ODBC drivers. The CA-Visual Objects package does not, however, include complete engines for all of these drivers—you must obtain these from the appropriate vendor in order to use the supplied drivers. CA-Visual Objects does supply the CA-OpenIngres / Desktop, which provides complete DBMS for the Windows or Windows NT desktop environment and creates an ODBC data source, CA-OpenIngres / DT Demo, so that you can experiment with the SQL Editor.

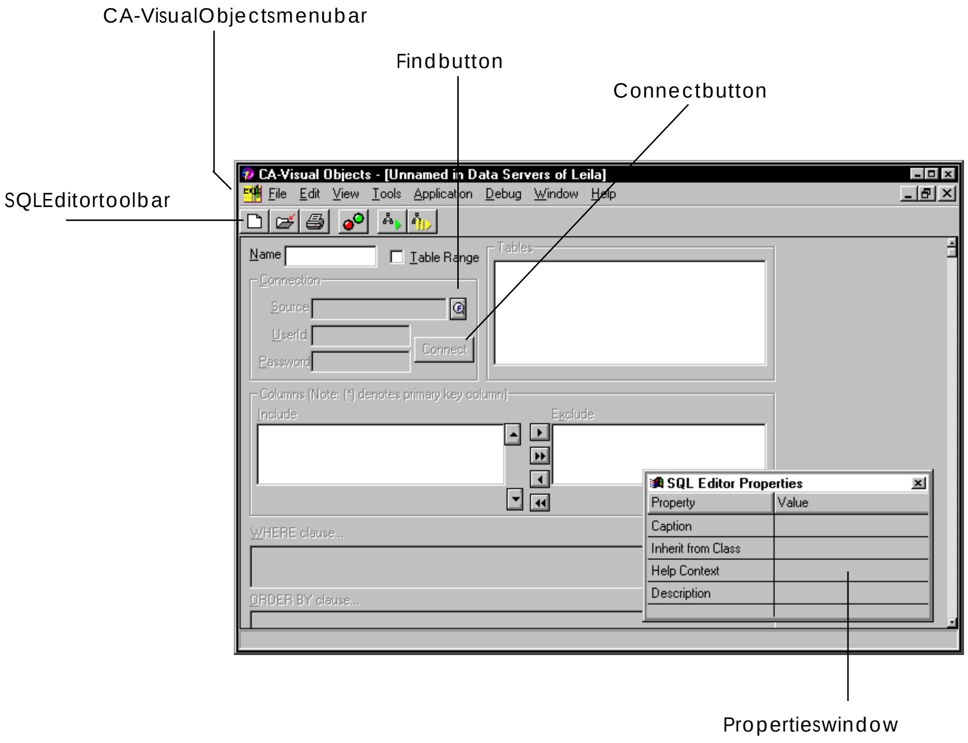
Note: When you install various drivers, a separate help file for each driver will also be installed. (Refer to the “Installing and Starting CA-Visual Objects” chapter of the *Getting Started* guide if you are interested in installing these components.)

Keep in mind, also, that your end users must have the appropriate engine and driver installed to use the resulting data server in an application. The supplied drivers can be distributed with your application as described in the “Operating Environment” chapter of the *Programmer’s Guide.* However, database engine redistribution rights are determined by the individual engine manufacturers. In most cases, the end user must own a separate, licensed copy of the engine software.

Important! *To use a data server created using the SQL Editor, the SQL Classes and System Classes libraries must be included in the search path sequence defined for your application. (For more information on setting a search path, please see the* ***Setting the Search Path*** *section in “Using the Repository Explorer.”)*

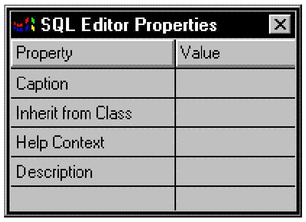
The SQL Editor

The SQL Editor is very similar to the DB Server Editor—it also has its own toolbar (which is identical to the DB Server Editor toolbar, described earlier) and an associated Properties window. When first loaded for a new data server entity, the SQL Editor looks as follows:



The Properties Window

The SQL Editor also features a floating Properties window that is initially set to allow you to specify properties for the current data server:


SQL Editor Properties

Property	Value
Caption	
Inherit from Class	
Help Context	
Description	

Like the Properties window in the DB Server Editor, this one takes on different roles depending on the current focus of the editor. After you create or modify the data server using the SQL Editor Properties window, you can modify its fields using the FieldSpec Properties window. (See the Specifying SQL Editor Properties and Specifying Field Properties sections later in this chapter for more information about this window.)

Other Options

The remainder of the options in the SQL Editor workspace allow you to specify various properties for the current data server. See Defining an SQL Server next for details.

Defining an SQL Server

Defining a data server in the SQL Editor requires you to do the following:

- Create the data server by specifying its properties, connecting to an SQL database, and choosing a table to use
- Optionally specify a WHERE clause
- Optionally specify an ORDER BY clause
- Optionally specify additional properties for the fields

Creating an SQL Server

Creating an SQL data server requires that you have an existing SQL database available. To create a data server based on an existing SQL database, you need to perform the following steps:

1. Start the SQL Editor.

   Like all CA-Visual Objects tools, the SQL Editor is accessed using either the Tools menu or the New Entity toolbar button.

   The SQL Editor appears.

2. In the Name edit control, type the name of the data server (for example, **Customer_Order**). This information is included in the data server's automatically generated hyperlabel.

   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.

   This name is used to create the SQL server entity and to subclass the data server in the code generated by the SQL Editor. For example:

   ```
   CLASS Customer_Order INHERIT SQLSelect
   ```

   It is also the name used to refer to this data server in your program code. For example:

   ```
   oSQLCust := Customer_Order{}
   ```

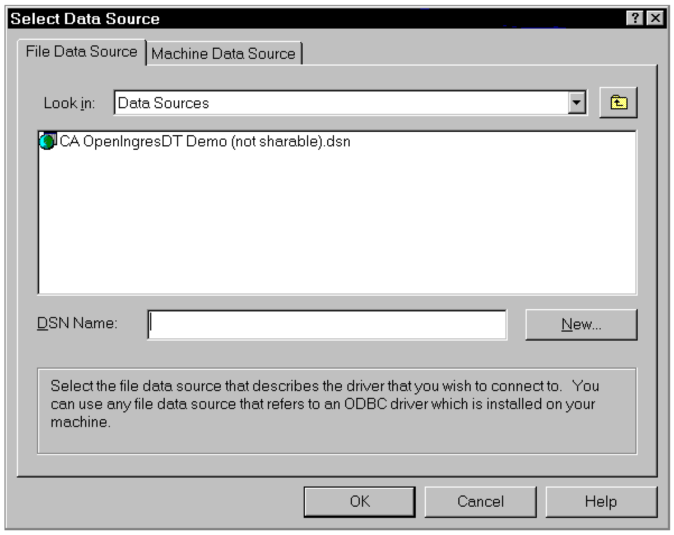
3. Optionally set additional data server properties using the SQL Editor Properties window.

   See Specifying SQL Editor Properties below.

4. Select the Table Range check box if you want to limit the available tables when you connect to a database.

5. In the Connection group box, type the name of an SQL database in the Source edit control, and enter the appropriate User ID and Password.

   Alternatively, click on the Find button to the right of the Source edit control to select an ODBC source (for example, CA-OpenIngres / DT Demo) from the Select Data Source dialog box:



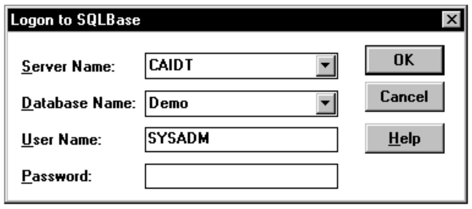
Select the file data source that describes the driver that you wish to connect to. You can use any file data source that refers to an ODBC driver which is installed on your machine.

   If the data source you need is not listed here, you can add a new one by clicking on the New button. For more information, see your ODBC documentation.

   Note: The ODBC source name must already have been created and configured using the Microsoft ODBC Administration Utility (or the equivalent).

6. Click OK.

   A standard logon dialog box appears, which varies according to your data source connection. For example:



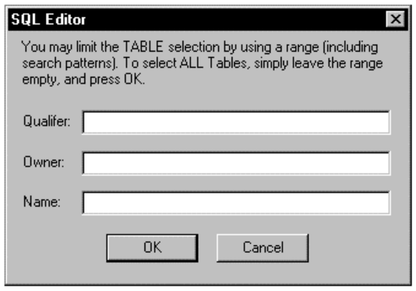
Server Name: CAIDT
Database Name: Demo
User Name: SYSADM
Password:

7. Enter the appropriate information, and click OK.

   You are returned to the SQL Editor.

8. Click on the Connect button.

9. If you selected the Table Range option, you must complete the following dialog box to specify which tables to include:



SQL Editor

You may limit the TABLE selection by using a range (including search patterns). To select ALL Tables, simply leave the range empty, and press OK.

Qualifer:

Owner:

Name:

After clicking OK, the tables associated with this database appear in the Tables list box to the right.

10. In the Tables list box, click on one or more tables to include in this data server (for example, order.Customer).

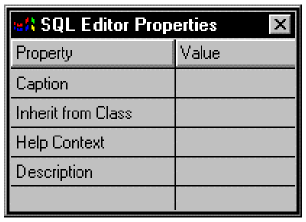
    The fields associated with the selected table(s) appear in the Include list box, with primary key fields indicated by (*).

11. Choose the Save toolbar button to save the SQL server entity.

Note: This last step can be repeated whenever you make changes to the data server and want to save your work without closing the SQL Editor.

Specifying SQL Editor Properties

You can define additional properties for a data server using the SQL Editor Properties window:



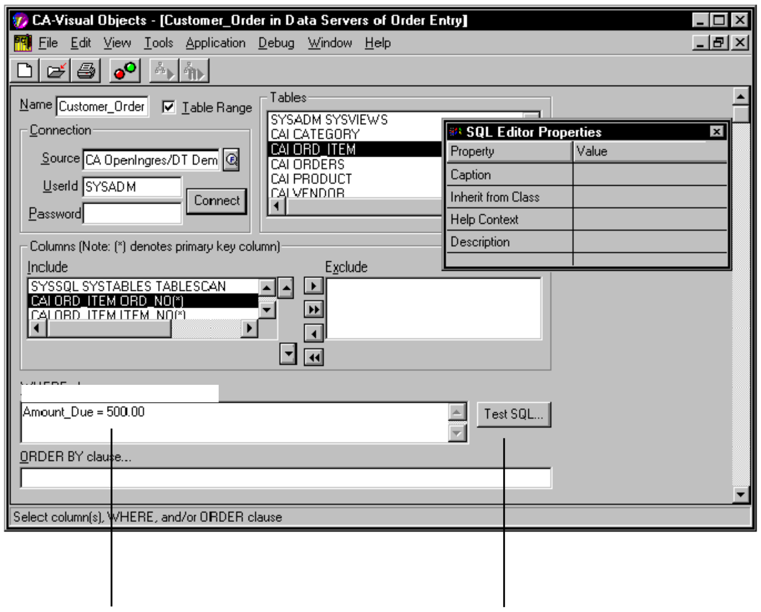
SQL Editor Properties

Property	Value
Caption	
Inherit from Class	
Help Context	
Description	

The four properties in this dialog box, Caption, Inherit from Class, Help Context, and Description, are identical to the ones described for the DB Server Editor in the Specifying Data Server Properties section.

Defining a WHERE Clause

To define a WHERE clause, click on the WHERE Clause edit control and enter the clause (for example, **State = 'CA'** or **Amount_Due = 500.00**):



This clause defines the selection criteria for the database when accessed with this data server.

Tip: To check the syntax of the WHERE clause, click on the Test SQL button. This gives you a quick way to make sure that the clause is valid without building the entire application.

Defining an ORDER BY Clause

To define an ORDER BY clause, click on the ORDER BY Clause edit control and type the clause. This clause defines the ordering criteria for the database when accessed with this data server, for example, **Cust_Num** or **Cust_Name**.

Specifying Field Properties

Using the FieldSpec Properties window, you can specify any field property that is not directly associated with the database. The Field Properties are the same as those described under Specifying Field Properties in Using the DB Server Editor.

Note: You cannot, however, change properties that would affect the structure of the underlying SQL table; that is, Name, Type, Length, Decimals, and Required are read-only fields.

Generating Code

When you save a data server, CA-Visual Objects automatically generates:

- An SQL server entity for the data server

  You can double-click on this entity in the Repository Explorer's list view pane to begin editing it with the SQL Editor.
- An SQLSelect subclass using the data server name
- A field spec entity for each field specification defined for the data server

  You can double-click on any field spec entity in the Repository Explorer's list view pane to begin editing it with the FieldSpec Editor.
- A FieldSpec subclass for each field specification in the data server
- Other default source code that can be modified

Modifying an SQL Server

After you have defined a data server, you can modify it by changing its properties or the properties of its fields. You can also change the data server name, its associated table or source database, and its WHERE and ORDER BY clauses.

Editing SQL Editor Properties

Name	To change the name of the data server, type a new name in the Name edit control.
Connection	To change the associated source database, either type a new Source, User ID, and Password in the Connection group box and click on the Connect button, or connect to a new source database by clicking on the Find button. This will clear the Include field list and replace the Tables list with the table names from the new source database.

Note: You can optionally check the Table Range check box before connecting to a new database to limit the table selection using the SQL Editor dialog box.

Tables

To associate another table, click on the new table name in the Tables list box. This will add the field list from the selected table to the prior Include field list.

To remove an associate table that is currently highlighted in the Tables list box, click on the table name. The fields from the table will be removed from the Include field list.

WHERE/ORDER BY Clauses

To change the WHERE or ORDER BY clause, click on the appropriate edit control and type the new clause.

Other Properties

To change any other property associated with a data server:

1. Click on the Name edit control.

   The Properties window changes so that it relates to the data server.

2. Click on the property that you want to change in the Properties window and specify the new value.

See Specifying Data Server Properties for more information.

Editing Fields

If there are fields in the Include list box, you can change certain properties, change their order, or exclude them from use by the data server. Except for the fact that you cannot delete fields and cannot make structural changes to the underlying database in the SQL Editor, the instructions for making these changes are the same as those described in the Editing Fields section for the DB Server Editor earlier in this chapter.

Using the FieldSpec Editor

A field specification is a unique, independent entity in the CA-Visual Objects architecture. It allows you to specify and store a wide variety of properties for an item in your application, all in a single, manageable location. (Note that it is derived from the FieldSpec class.)

Typically, a field specification is "associated" with a field in a data server. Some of the properties you might define in such a field specification are a type, a length, a picture clause, and validation checks. In fact, when you create a data server in the DB Server Editor or the SQL Editor, the system automatically generates a field specification for every field in that server.

The automatic association of field specifications with data server fields is quite useful, because many of your fields will require formatting, validation, etc., and field specifications provide a convenient and intuitive way to define these properties.

In addition, many databases have similar (if not identical) fields, which require the same formatting, validation, etc. With the CA-Visual Objects architecture, you can design a single field specification that can be "attached" to each common field. For example, if you create a Salary field specification, you can simply reuse its properties when creating an EmpSalary field in a data server for an Employee database. You can then reuse its properties when creating a similar field for a data server for a Payroll database.

However, field specifications do not need to be associated with data server fields. For example, when designing a form in the Window Editor, you might add an edit control for which you want to specify a picture clause, a required flag, and a validation check. Even though the control is not linked to a data server field, you can attach a field specification to it to define these properties.

Field specifications, therefore, are useful in a variety of ways, saving you both time and resources. And, because of the nature of repository-based development, field specifications can easily be selected and reused throughout the system.

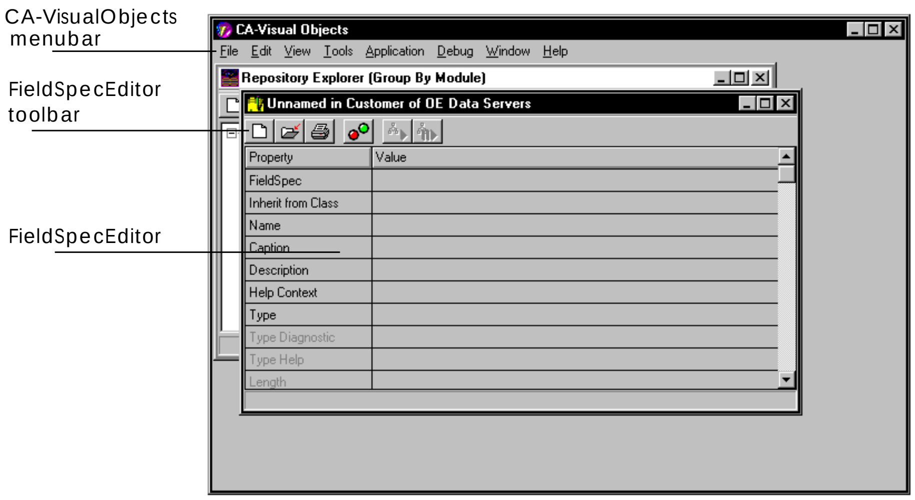
This section describes the FieldSpec Editor and explains how to:

- Define a new field specification
- Modify an existing field specification
- Copy properties from one field specification to another

The FieldSpec Editor

The CA-Visual Objects 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 CA-Visual Objects menu bar.

When first loaded for a new field spec entity, it looks as follows:


CA-VisualObjects menubar
FieldSpecEditor toolbar
FieldSpecEditor
CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Unnamed in Customer of OE Data Servers

Property	Value
FieldSpec	
Inherit from Class	
Name	
Caption	
Description	
Help Context	
Type	
Type Diagnostic	
Type Help	
Length	

Note that the FieldSpec Editor is similar in appearance to the FieldSpec Properties window in both the DB Server and the SQL Editors.

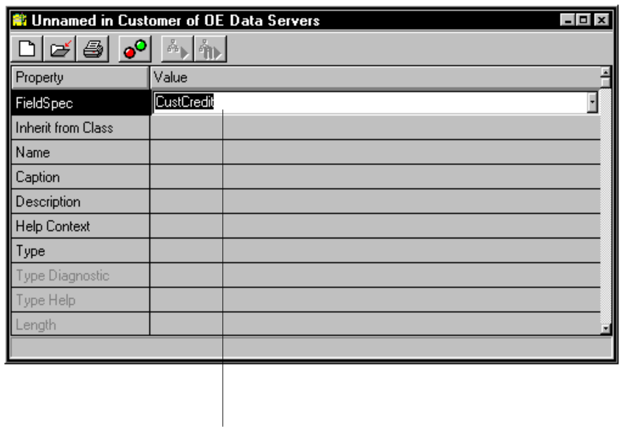
Defining a Field Specification

To define a new field specification that will be stored in the current module, you need to perform the following steps:

1. Start the FieldSpec Editor.

   Like all CA-Visual Objects tools, the FieldSpec Editor is accessed using either the Tools menu or the New Entity toolbar button.

2. In the FieldSpec value cell, type the name of the new field (for example, **CustCredit**):



Property	Value
FieldSpec	CustCredit
Inherit from Class	
Name	
Caption	
Description	
Help Context	
Type	
Type Diagnostic	
Type Help	
Length	

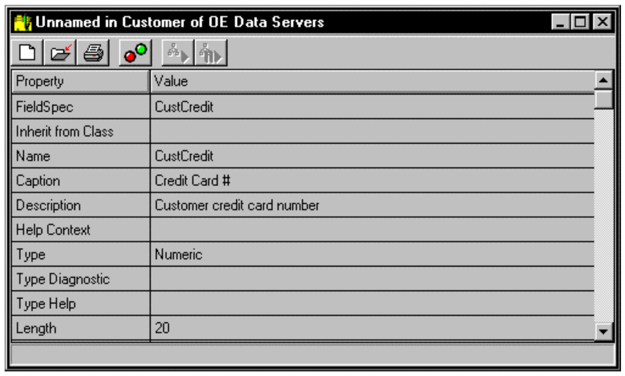
   You can enter a name using a maximum of 64 characters. The first character must be alphabetic or an underscore; the other characters can be alphanumeric and can include the underscore character.

   This name is used to create the field spec entity and to subclass the field specification in the code generated by the FieldSpec Editor. For example:

```
CLASS CustCredit INHERIT FieldSpec
```

   It is also used to associate this field specification with a field in a data server.

3. Enter the remaining properties for the field specification by clicking on the appropriate cell and specifying a value. For example:



Property	Value
FieldSpec	CustCredit
Inherit from Class	
Name	CustCredit
Caption	Credit Card #
Description	Customer credit card number
Help Context	
Type	Numeric
Type Diagnostic	
Type Help	
Length	20

Note: Refer to the online help for detailed information about new and updated field specification properties.

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. Note, however, that Name, Caption, and Description correspond to FS Name, FS Caption, and FS Description, respectively.

4. Choose the Save toolbar button to save the field specification.

Note: This last step can be repeated whenever you make changes to a field specification and want to save your work without closing the FieldSpec Editor.

Generating Code

When you save a field specification, CA-Visual Objects automatically generates a field spec entity. You can double-click on this entity in the Repository Explorer's list view pane to begin editing it with the FieldSpec Editor.

CA-Visual Objects also creates a subclass of the FieldSpec class using this same name, and an Init() method to initialize all the properties you defined.

Editing Field Specifications

Within the FieldSpec Editor, you can edit a field specification and copy properties from one field specification to another.

Modifying a Field Specification

Basic Steps

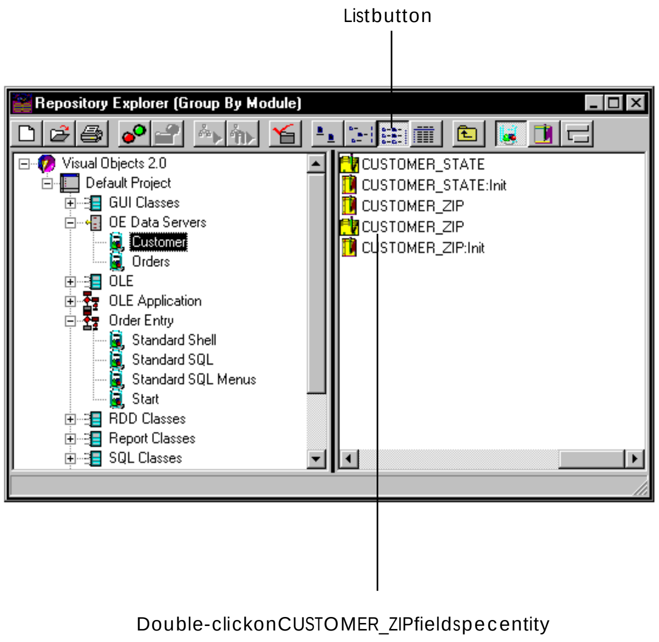
To modify any field specification defined for an application, follow these basic steps:

1. Start the FieldSpec Editor.
2. Click on the FieldSpec cell.
3. Type the name of an existing field specification, or 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.

Order Entry Example

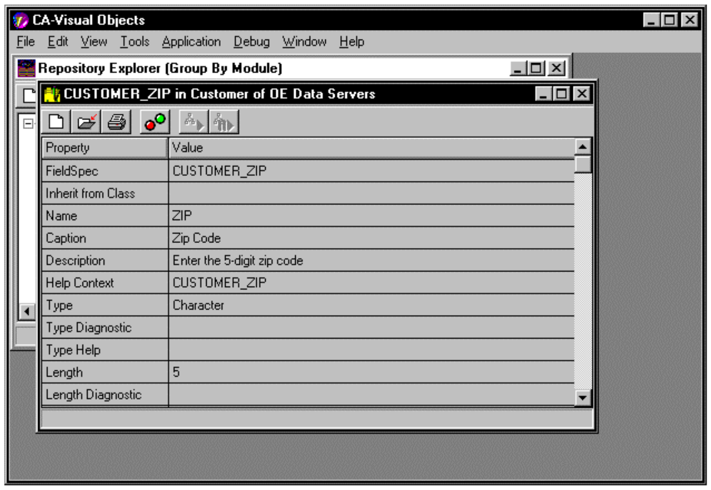
For example, if you did not follow the tutorial in the *Getting Started* guide, you should make the following small change to one of the field specifications shared by the Customer and Orders data servers.

1. Select the Customer module in the OE Data Servers library, and then double-click on the CUSTOMER_ZIP field spec entity:



Tip: Use the List button in the Repository Explorer's toolbar to assist you in your search for the specified field spec entity.

The FieldSpec Editor appears:

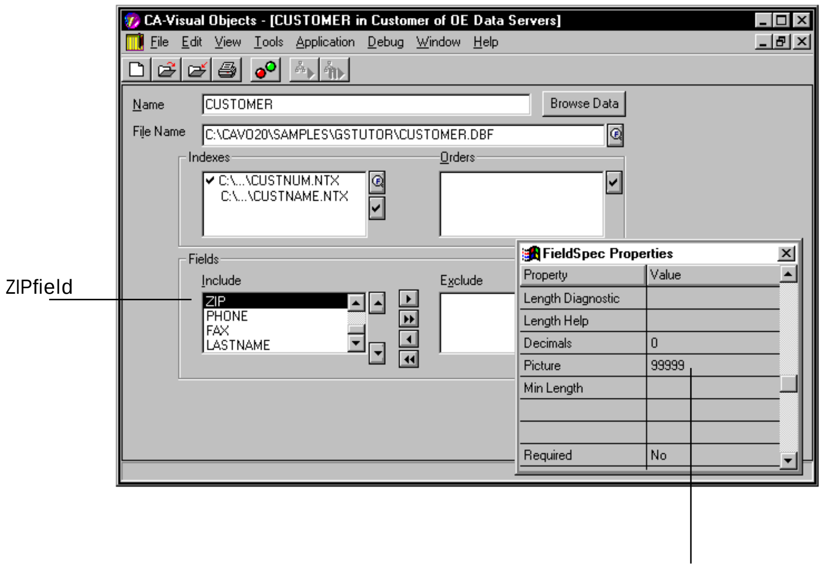

CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
CUSTOMER_ZIP in Customer of OE Data Servers

Property	Value
FieldSpec	CUSTOMER_ZIP
Inherit from Class	
Name	ZIP
Caption	Zip Code
Description	Enter the 5-digit zip code
Help Context	CUSTOMER_ZIP
Type	Character
Type Diagnostic	
Type Help	
Length	5
Length Diagnostic	

Note: The FieldSpec Editor is remarkably similar to the FieldSpec Properties window of the DB Server Editor when a field is selected. The only difference is that the field-specific properties—that is, Name, Caption, Description, and HelpContext—are not present.

2. Scroll to the Picture property and click on it.
3. Type **99999** in its value cell, and press Enter.
4. Close the FieldSpec Editor and save your changes.

This one change is propagated automatically to the two data servers that use the CUSTOMER_ZIP field specification to format their individual zip code fields. To see this for yourself, double-click either on the Customer or the Orders DB server entity in the Repository Explorer's list view pane. For example:


CA-Visual Objects - [CUSTOMER in Customer of OE Data Servers]
File Edit View Tools Application Debug Window Help
Browse Data
Indexes
Orders
Fields
Include
Exclude
FieldSpec Properties
ZIPfield
Picture clause value propagated to data server

Name: CUSTOMER
File Name: C:\CAVO20\SAMPLES\GSTUTOR\CUSTOMER.DBF

Indexes: ✓ C:\...\CUSTNUM.NTX, C:\...\CUSTNAME.NTX

Fields Include: ZIP, PHONE, FAX, LASTNAME

Property	Value
Length Diagnostic	
Length Help	
Decimals	0
Picture	99999
Min Length	
Required	No

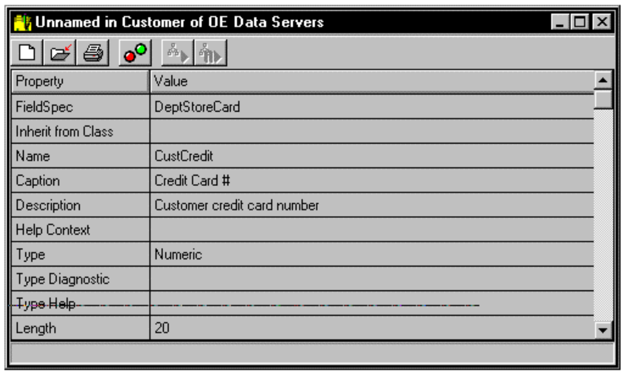
Copying a Field Specification

You can create a copy of a field specification and rename it, or copy the properties from one field specification to another easily using the FieldSpec Editor.

For example, if you want to create a *new* field specification called DeptStoreCard, copying the property values from the CustCredit field specification defined earlier:

1. Load the CustCredit field specification following the basic steps 1–3 in Modifying a Field Specification above.

2. Click on the FieldSpec value cell again, and replace CustCredit with DeptStoreCard:



Property	Value
FieldSpec	DeptStoreCard
Inherit from Class	
Name	CustCredit
Caption	Credit Card #
Description	Customer credit card number
Help Context	
Type	Numeric
Type Diagnostic	
Type Help	
Length	20

3. Choose Save.

The properties for the existing field specification, CustCredit, are copied to the new field specification, DeptStoreCard, but will remain intact in the original. At this point, you can change some of the properties in the copy, such as the description, if you wish.

Note: You can also use the Clipboard to cut, copy, and paste text in individual field specification properties using the Cut, Copy, and Paste commands on the Edit menu.

Tip: Field specifications, like all other entities, can easily be copied from one application to another within the Repository Explorer by pressing the Ctrl key and using the drag-and-drop technique. Note that only the field spec entity is copied and not the generated source code.

Printing

To print both the data server properties and the field properties generated by a data server, click on the Print toolbar button from within either the DB Server or SQL Editor.

To print the properties for a field generated by a field specification, click on the Print toolbar button from within the FieldSpec Editor.

Chapter

8

Creating Data-Aware Windows

Data windows and data dialog windows are data entry windows that are associated with *data servers*. Both are *data-aware*—each type "knows" about the data with which it is intended to operate via properties that you specify for each *data control* in its window.

As described earlier in the "Using the Window Editor" chapter, data windows and data dialog windows are quite similar. However, whereas a regular data window is an MDI child window, a data dialog window behaves like a modal dialog window. This means that the end user must respond to the window and close it before continuing with the application. Data dialog windows are typically used for small data entry tasks, such as password entry.

Note: In CA-Visual Objects 2.7, a new window type—OLEDataWindow—was added. An OLE data window is essentially the same as a regular data window, except that it contains additional logic for the deactivation of an in-place active OLE object by clicking anywhere in the window. Additionally, all text controls are now data aware. For more detailed information about OLE data windows and data aware controls, refer to the online help.

The Window Editor allows you to create data dialog windows and data windows using one of the following techniques:

- Customizing a DataWindow or DataDialog form by choosing a data server, manually placing controls on the form, and specifying properties for those controls
- Using the Auto Layout feature to choose a data server (or two related data servers) and automatically place fixed text and single-line edit controls on the form whose properties are based on the data server

Using either technique, the key to designing a data-aware window is in linking individual controls to fields defined in the data server.

In addition, you can specify a *field specification* for each control. You bind a control to a specific field using the field name as the name of the control. Because the field specifications store information designed specifically for use with a data window (such as captions, descriptions, help keywords, picture formatting, and validation rules), such links are quite useful. Data windows automatically inherit and use the property values from the field specifications, eliminating the need for you to specify this information repeatedly. The only controls you *cannot* make data-aware are fixed icons, fixed text, group boxes, push buttons, and radio buttons.

Creating a Data-Aware Window

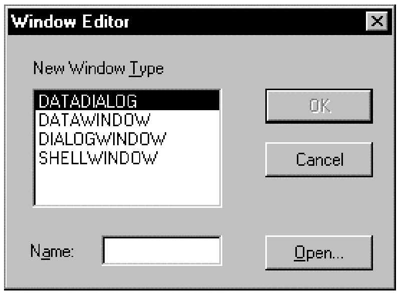
To create a window that is linked to a data server, such as the data window that "links" the Customer and Orders data servers in the Order Entry sample application, you need to perform the following steps:

1. Select the desired application in the Repository Explorer (for example, Order Entry).
2. Click the New Module toolbar button.

   The Create Module dialog box appears.
3. Type **App Windows** in the Enter Module name edit control, and choose OK.

   The App Windows module is added to Order Entry in the Repository Explorer's tree view pane.
4. Highlight the App Windows module in the tree structure, and then select the Window Editor command from the Tools menu.

   The Window Editor dialog box appears:

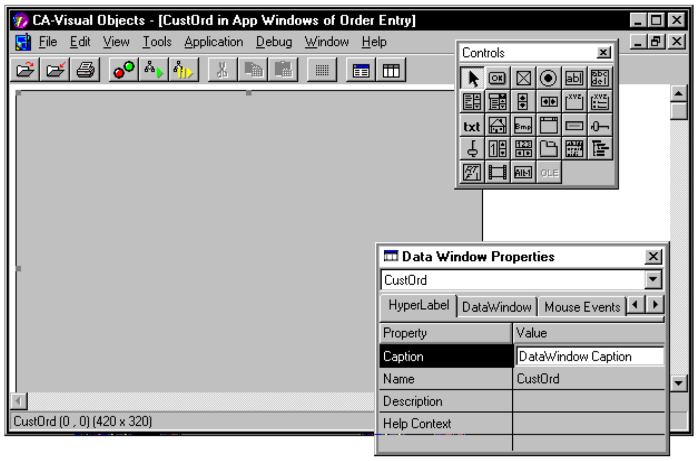


   Note: This dialog box has been updated in this version of CA-Visual Objects to reflect the addition of the new window type, OLEDATAWINDOW.

   See Creating a Window in "Using the Window Editor" for details about this dialog box.
5. In this instance, select DATAWINDOW in the New Window Type list box.
6. Enter **CustOrd** in the Name edit control.

7. Choose OK.

   The Window Editor appears:


CA-Visual Objects - [CustOrd in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Controls
Data Window Properties
HyperLabel
DataWindow
Mouse Events

CustOrd

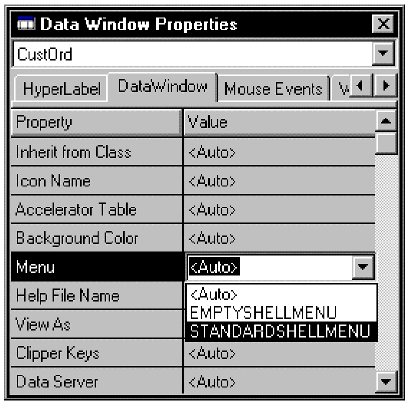
Property	Value
Caption	DataWindow Caption
Name	CustOrd
Description	
Help Context	

CustOrd (0 , 0) (420 x 320)

8. Specify the form's properties using the Data Window Properties window. For example, click on the Caption property and then type **Customer Orders** in its value cell.

   Note: As mentioned earlier in "Using the Window Editor," data windows and data dialog windows have several new properties, including Browser Inherits From, Columns Inherit From, Defer USE, Allow Server Close, and Quit on Close. Other properties have been updated. Therefore, refer to the online help for detailed information about data window and data dialog window properties.

9. Switch to the DataWindow tab, click on the Down arrow in the Menu property's value cell, and choose STANDARDSHELLWINDOW from the list box that appears:



CustOrd

Property	Value
Inherit from Class	<Auto>
Icon Name	<Auto>
Accelerator Table	<Auto>
Background Color	<Auto>
Menu	<Auto>
Help File Name	
View As	
Clipper Keys	<Auto>
Data Server	<Auto>

10. Associate a data server with the data window, using one of two methods: *Auto Layout* or *Properties.*

    These methods are described next in Associating Data Servers.

11. Choose the Save toolbar button to save the form.
12. Close the Window Editor.

    (You will return to the CustOrd data window shortly.)

Associating Data Servers

You can associate a data server with a data window or data dialog window in either one of two ways:

- 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 window 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. This is described in the next section, Using Auto Layout.

- 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 window or data dialog form. This is described in the section Customizing a Data-Aware Window later in this chapter.

Using Auto Layout

Using the Window Editor's Auto Layout feature is much easier than creating a customized data-aware window "from scratch." At the touch of a button, Auto Layout automatically creates a fixed text caption and single-line edit data control for every available field in the associated data server(s).

Tip: If you have an Xbase or SQL database on disk that is not yet known to the repository, simply bring up the DB Server or SQL Editor using the appropriate command from within the Tools menu, import the database structure, save it in the repository, and switch back to the Window Editor. See **Chapter 7: Defining Data Servers** and Field Specifications for more information on how to do this and on data servers and field specifications, in general.

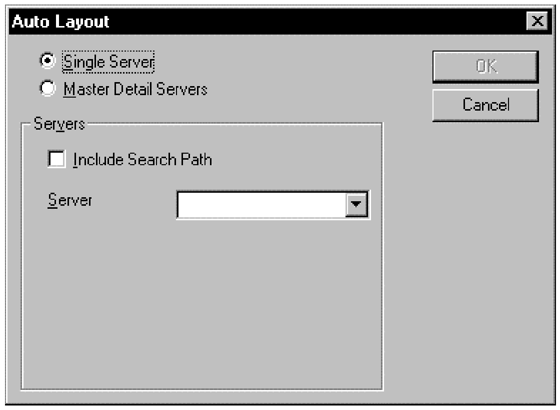
The Auto Layout feature presents you with two options for defining the type of data server link, *Single Server* or *Master Detail Servers*.

Single Server Option

The Single Server option is used to link a data-aware window to a single data server. The example below steps you through the creation of a data entry form, Inventory, that accesses a parts data server that you might want to create to extend the Order Entry sample application. (The Parts data server would be based on the DETAIL.DBF file located in the SAMPLES\GSTUTOR folder.)

1. From within the App Window module of Order Entry, start the Window Editor.
2. Create another data window and name it **Inventory**.
3. Click on the Auto Layout toolbar button.

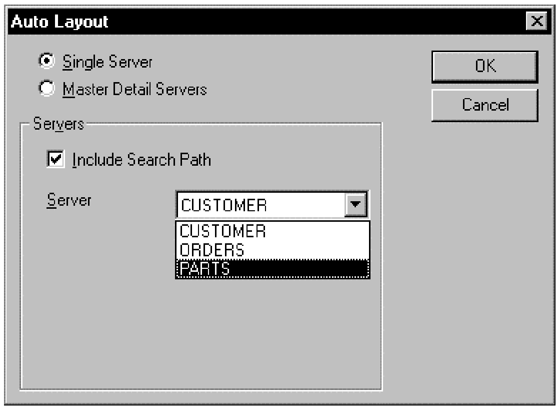
   The Auto Layout dialog box appears:



4. Optionally, check the Include Search Path check box if you want to choose from data servers that are accessible via the current application's search path, as well as those defined to it. Otherwise, leave it unchecked.

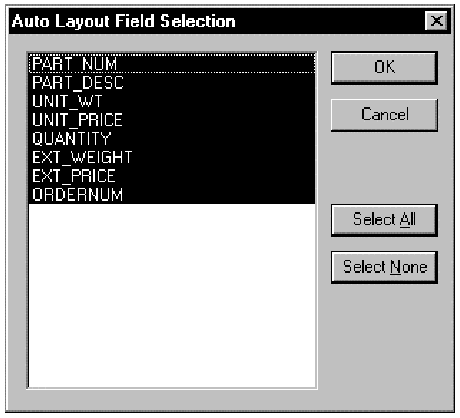
   Assuming that the Parts data server would be defined in the OE Data Servers library, like Customer and Orders, select the Include Search Path option.

5. From the Data Server list box, select the desired data server (for example, Parts):



6. Click OK.

The Auto Layout Field Selection dialog box appears:



Auto Layout Field Selection

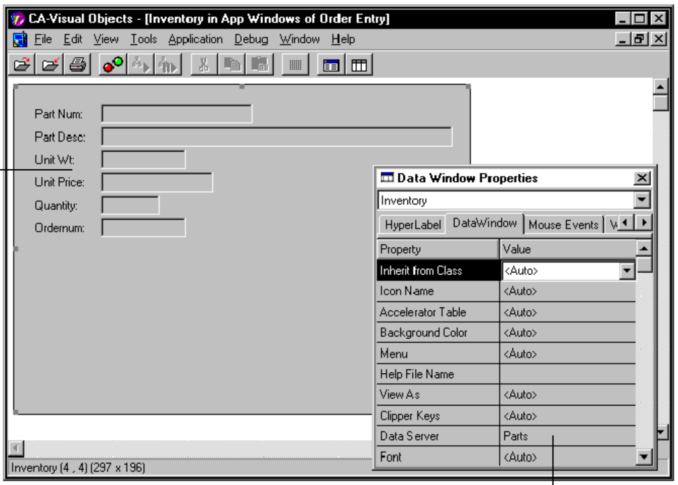
- PART_NUM
- PART_DESC
- UNIT_WT
- UNIT_PRICE
- QUANTITY
- EXT_WEIGHT
- EXT_PRICE
- ORDERNUM

7. Optionally, deselect any fields that you do not want displayed in the data window.

   For Inventory, click on EXT_WEIGHT and EXT_PRICE to exclude them, and then click OK.

   The result is a predefined form with fixed text and single-line edit controls for each field defined for the specified data server:

Controlscreatedfor eachdataserverfield



CA-Visual Objects - [Inventory in App Windows of Order Entry]

File Edit View Tools Application Debug Window Help
Part Num:
Part Desc:
Unit Wt:
Unit Price:
Quantity:
Ordernum:
Inventory (4 , 4) (297 x 196)
Data Window Properties
Inventory
HyperLabel DataWindow Mouse Events

Property	Value
Inherit from Class	<Auto>
Icon Name	<Auto>
Accelerator Table	<Auto>
Background Color	<Auto>
Menu	<Auto>
Help File Name	
View As	<Auto>
Clipper Keys	<Auto>
Data Server	Parts
Font	<Auto>

Associateddataserver

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

8. Save this form, if you wish. Otherwise, click on the Close button and choose No when prompted to save this form entity.

Tip: You can edit or delete any of the controls in a data-aware window. Additionally, you can rearrange their placement or manually add other controls using the tool palette. If you wish, you can use a multi-line edit control for certain fields; simply delete the single-line edit control, and add a multi-line edit control. See **Modifying a Window** in the "Using the Window Editor" chapter earlier in this guide for more information.

Master Detail Servers Option

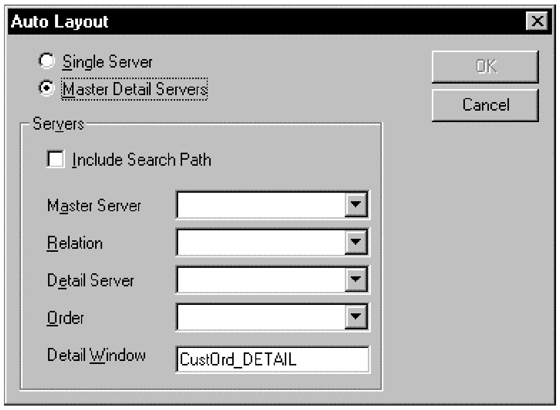
The Master Detail Servers option allows you to link a window to two data servers in a master-detail relationship. In this case, a data-aware window is created for the master server with a "nested" data window for the detail server. (See Sub-Data Windows later in this chapter for additional information about using sub-data window controls in the Window Editor for nesting purposes.)

To create a data-aware window linked to two data servers in a master-detail relationship (for example, the CustOrd form that "links" the Customer and Orders data servers in the Order Entry sample application):

1. Reopen the Window Editor from within the App Windows module, and open the CustOrd data window you created earlier.
2. Click on the Auto Layout toolbar button.

   The Auto Layout dialog box appears.
3. Select the Master Detail Servers radio button.

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



Single Server [✓ Master Detail Servers]

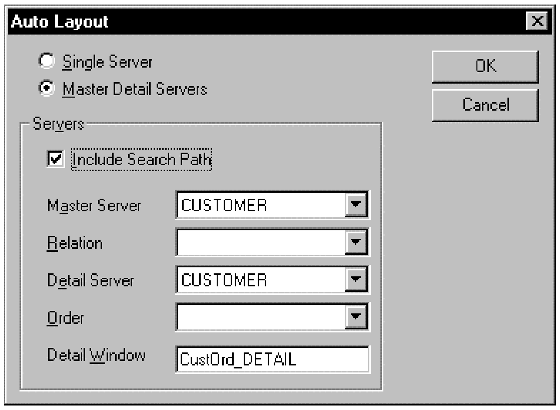
Servers

☐ Include Search Path

Master Server	
Relation	
Detail Server	
Order	
Detail Window	CustOrd_DETAIL

4. Since both of Order Entry's data servers are defined in the OE Data Servers library instead of the application itself, select the Include Search Path option.

The Customer data server appears in both the Master Server and Detail Server combo boxes:



[✓] Include Search Path

Master Server	CUSTOMER
Relation	
Detail Server	CUSTOMER
Order	
Detail Window	CustOrd_DETAIL

5. Select a common field from the Relation combo box (for example, **#CustNum**).

   The field list is taken from the master data server. The field that you choose will be used to perform a lookup operation in the detail data server's controlling order. This means that the detail server must have an open index file with a controlling order whose key is based on a similar field.

   The field name in the detail server does not have to be the same as the field name that you use for the relation string, but in some cases it will be. The important thing is that the data in the relation string field matches the data in the key field of the detail server's controlling order. (For more information about index files and orders, see Adding Index Files in **Chapter 7: Defining Data Servers** and Field Specifications.)

6. Select another data server from the Detail Server drop-down list box (for example, **Orders**).

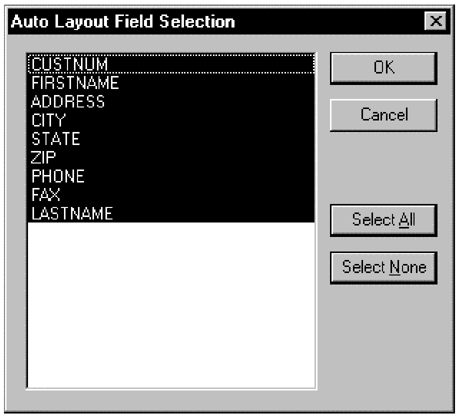
7. In the Order combo box, enter the *controlling order* for the CustOrd form (for example, **ORDCUST**). For more information about the controlling order, see Adding Index Files in **Chapter 7: Defining Data Servers** and Field Specifications earlier in this guide.

8. Note that the system automatically generates a name for the detail window and displays it in the Detail Window edit control. (This same name will be displayed in the Properties window for the sub-data window control.)

   Optionally, enter a new name for the detail form. For OrdCust, accept the default name.

9. Choose OK.

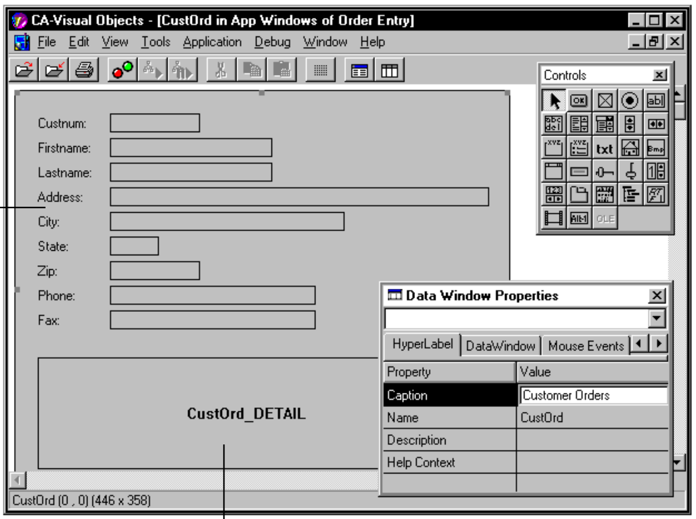
The Auto Layout Field Selection dialog box appears:



10. Optionally, deselect any fields that you do not want displayed in the data window.

    For CustOrd, simply click OK, accepting all fields.

A default CustOrd data window form is created for you, using information you already defined for the Customer and Orders data servers:


Maindatawindow
Detailwindow(sub-datawindowcontrol)
CA-Visual Objects - [CustOrd in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Controls
Custnum:
Firstname:
Lastname:
Address:
City:
State:
Zip:
Phone:
Fax:
CustOrd_DETAIL
Data Window Properties
HyperLabel DataWindow Mouse Events
CustOrd (0 , 0) (446 x 358)

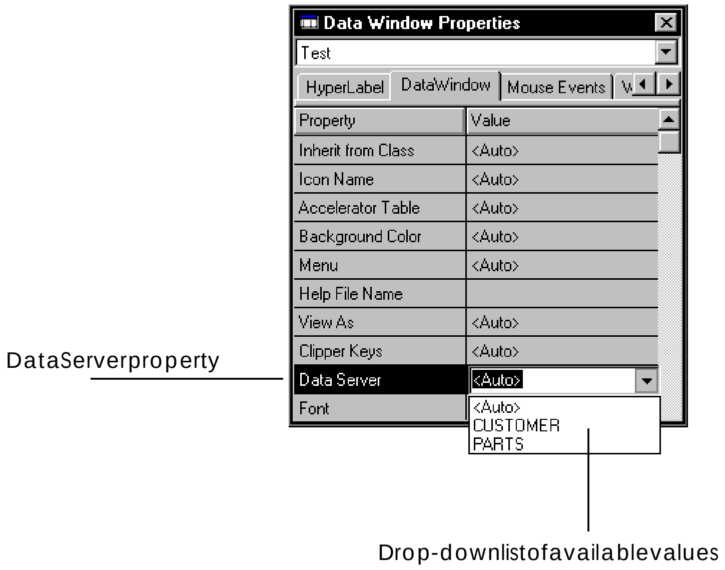
Property	Value
Caption	Customer Orders
Name	CustOrd
Description	
Help Context	

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

You can now edit the following: the data window form itself, the controls for each of the data window'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 Window** in "Using the Window Editor" for more details on editing form properties and controls. See Sub-Data Windows later in this chapter for information on how to modify the actual layout of the detail form.

Customizing a Data-Aware Window

To create a customized data window or data dialog window, start with either a blank DataWindow or DataDialog form and 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 list of all data servers defined for the current application. For example:



Property	Value
Inherit from Class	<Auto>
Icon Name	<Auto>
Accelerator Table	<Auto>
Background Color	<Auto>
Menu	<Auto>
Help File Name	
View As	<Auto>
Clipper Keys	<Auto>
Data Server	<Auto>
Font	

<Auto>
CUSTOMER
PARTS

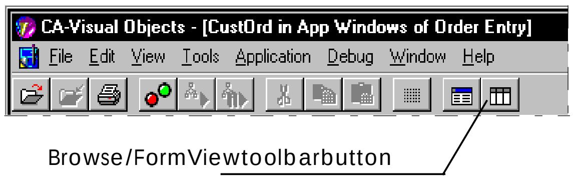
Then you can customize the blank DataWindow or DataDialog form manually, selecting controls from the tool palette or using the Select from Palette commands from the Edit menu, placing them in the desired locations on the form, and specifying properties for them, such as field specification, size, color, and caption. See **Placing Controls on a Form** in "Using the Window Editor" earlier in this guide for details.

Note that when creating a customized data-aware window, controls are linked to fields via the Field Spec and Name properties. You should specify the field names as the Name as well as choose an associated field specification.

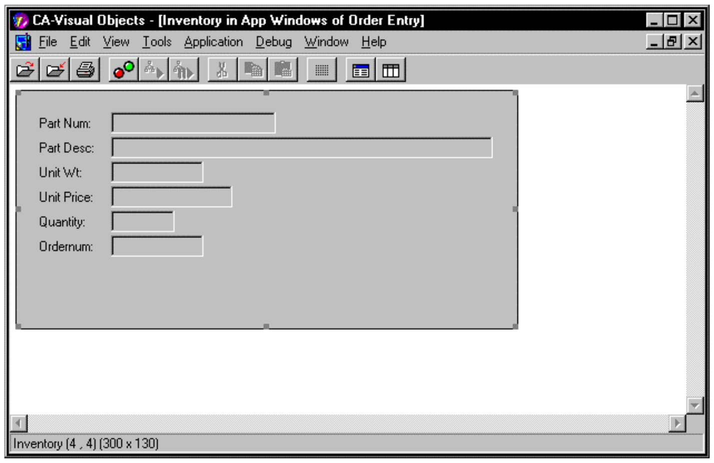
Browse and Form View

So far, you have been using the Window Editor to paint a data-aware window 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.

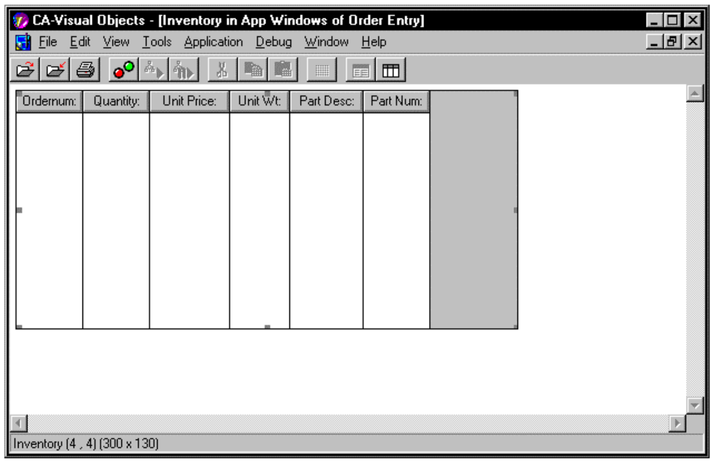
To change from one view to another, use the Browse / Form View toolbar button:


CA-Visual Objects - [CustOrd in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Browse/FormViewtoolbarbutton

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


CA-Visual Objects - [Inventory in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Part Num:
Part Desc:
Unit Wt:
Unit Price:
Quantity:
Ordernum:
Inventory (4 , 4) (300 x 130)

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


CA-Visual Objects - [Inventory in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Ordernum: Quantity: Unit Price: Unit Wt: Part Desc: Part Num:
Inventory (4 , 4) (300 x 130)

Note: Changing the view associated with a data-aware window also changes the form's View As property, as described earlier under Specifying Window Properties.

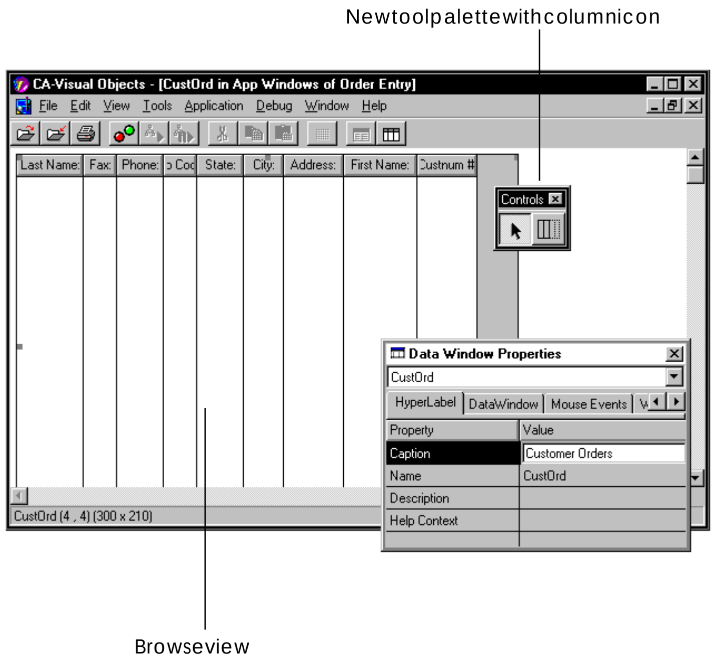
Tip: You can start out by designing the form in browse view, if you like. Similar to the way in which the Window Editor creates a default browse view for forms designed using form view, it will create a default form view for forms designed using browse view. Note, however, that Auto Layout is available only in form view.

Working in Browse View

If you are planning to give the end user the option to switch between views at runtime or if you are going to use browse view as the default, you will probably want to switch to browse view and make any modifications that are necessary.

For example, our sample CustOrd data window form was created initially using Customer as the master server and Orders as the detail server (see Master Detail Option in the Using Auto Layout section earlier in this chapter). To change the data window form so that it displays in browse view, click the Browse/Form View toolbar button.

Note the new tool palette with the column icon that allows you to place additional columns in the table:


Newtoolpalettewithcolumnicon
CA-Visual Objects - [CustOrd in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Last Name: Fax: Phone: Zip Code State: City: Address: First Name: Custnum #
Controls
CustOrd (4 , 4) (300 x 210)
Data Window Properties
CustOrd
HyperLabel DataWindow Mouse Events
Property Value
Caption Customer Orders
Name CustOrd
Description
Help Context
Browseview

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.

   You will see a vertical bar with left and right arrows 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.

Tip: If necessary, move the Properties window out of the way by clicking on its title bar and dragging it to a new position.

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

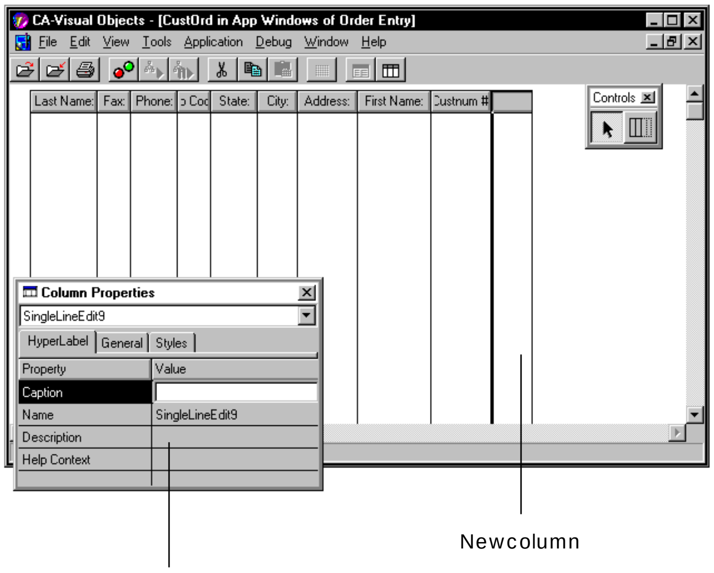
If you like, you can use the Cut, Copy, and Paste toolbar buttons to manipulate columns using the Clipboard. When you paste a column, it is inserted as the rightmost column in the table. Then you can move the column to its desired position.

Inserting Columns

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

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

   A column corresponding to a single-line edit control is added, and the appropriate Properties window appears:


CA-Visual Objects - [CustOrd in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Last Name: Fax: Phone: Zip Cod State: City: Address: First Name: Custnum #
Controls
Column Properties
HyperLabel General Styles
Newcolumn
ColumnPropertieswindow

SingleLineEdit9

Property	Value
Caption	
Name	SingleLineEdit9
Description	
Help Context	

2. Specify the values for the new column.

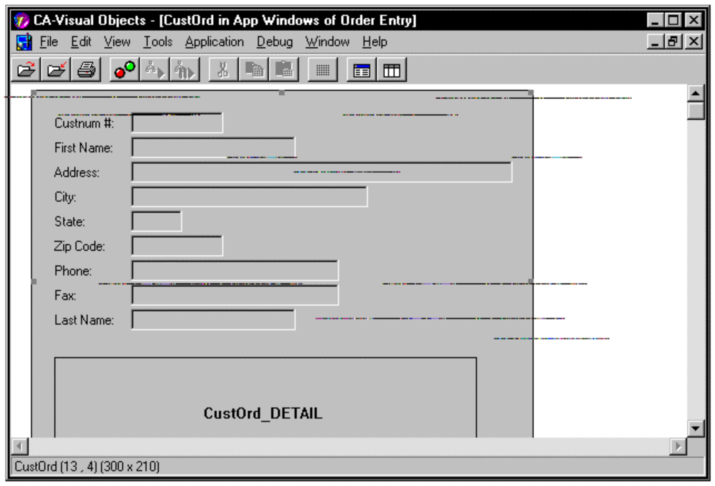
   See **Specifying Control Properties and Style** Settings in “Using the Window Editor” earlier in this guide for more details.

Tip: The browse view also allows runtime configuration by the user; that is, column widths may be adjusted by dragging on the border, and columns may be rearranged by simply dragging them to a new place.

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

Form View

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


CA-Visual Objects - [CustOrd in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Custnum #:
First Name:
Address:
City:
State:
Zip Code:
Phone:
Fax:
Last Name:
CustOrd_DETAIL
CustOrd (13 , 4) (300 x 210)

Sub-Data Windows

A *sub-data window* is nothing more than a specialized name to refer to a data window that is a control on another data window (or data dialog window). It is the technique that you will use in the Window Editor for nesting one data window within another. There is no difference between a sub-data window and a data window, other than the fact that a sub-data window defaults to browse view and that it can be manipulated as a control on a data window.

Creating a Sub-Data Window Using Auto Layout

One way to create a sub-data window is to select the Master Detail option of the Auto Layout feature discussed earlier. This gives you a typical window layout in which the master record is displayed in form view with its related detail records in browse view. See Master Detail Option in the Using Auto Layout section for detailed information about linking a window to two data servers at the touch of a button.

Manually Creating a Sub-Data Window

Instead of using the Auto Layout feature, however, you can create a data window and manually place one or more sub-data windows within it, using the sub-data window icon on the tool palette.

Using this technique, you can either insert an existing data window as a sub-data window control or create a new sub-data window that you customize. After placing the control on the data window form, you then specify its Name property. To use an existing data window as your sub-data window control, simply type its window name in the Properties window. To create a new sub-data window, specify a new name.

Tip: You can manually convert an existing single server data form to one that is linked to two data servers in a master-detail relationship using this same technique.

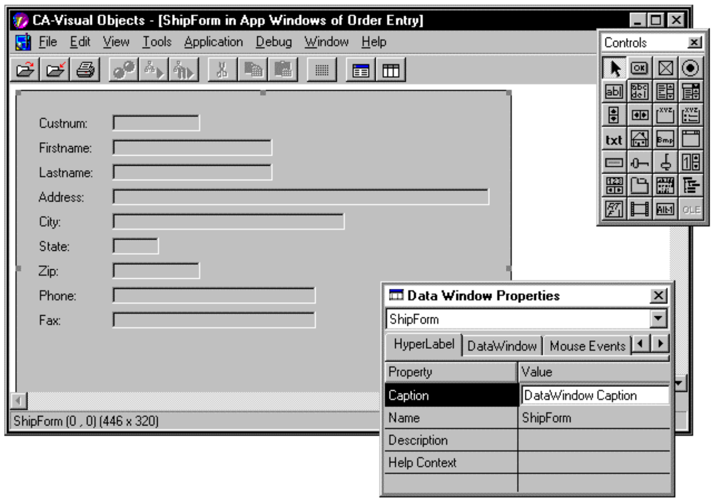
The relationship between the sub-data window and its owner data window is specified using the Relation String property, discussed previously in this chapter. (For more information about the Relation String property, see **Specifying Control Properties and Style** Settings in "Using the Window Editor" earlier in this guide.)

Note: If the data window that you are using as a sub-data window is larger than the owner window, the sub-data window will be clipped to fit within the owner window. This is a useful feature since the data window defining the sub-data window will be displayed full size when used alone.

Example 1: Using an Existing Data Window

For example, if you want to create a new data window for shipping data and insert an *existing* data window with customer information as a sub-data window control, you would follow these basic steps:

1. Create a new single server data window form (for example, **ShipForm**), linking it to an existing data server (for example, **Customer**):


CA-Visual Objects - [ShipForm in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Controls
Custnum:
Firstname:
Lastname:
Address:
City:
State:
Zip:
Phone:
Fax:
ShipForm (0 , 0) (446 x 320)
Data Window Properties
HyperLabel DataWindow Mouse Events

ShipForm

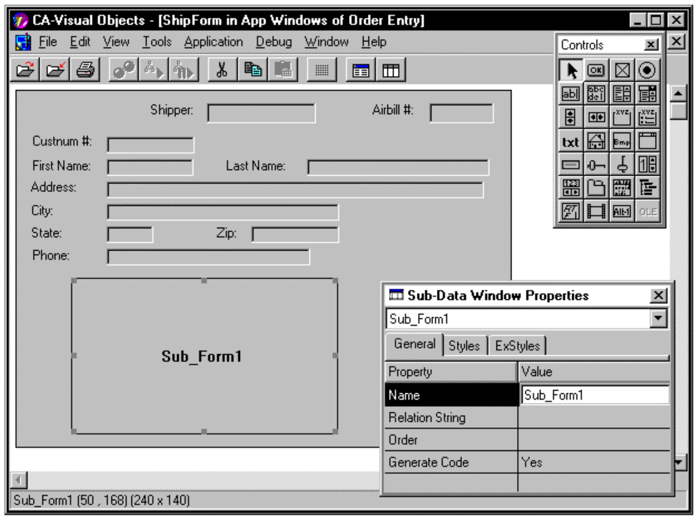
Property	Value
Caption	DataWindow Caption
Name	ShipForm
Description	
Help Context	

2. Optionally change the Caption property to **Shipping Form**.

3. Rearrange these fields as you like, perhaps adding new fields for the shipper and airbill number.

4. Click on the tool palette's sub-data window icon, and place the control onto the ShipForm data window form.

   A sub-data window with the default name of Sub_Form1 appears along with the appropriate Properties window:

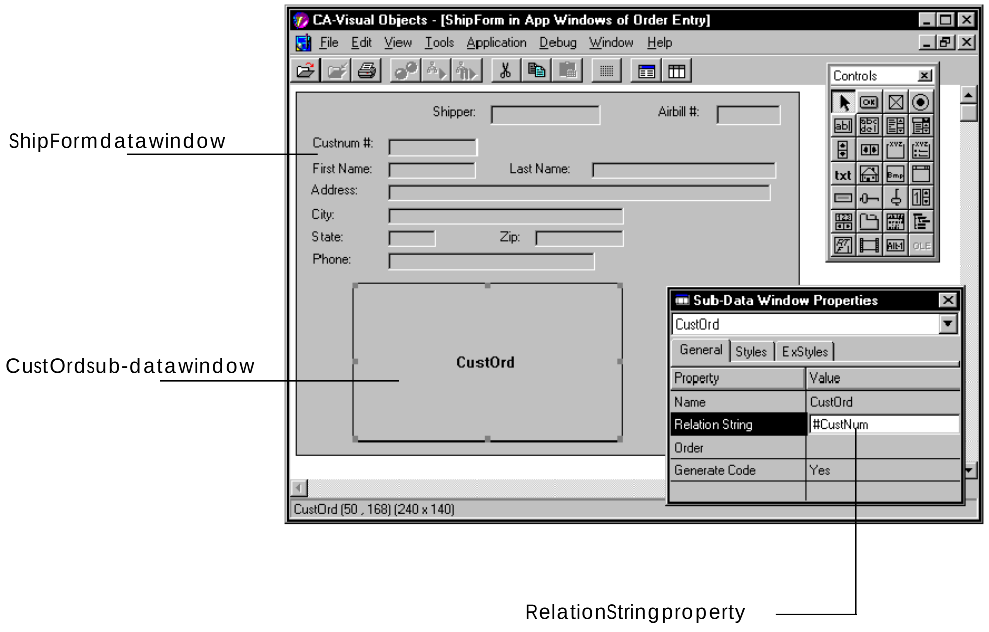

CA-Visual Objects - [ShipForm in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Controls
Shipper:
Airbill #:
Custnum #:
First Name:
Last Name:
Address:
City:
State:
Zip:
Phone:
Sub_Form1
Sub_Form1 (50 , 168) (240 x 140)
Sub-Data Window Properties
General Styles ExStyles

Sub_Form1

Property	Value
Name	Sub_Form1
Relation String	
Order	
Generate Code	Yes

5. Replace the default sub-data window name, Sub_Form1, with the name of the existing data window (for example, **CustOrd**).
6. In the Relation String value cell, enter a field that is common to both the data window's data server and the sub-data window's data server (for example, #CustNum).

   Your ShipForm data window form should now look something like this:



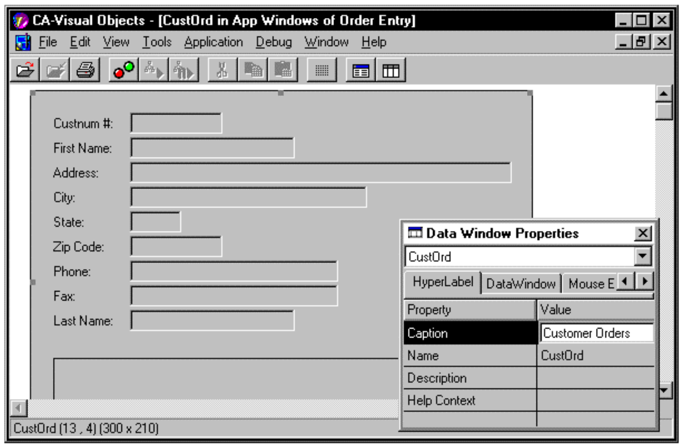
CustOrd

Property	Value
Name	CustOrd
Relation String	#CustNum
Order	
Generate Code	Yes

Note: As mentioned earlier in this chapter, this common field must be used to define the controlling order for the sub-data window's server.

7. Choose the Save toolbar button to save the form.
8. Double-click on the sub-data window control to view the nested data window's complete properties.

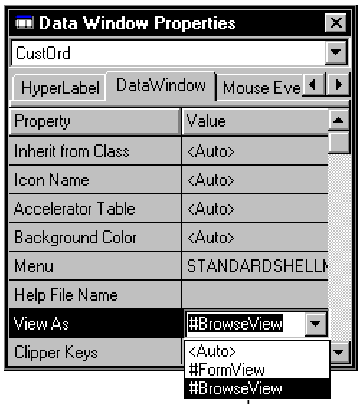
A new copy of the Window Editor appears, displaying the sub-data window, CustOrd, in its default view, which in this case is *form view*:


CA-Visual Objects - [CustOrd in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Custnum #:
First Name:
Address:
City:
State:
Zip Code:
Phone:
Fax:
Last Name:
Data Window Properties
CustOrd
HyperLabel DataWindow Mouse E
CustOrd (13 , 4) (300 x 210)

Property	Value
Caption	Customer Orders
Name	CustOrd
Description	
Help Context	

Because Customer was specified previously as the data server for the sample CustOrd data window form, which is now being used as a sub-data window, its fields are displayed here.

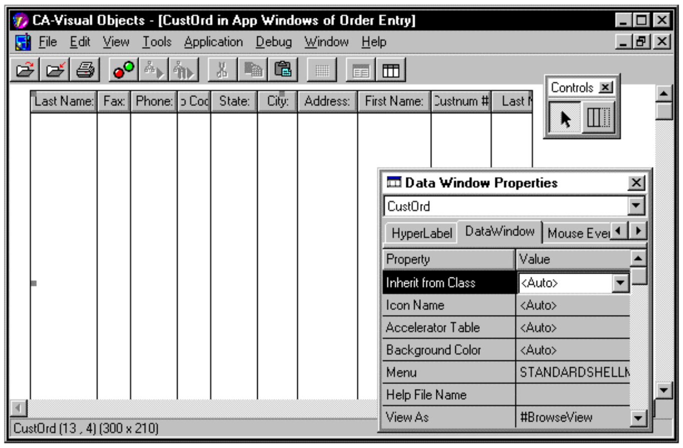
9. Switch to the DataWindow tab in the Data Window Properties window, and change the View As property to #BrowseView:



Property	Value
Inherit from Class	<Auto>
Icon Name	<Auto>
Accelerator Table	<Auto>
Background Color	<Auto>
Menu	STANDARDSHELLI
Help File Name	
View As	#BrowseView
Clipper Keys	

Select#BrowseView

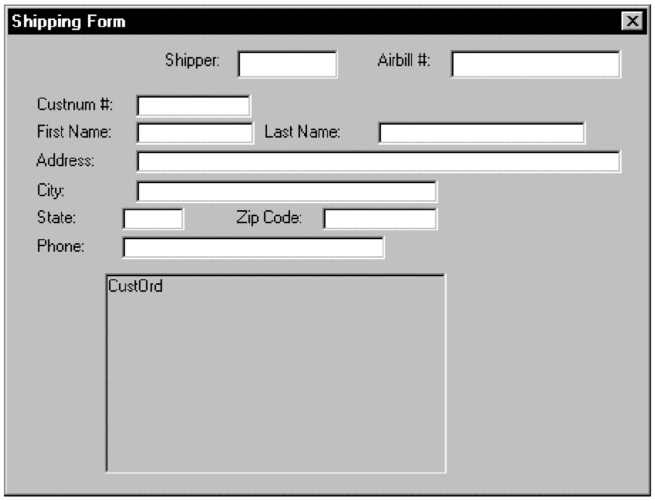
The Window Editor should now look as follows:


CA-Visual Objects - [CustOrd in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Last Name: Fax: Phone: Zip Code State: City: Address: First Name: Custnum #: Last Name
Controls
Data Window Properties
CustOrd
HyperLabel DataWindow Mouse Events
CustOrd (13 , 4) (300 x 210)

Property	Value
Inherit from Class	<Auto>
Icon Name	<Auto>
Accelerator Table	<Auto>
Background Color	<Auto>
Menu	STANDARDSHELLM
Help File Name	
View As	#BrowseView

10. Optionally, modify this sub-data window.
11. Close the sub-data window by double-clicking on its system menu.

    If you have made any changes since you last saved this form, you will be prompted to save them before returning to the Window Editor session for the main data window form, ShipForm.
12. Before going on to the next example, choose the Test Mode command from the View menu (or press Ctrl+T) to view the ShipForm data window form as the end user will see it:



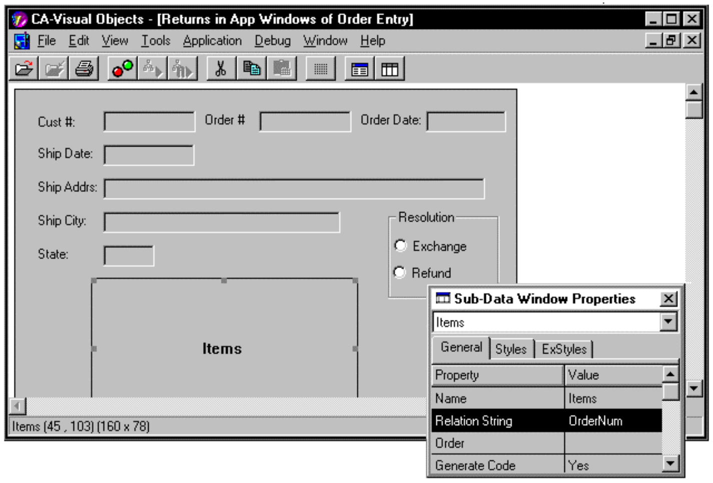
Example 2: Creating a New Sub-Data Window

If, however, you want to create a *new* sub-data window control for a single-server data window form that links, for example, returned items to customers in the Order Entry sample application, you would follow these basic steps:

1. Create a new single-server data window form (for example, **Returns**), linking it to an existing data server (for example, **Customer**).
2. Optionally change the Caption property to **Returns**.
3. Use the Auto Layout feature and the Auto Layout Field Selection dialog box to place relevant fields on the main data window form.
4. Rearrange these fields as you like, perhaps adding new fields such as a radio button group, **Resolution**, and two radio buttons, **Exchange** and **Refund**.
5. Click on the tool palette's sub-data window icon, and place the control onto the data window form.

   A sub-data window with the default name of Sub_Form1 appears along with the appropriate Properties window.
6. Optionally, rename the sub-data window form (for example, **Items**).
7. In the Relation String value cell, enter a field that is common to both the data window's data server and the sub-data window's data server (for example, **#OrderNum**).

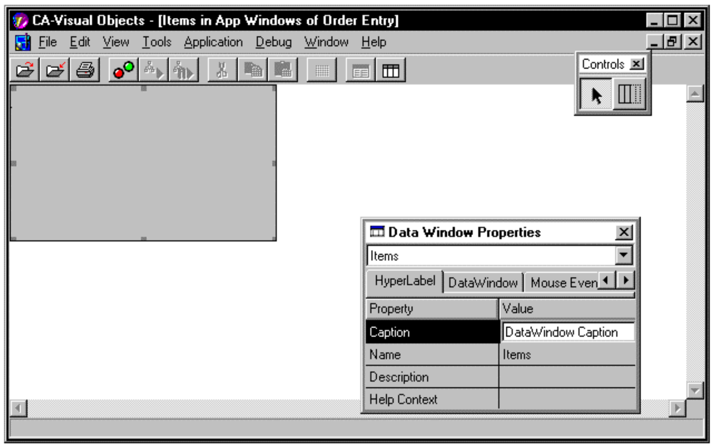
   The Returns data window form should look something like this:


CA-Visual Objects - [Returns in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Cust #:
Order #
Order Date:
Ship Date:
Ship Addrs:
Ship City:
State:
Resolution
Exchange
Refund
Items
Items (45 , 103) (160 x 78)
Sub-Data Window Properties
Items
General Styles ExStyles

Property	Value
Name	Items
Relation String	OrderNum
Order	
Generate Code	Yes

8. Choose the Save toolbar button to save the form.
9. Double-click on the sub-data window control.

A new copy of the Window Editor appears, displaying the Items sub-data window form in browse view:


CA-Visual Objects - [Items in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Controls
Data Window Properties
Items
HyperLabel DataWindow Mouse Even

Property	Value
Caption	DataWindow Caption
Name	Items
Description	
Help Context	

Note: The previous example of a sub-data window used an existing data window as the sub-data window control and, consequently, inherited its fields from that data window's data server. Since this form has not yet inherited any fields, the form is blank.

10. Optionally, change the Caption property to **Items**.

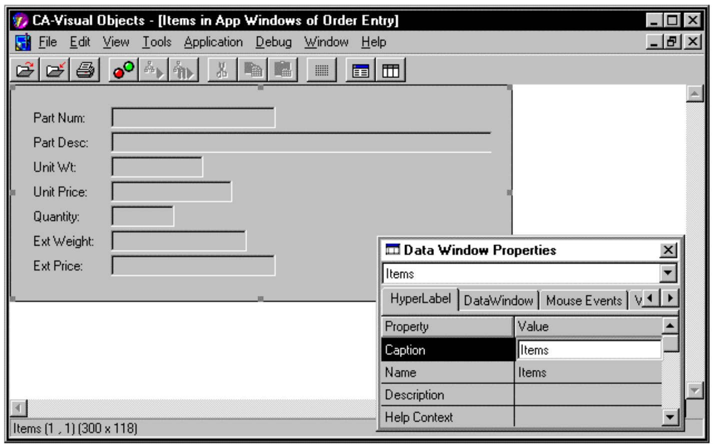
Now, you need to associate the Returns sub-data window with a data server, and the easiest way to do this is using the Auto Layout feature, as described below:

1. Change to form view using the Browse/Form View toolbar button, if necessary.
2. Click the Auto Layout toolbar button.

   The Auto Layout dialog box appears.
3. Choose the Single Server option.
4. Select the Include Search Path option.
5. From the Servers drop-down list box, select the data server for the sub-data window form (for example, **Parts**).
6. Click OK.

   The Auto Layout Field Selection dialog box appears.
7. Select relevant fields, such as PartNum and Quantity, for the sub-data window form.
8. Click OK.

The sub-data window form, Items, in form view now looks something like this:

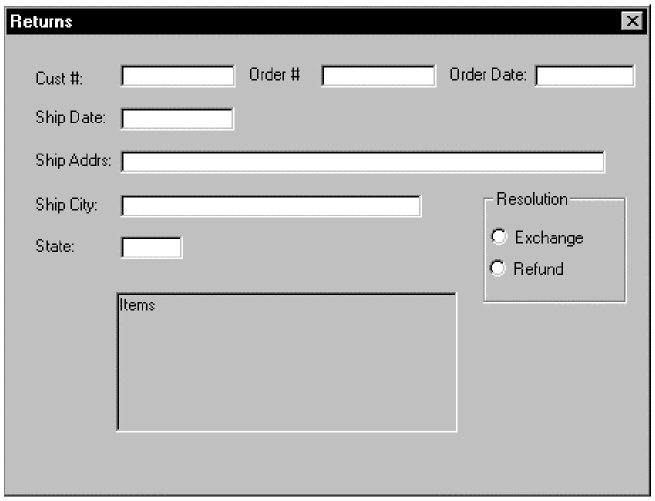

CA-Visual Objects - [Items in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
Part Num:
Part Desc:
Unit Wt:
Unit Price:
Quantity:
Ext Weight:
Ext Price:
Data Window Properties
Items
HyperLabel DataWindow Mouse Events
Items (1 , 1) (300 x 118)

Property	Value
Caption	Items
Name	Items
Description	
Help Context	

9. Optionally modify any of the other properties for the sub-data window form.
10. Switch to the DataWindow tab in the Data Window Properties window, and change the View As property to **#BrowseView**.
11. Click on the Save toolbar button to save the new sub-data window form.
12. Close the sub-data window form by double-clicking on its system menu to return to the Returns data window form.
13. Choose the Test Mode command from the View menu.

    Alternatively, press Ctrl+T.

    The Returns data window form appears as the end user will see it:



14. Close the Window Editor.

Modifying a Sub-Data Window

Modifying a sub-data window form is the same as modifying the layout of any other data-aware window. You can customize the sub-data window form in browse view as described earlier under Working in Browse View, or you can work with it in form view.

You can also change any of the sub-data window properties. When it is displayed as a control, a sub-data window has limited properties. These include its name and its relationship to the master data server, as described earlier in the Specifying Window Properties section in “Using the Window Editor.”

Tip: Remember, you can gain access to a sub-data window form’s *full* set of properties by double-clicking on its sub-data window control. A new copy of the Window Editor will be invoked, allowing you to edit any of the properties displayed.

Chapter

9

Using the Report Editor

The CA-Visual Objects Report Editor is a state-of-the-art, database publishing tool that allows you to create sophisticated reports. It consists of the CA-Report Writer and the CA-Report Viewer.

The CA-Report Writer's WYSIWYG environment allows you to design and produce custom database reports at the press of a button directly from within the IDE. You can mix text, graphics, and data using a wide variety of publishing features. Use or customize a predefined report style or design your own report from scratch. Group related data together and perform mathematical calculations, such as group and report totals. Then preview your report online using the CA-Report Viewer to see how it looks and make any necessary modifications before printing it.

Once a report has been designed and saved, CA-Visual Objects automatically generates object-oriented code for it, using the appropriate classes, and adds it to the repository. When executing the application, you can preview the report using the Report Editor's runtime engine. With a minor source code modification, you can also print the report.

Tip: Since the Report Editor includes a royalty-free runtime version, you can ship this report engine with any of your CA-Visual Objects applications. See the "Operating Environment" chapter in the *Programmer's Guide* for more information about preparing your application for delivery.

This chapter describes how to use the Report Editor. In it you will learn how to:

- Access the CA-Report Writer window and set its properties
- Use queries in a report
- Create reports, including working with report sections and tables, formatting the report, determining page layout, and adding graphics
- Using report fields
- Preview reports online using the CA-Report Viewer
- Print reports and export reports to files

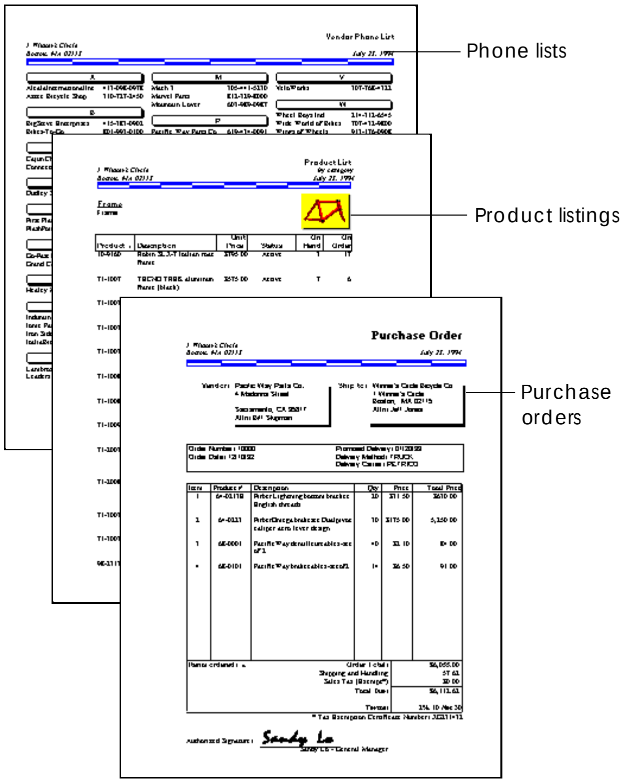
Report Styles and Definitions

Report Styles

The CA-Report Writer's predefined report styles make it easy to create reports. When you choose a style, the CA-Report Writer creates a report definition for you that you can use to enhance a report. The following is a list of the types of report styles from which you can choose:

Style	Description
Tabular	Produces columnar-style reports that display the data in multiple rows and columns; suitable for sales and inventory reports, for example.
Form	Produces continuous reports that display one record of data at a time, with each field individually labeled. Form reports are useful when reporting on a large number of data fields that would not easily fit into columns; suitable for personnel reports and customer statements, for example.
Labels	Produces labels, such as mailing labels, inventory tags, name badges.
Letter	Produces form letters.
Free style	Creates an empty report form so that you can build a customized report from scratch; suitable for a variety of customized reports, such as aged trial balances, money market portfolios, catalogs, or customized invoices.
Cross Tabular	Creates cross tabular reports that display report data in rows and columns in which the data values are aggregated for each row and column pair; suitable for sales and inventory control reports, for example.

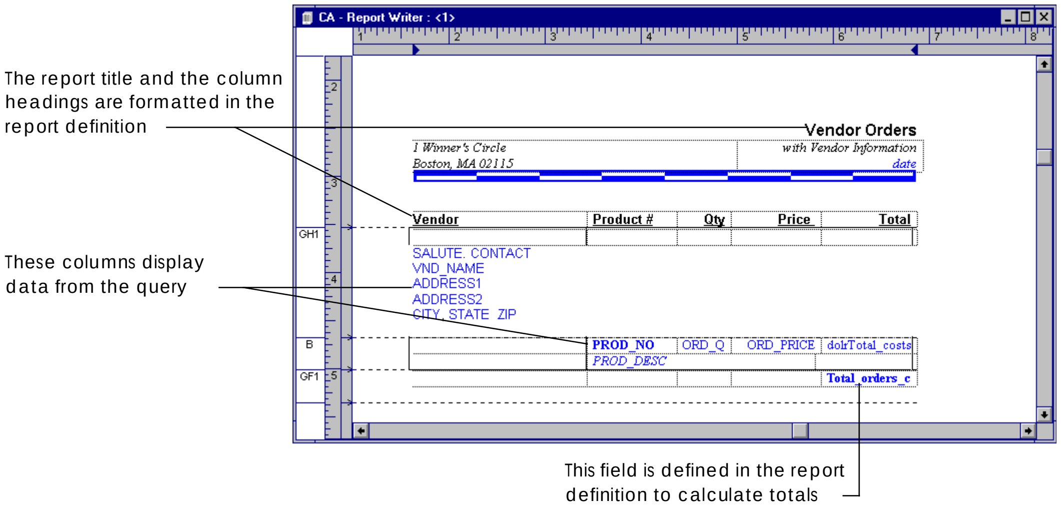
The following examples illustrate several report styles:



To see sample reports using each of these styles and learn how you can customize them, see Customizing Report Definitions later in this chapter.

Report Definitions

The information to format a report comes from a query and from the report's design. This information is called the *report definition*. The following illustration shows a report definition that is comprised of the row and column titles of a query:



Important! *When you view the CA-Report Writer window, you are viewing the report definition—**not** the data associated with the report. You can view report data in print preview.*

Note: Columns from a query are referred to as *fields* in a report definition.

Workspace Overview

The CA-Report Writer window contains its own menu bar, toolbar, and group categories for formatting the style of the body of a report. The window also contains rulers for controlling page margins and section formatting, and a status bar for displaying cursor information.

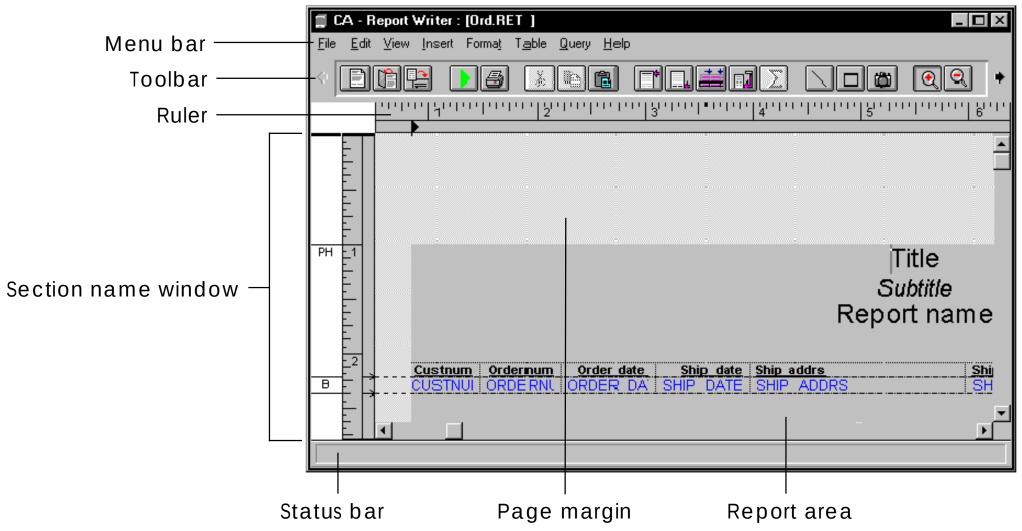
When you create or open a report, the report definition is displayed in the CA-Report Writer window. The report definition is composed of the row and column titles from the table in the query in the format of the selected style. The data from the query is also attached to the report definition, and is available for you to view in Print Preview.

When you create a new report, you are prompted to:

- Connect to a data source
- Select a report style

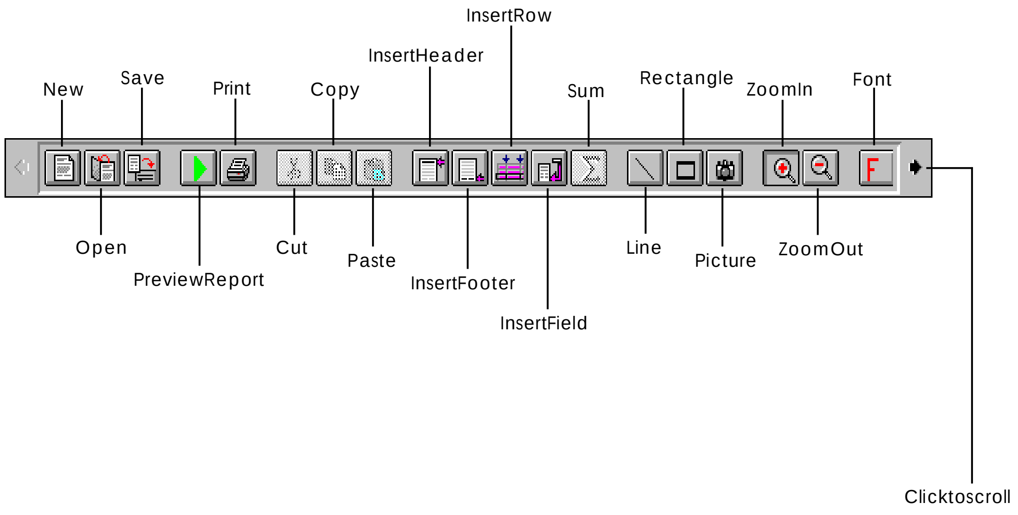
When you create or open a report, the report definition is associated with a query. You can display or print the query data using the information in the report definition. You can enhance the report by adding text or graphics to the report definition in the CA-Report Writer window.

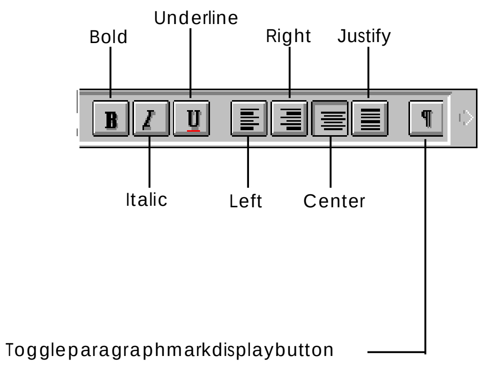
The following illustration shows the various features of the CA-Report Writer window:



The Toolbar

The CA-Report Writer toolbar contains buttons for frequently used menu commands:





The Rulers

The CA-Report Writer horizontal and vertical rulers help you format the report page. Use the horizontal ruler to select and size columns and the vertical ruler to format sections.

To hide rulers, use the Rulers option in the Report Defaults dialog box. For more information about showing and hiding rulers, see Setting Report Properties later in this chapter.

Creating a Report

Now that you have a general overview of the Report Editor workspace, you are ready to use the CA-Report Writer to create or open a report and the CA-Report Viewer to preview it.

Creating a New Report

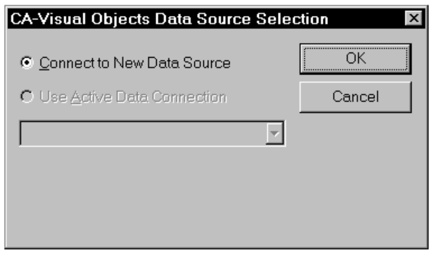
Basic Steps

To create a new report, you need to perform the following steps:

1. Start the Report Editor.

   Like all CA-Visual Objects tools, the Report Editor is accessed using the Tools menu or the New Entity toolbar button.

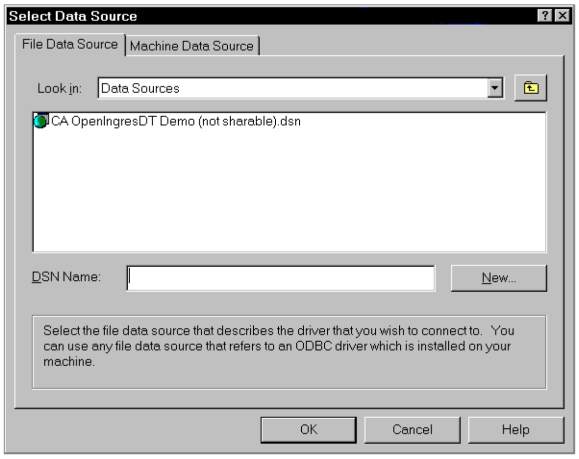
The CA-Visual Objects Data Source Selection dialog box appears:


CA-Visual Objects Data Source Selection
Connect to New Data Source
Use Active Data Connection
OK
Cancel

This dialog box allows you to connect to the database containing the tables to be associated with the report.

2. Select a data source for the report doing one of the following:

   ■ Select the Connect to New Data Source option to access the Select Data Source dialog box:



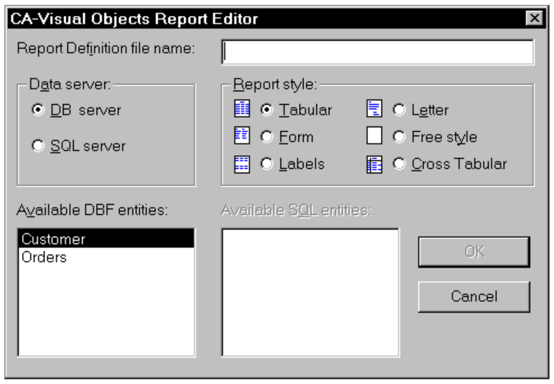
Select the file data source that describes the driver that you wish to connect to. You can use any file data source that refers to an ODBC driver which is installed on your machine.

   ■ Select the Use Active Data Connection option to access the Active Data Connection combo box.

3. From the Machine Data Source tab page of the Select Data Source dialog box or from the Active Data Connection combo box, select an SQL data source (for example, dBASE Files).

4. Choose OK.

The CA-Visual Objects Report Editor dialog box appears:



5. Enter a file name for the report (for example, **Salesord** or **Ord**) in the Report Definition File Name edit control.
6. From the Data Server group box, choose the type of data server to be associated with the report.
7. Select one or more data servers from either the Available DBF Entities or the Available SQL Entities list box.

   Note: If you choose more than one data server, you will be prompted for information on how to join them. Refer to your online help system for more information on how to complete the Report Editor Joins dialog box.

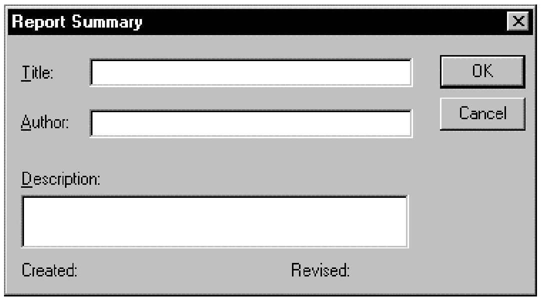
   An SQL query is automatically generated, and the data server's fields are added automatically to the report definition.
8. Choose a report style from the Report Style group box.

   Note: If you select the Cross Tabular option, you are presented with another dialog box, Crosstab Report Definition, to define the report definition. For information about this dialog box, see Cross Tabular Reports later in this chapter.

   The CA-Report Writer creates a report definition based on the report style you selected and the data associated with the SQL query.
9. Click OK to start the CA-Report Writer.
10. Format the predefined report template (name, page layout, headings, and so on) in the CA-Report Writer.

    Naturally, you can customize a basic report by adding details like literal, database, and computed fields; spreadsheet-like functions; text; and graphics. You can also add graphics using CA-Report Writer's drawing features. Typeface, color, and size can be selected for your report text, and you can add bolding, italics, and other print features.
11. Before saving your report, you can optionally select the Report Summary command from the Edit menu.

The Report Summary dialog box appears:



Enter information about the report and its author in the appropriate edit controls.

12. Click OK.

    You are returned to the CA-Report Writer.

13. Save the report.

    At this point, CA-Report Writer saves the report definition as an external .RET file.

14. Close the CA-Report Writer.

You will be prompted to store the changed entities. If you choose Yes, a binary entity and a subclass for the report are saved to the Repository Explorer. If No, the .RET file remains but no entities are added to the Repository Explorer.

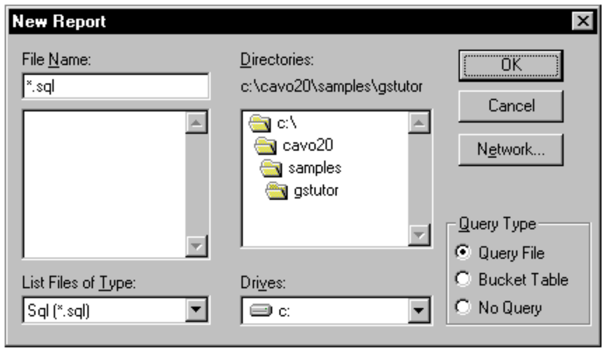
Note: To create another report definition without quitting the current definition, repeat the steps in the preceding procedure, which starts another CA-Report Writer session. You can define only one report per CA-Report Writer session, but you can run multiple sessions simultaneously.

Alternative Method

You can also create a new report from within the CA-Report Writer itself. To do so:

1. Click the CA-Report Writer's New toolbar button.

   The New Report dialog box appears:



File Name: *.sql

Directories: c:\cavo20\samples\gstutor

c:\
cavo20
samples
gstutor

List Files of Type: Sql (*.sql)

Drives: c:

Query Type: [✓ Query File] Bucket Table No Query

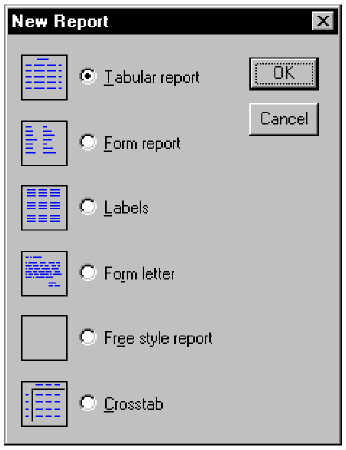
2. Specify the .SQL file and its path name and folder.

3. Select a query type using the Query Type group box.

   Note: You cannot create a query file in CA-Visual Objects; however, you can import both .SQL and .QBE files. See Working with Queries in Reports later in this chapter for more detailed information.

4. Click OK.

   The New Report dialog box appears:



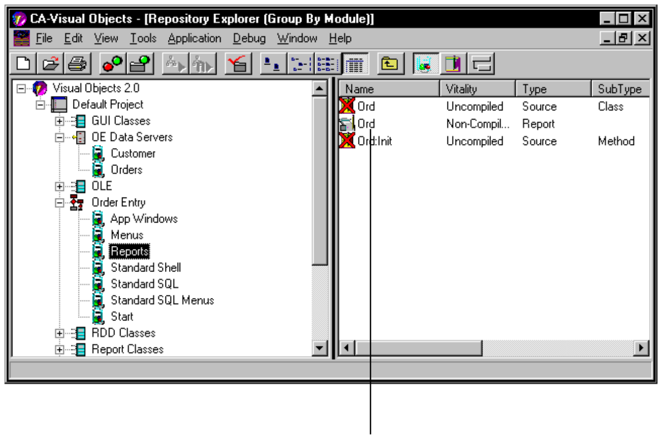
[✓ Tabular report]
Form report
Labels
Form letter
Free style report
Crosstab

5. Select a report style for the new report definition, and then click OK.
6. Format the predefined report template (name, page layout, headings, and so on).
7. Save the report.

Opening a Report

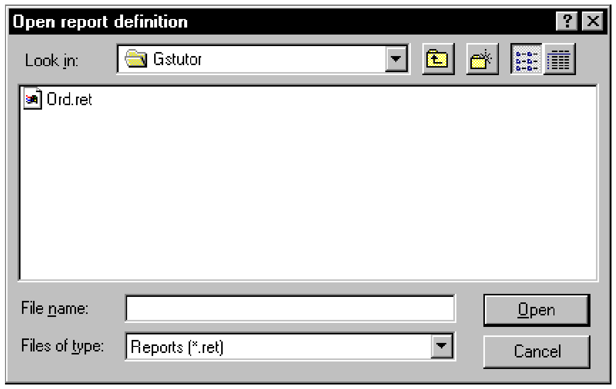
To open an existing report, do one of the following:

- Double-click on the specified report entity in the Repository's list view pane. For example:



Name	Vitality	Type	SubType
Ord	Uncompiled	Source	Class
Ord	Non-Compil...	Report	
Ord:Init	Uncompiled	Source	Method

- From within the CA-Report Writer, click on the Open toolbar button and select an .RET file from the Open Report Definition dialog box that appears:



Open report definition

Look in: Gstutor

Ord.ret

File name:

Files of type: Reports (*.ret)

In both cases, the specified report is loaded immediately in a CA-Report Writer window.

Note: CA-RET 1.0 reports will also require an active database connection. Before opening a CA-RET 1.0 report, you must create a data source for the report database and connect to it.

Generating Code

When you save a report—responding Yes to the prompt to store entity changes—CA-Visual Objects creates the following:

- A report entity

  You can double-click on this entity in the Repository Explorer's list view pane to begin editing it with the CA-Report Writer.

- A ReportQueue subclass, using the report name

  It is the Init( ) method of this subclass that determines the action to take when the end user chooses to run the report. By default, it displays a preview of the report. Refer to Printing a Report later in this chapter and to the ReportQueue class in the online help system for detailed information on how to customize this code.

- An external .RET file

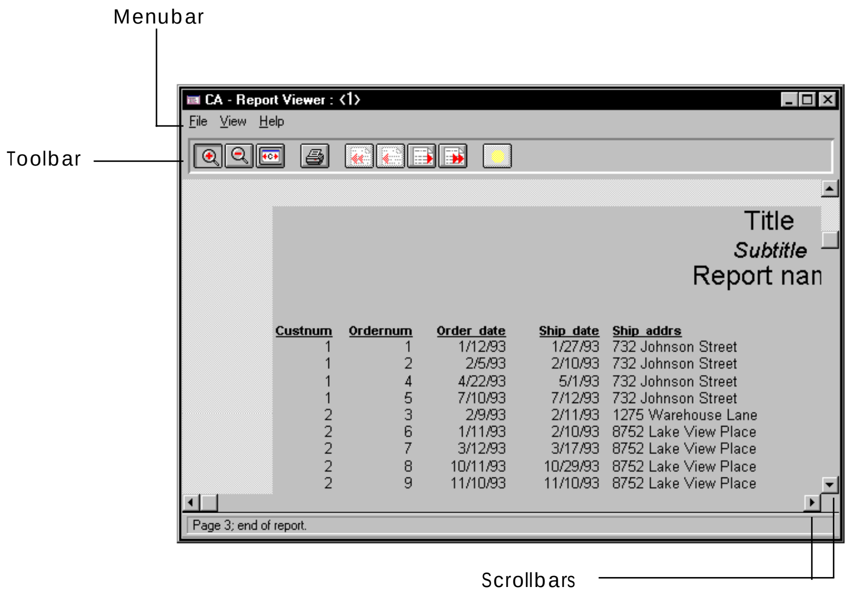
Viewing Report Data

When you create a new report or open an existing one, the CA-Report Writer displays the report definition. You can view the actual data via the CA-Report Viewer when you use the Print Preview option.

To preview a report, do one of the following in the CA-Report Writer window:

- Click the Preview Report toolbar button.
- Choose the Print Preview command from the File menu.

The CA-Report Viewer appears. For example:



Title
Subtitle
Report nan

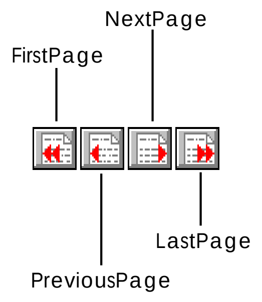
Custnum	Ordernum	Order_date	Ship_date	Ship_addrs
1	1	1/12/93	1/27/93	732 Johnson Street
1	2	2/5/93	2/10/93	732 Johnson Street
1	4	4/22/93	5/1/93	732 Johnson Street
1	5	7/10/93	7/12/93	732 Johnson Street
2	3	2/9/93	2/11/93	1275 Warehouse Lane
2	6	1/11/93	2/10/93	8752 Lake View Place
2	7	3/12/93	3/17/93	8752 Lake View Place
2	8	10/11/93	10/29/93	8752 Lake View Place
2	9	11/10/93	11/10/93	8752 Lake View Place

Page 3; end of report.

Notice that the CA-Report Viewer has its own menu bar, toolbar, and scroll bars, so navigating within this window is easy. You can view different pages of the report or you can print the report just by clicking on various toolbar buttons.

Moving to Another Page

You can use the following toolbar buttons to view different pages of the report:

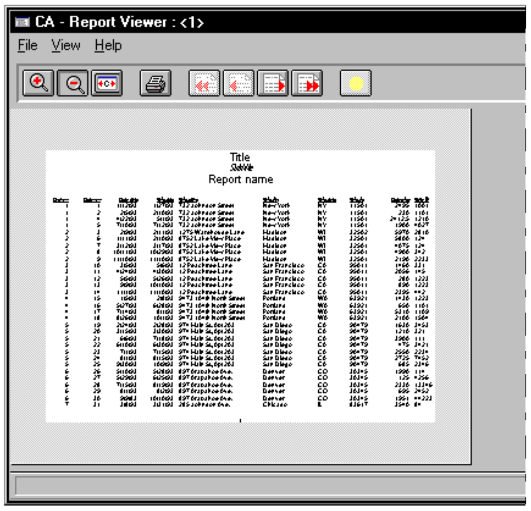


Viewing Entire Page

To view an entire report page:

1. Click the Page View toolbar button.

   The report page is reduced in its entirety. For example:



2. To return to normal view, click the Normal View toolbar button.

Centering a Report

To center the current report in the Preview window, click the Center toolbar button.

Returning to the Report Definition

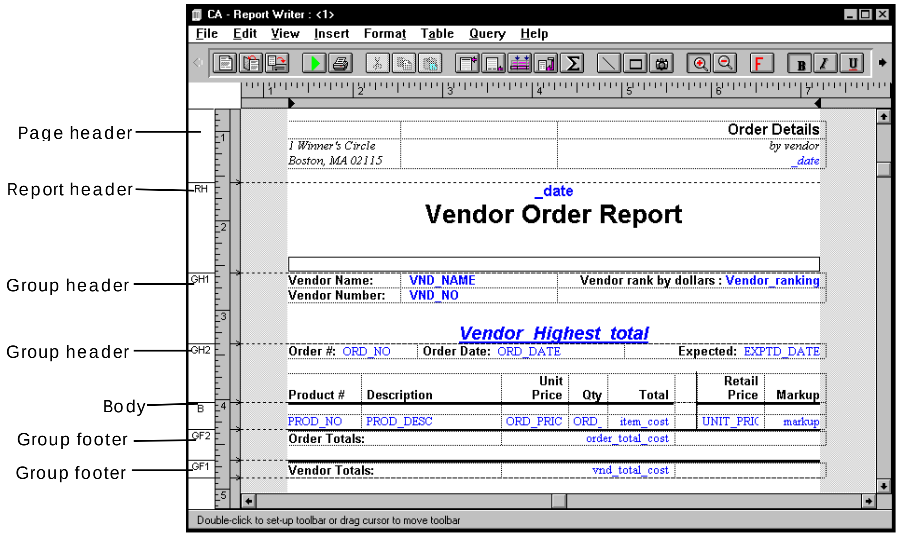
To return to the report definition, do one of the following:

- If a report is one page long, click the Close toolbar button.
- If a report is longer than one page, click the Cancel toolbar button and then click Close.

Customizing Report Definitions

As stated earlier, you can customize your report definition by selecting various report styles that are described in more detail in the following sections. For instructions about making these design changes, see Designing Reports later in this chapter.

The following illustration shows the various sections that can be included in a report definition:



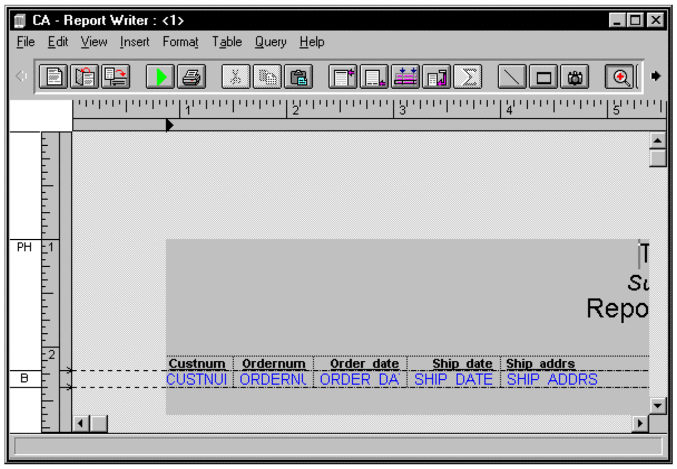
Note that the available sections will vary depending on the report style selected.

Tabular Reports

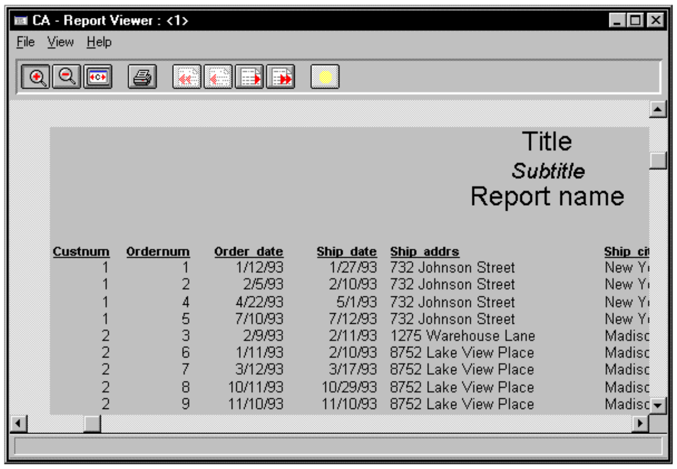
When you select the tabular report style, the CA-Report Writer creates a report definition comprised of the following:

- A page header with sample title and subtitle
- A body with one or more rows containing data from each column in the query with a corresponding column header
- A page footer containing the page number

Below is an example of a tabular report definition:


CA - Report Writer : <1>
File Edit View Insert Format Table Query Help
PH
B
T
Su
Repo
Custnum Ordernum Order_date Ship_date Ship_addrs
CUSTNUI ORDERNU ORDER_DA SHIP_DATE SHIP_ADDRS

Here is its data displayed in the CA-Report Viewer:



Title

Subtitle

Report name

Custnum	Ordernum	Order_date	Ship_date	Ship_addrs	Ship_ci
1	1	1/12/93	1/27/93	732 Johnson Street	New Y
1	2	2/5/93	2/10/93	732 Johnson Street	New Y
1	4	4/22/93	5/1/93	732 Johnson Street	New Y
1	5	7/10/93	7/12/93	732 Johnson Street	New Y
2	3	2/9/93	2/11/93	1275 Warehouse Lane	Madisc
2	6	1/11/93	2/10/93	8752 Lake View Place	Madisc
2	7	3/12/93	3/17/93	8752 Lake View Place	Madisc
2	8	10/11/93	10/29/93	8752 Lake View Place	Madisc
2	9	11/10/93	11/10/93	8752 Lake View Place	Madisc

You can change the generic page title and column headings of a tabular report, as well as insert group headers and footers to break the report into groups by which data can be categorized or summarized. For example, you can group all sales statistics by office or region or all parts by inventory category. You can also do the following:

- Delete columns of unwanted data
- Rearrange or change the width of columns
- Modify the format strings that control the appearance of data fields
- Add fields that are calculated from other data in the report
- Enhance the report appearance using different fonts, colors, and styles, graphics images, lines, boxes, and borders

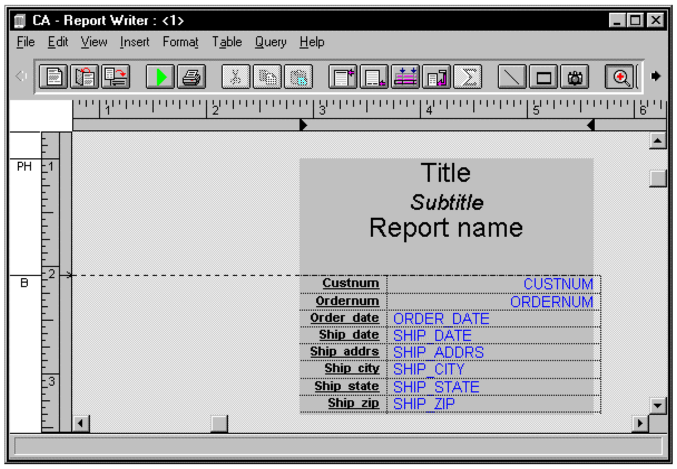
To perform these tasks, see Working with Tables, Working with Rows, and Using Report Fields later in this chapter.

Form Reports

When you select the form report style, the CA-Report Writer creates a report definition comprised of the following:

- A page header section
- A page footer with page numbers
- A body section that lists a label and a value for each field

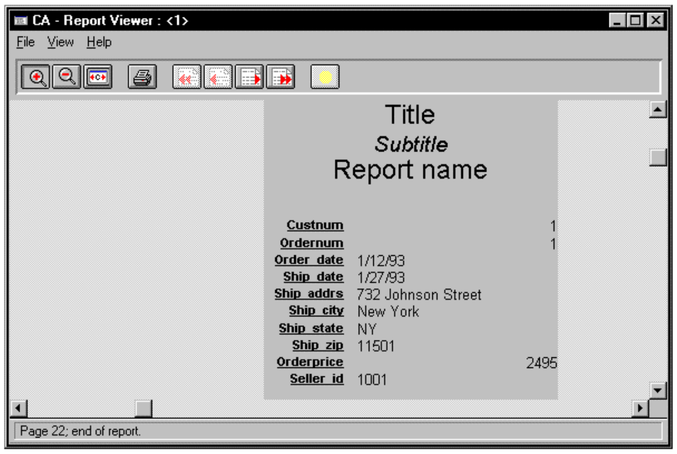
Below is an example of a form report definition:



Title
Subtitle
Report name

Custnum	CUSTNUM
Ordernum	ORDERNUM
Order_date	ORDER_DATE
Ship_date	SHIP_DATE
Ship_addrs	SHIP_ADDRS
Ship_city	SHIP_CITY
Ship_state	SHIP_STATE
Ship_zip	SHIP_ZIP

Here is its data displayed in the CA-Report Viewer:



Title
Subtitle
Report name

Custnum	1
Ordernum	1
Order_date	1/12/93
Ship_date	1/27/93
Ship_addrs	732 Johnson Street
Ship_city	New York
Ship_state	NY
Ship_zip	11501
Orderprice	2495
Seller_id	1001

Page 22; end of report.

You can change the generic report title and field labels of form reports, as well as insert group headers and footers to break the report into groups; for example, all employees by office or job title. You can also:

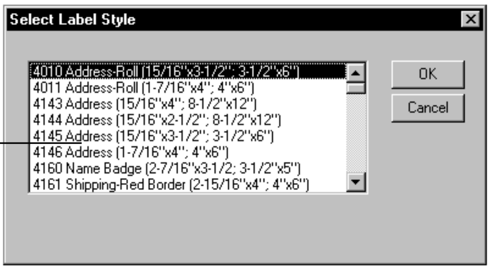
- Add fields to and delete fields from the report
- Rearrange the order of data fields
- Change the alignment and appearance of text and data
- Place each individual record on a separate page
- Add additional columns
- Add a single-page report summary to accompany a multi-page report

To perform these tasks, see Working with Tables, Working with Rows, and Using Report Fields later in this chapter.

Label Reports

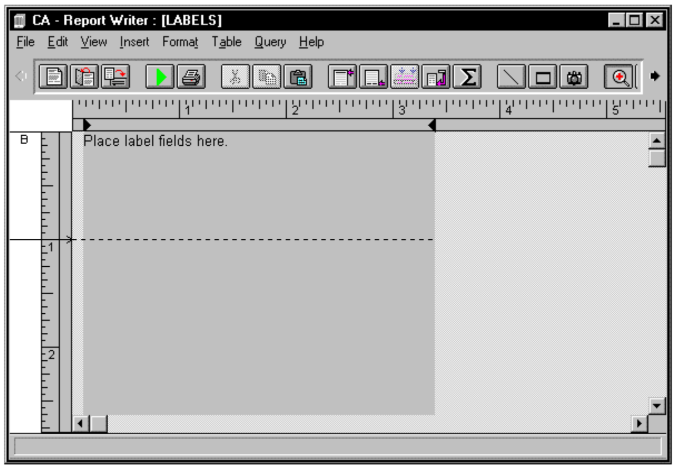
When you select the labels style, the CA-Report Writer creates a report definition with a body section, then displays the Select Label Style dialog box from which you choose a standard Avery label size or other label form:

Numberscorrespondtoproduct numbersonAverylabelboxes

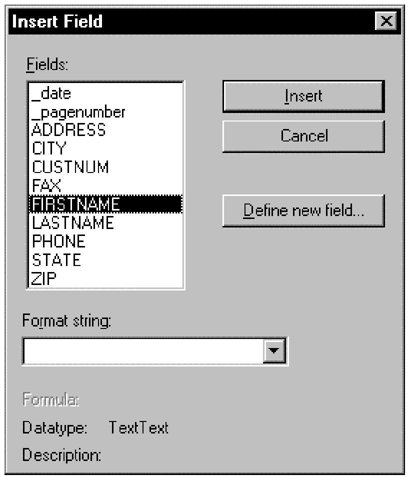


4010 Address-Roll (15/16"x3-1/2"; 3-1/2"x6")
4011 Address-Roll (1-7/16"x4"; 4"x6")
4143 Address (15/16"x4"; 8-1/2"x12")
4144 Address (15/16"x2-1/2"; 8-1/2"x12")
4145 Address (15/16"x3-1/2"; 3-1/2"x6")
4146 Address (1-7/16"x4"; 4"x6")
4160 Name Badge (2-7/16"x3-1/2; 3-1/2"x5")
4161 Shipping-Red Border (2-15/16"x4"; 4"x6")

When you click OK, a labels report definition displays:



To select the fields that you want to include on the labels, choose the Field command from the Insert menu. The Insert Field dialog box appears:



Fields:
_date
_pagenumber
ADDRESS
CITY
CUSTNUM
FAX
FIRSTNAME
LASTNAME
PHONE
STATE
ZIP

Format string:

Formula:

Datatype: TextText

Description:

After selecting fields such as FIRSTNAME, LASTNAME, ADDRESS, STATE, and ZIP, the report will print your labels. For example:

Report prints labels across and down the page

Note: See Using Report Fields for detailed information about the Insert Field dialog box.

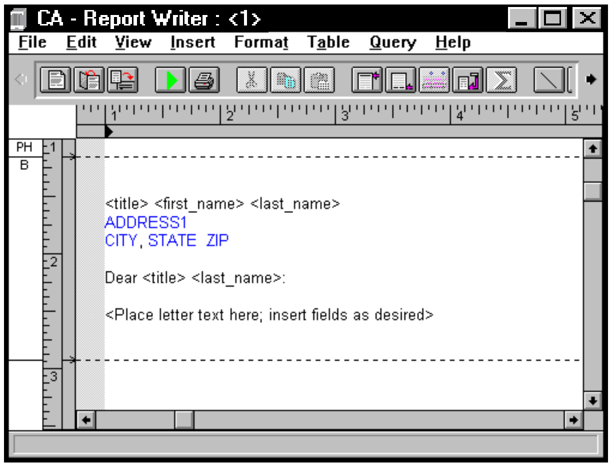
The CA-Report Writer sets the page size and margins for cut sheet (Avery 5000-series) labels, so that printing begins 1/10 in. from the left margin and 1/16 in. from the top of each label.

For continuous form labels (Avery 4000-series) used with tractor-feed printers, the CA-Report Writer sets the left margin at 1/10 in., but with no top margin. You should align the form in the printer so the top line prints properly. For continuous form labels, you should set the page size as specified in Printer Setup to match the size of one sheet (one fold) of labels.

Form Letter Reports

When you select the form letter style, the CA-Report Writer creates a report definition with an empty page header section and a body section for the text of the letter. It uses the query columns to build the body. If they match exactly, they are placed in the body; otherwise, they are placed in prompts (indicated by < >), which the CA-Report Writer uses to suggest field positions. You need to replace these with columns from the query.

Each form letter prints on a new page. Below is an example of a form letter report definition:

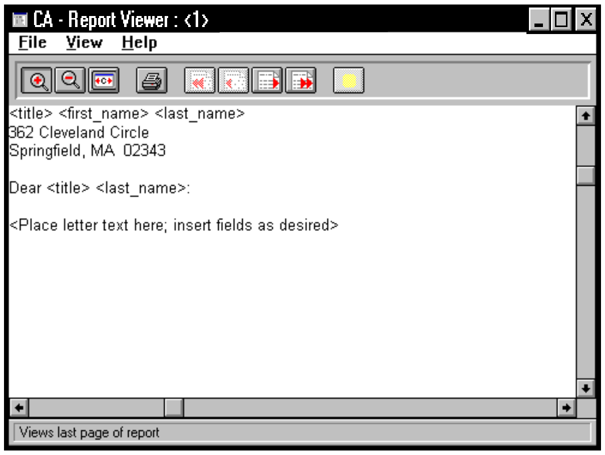


<title> <first_name> <last_name>
ADDRESS1
CITY, STATE ZIP

Dear <title> <last_name>:

<Place letter text here; insert fields as desired>

Here is its data displayed in the CA-Report Viewer:



<title> <first_name> <last_name>
362 Cleveland Circle
Springfield, MA 02343

Dear <title> <last_name>:

<Place letter text here; insert fields as desired>

Views last page of report

You can make the following changes to customize a form letter:

- Insert name and address fields at the beginning of the body section
- Insert an appropriate salutation
- Enter the text of the letter
- Place database fields within the text of your letter and / or create a table of data fields to include data from your database in the body of your form letter

To perform these tasks, see Working with Tables, Working with Rows, and Using Report Fields later in this chapter.

Free Style Reports

When you select the free style report style, the CA-Report Writer creates a report definition with an empty body section. It does not add any other report sections or features, and it does not place any data fields or columns on the report definition.

You add sections and other desired features to create a report definition.

Cross Tabular Reports

Cross tabular reports display report data in rows and columns. Data values are aggregated for each row and column pair. This report style is useful for inventory control, manufacturing control, and sales performance reports.

You could create a report to produce the same results as the preceding report by creating two group breaks, one on product and the other on date. The cross tabular report presents the same results in a compact, easy-to-read table.

Creating a Cross Tabular Report

To create a cross tabular report:

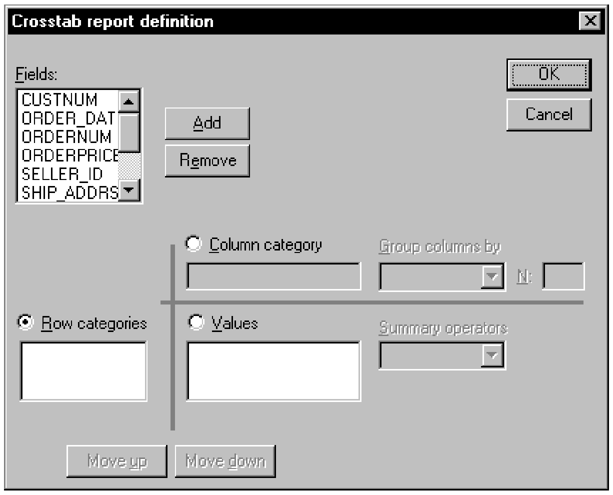
1. Access the Report Editor, connect to a data source, and then select or create a query.

   The CA-VO Report Editor dialog box appears.

   See Creating a Report earlier in this chapter for details.

2. Select the Crosstab report style option.

   The Crosstab Report Definition dialog box appears:



Note that the Row Categories option is preselected by default.

3. Select at least one field from the Fields list box to use as a row in the report, and click Add.

   If you select more than one field, the first field is a level 1 group header, the next is level 2, and so on. Click Move Up and Move Down to change the order of the fields.

4. Select the Column Category option.

5. Select one field from the Fields list box to use as a column category in the report, and click Add.

   You can use the Group Columns By list box to define how you want the column grouped. The grouping you can select depends on the data type of the column category field.

6. Select the Values option and select one or more fields from the Fields list box to specify the fields you want summed.

   If you do not specify a value, the CA-Report Writer totals the values of the column category field. If it is a numeric field, the CA-Report Writer uses the Sum function; otherwise, it uses the Count function. You can select a different function from the Summary Operators list box.

   If you specify more than one field, you can define each as an adjacent crosstab column. See Putting Columns of Like Data Together in later in this chapter for more detailed information.

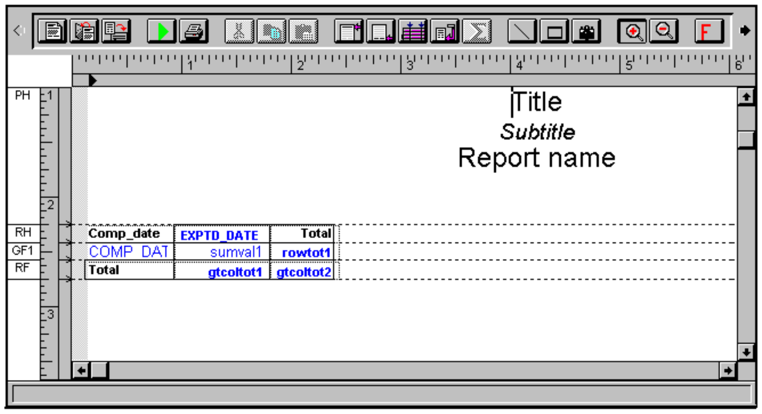
7. Click OK.

The CA-Report Writer creates a report definition with:

- A page header section
- A report header section listing the column headers, the column category field, and a total header for the row totals

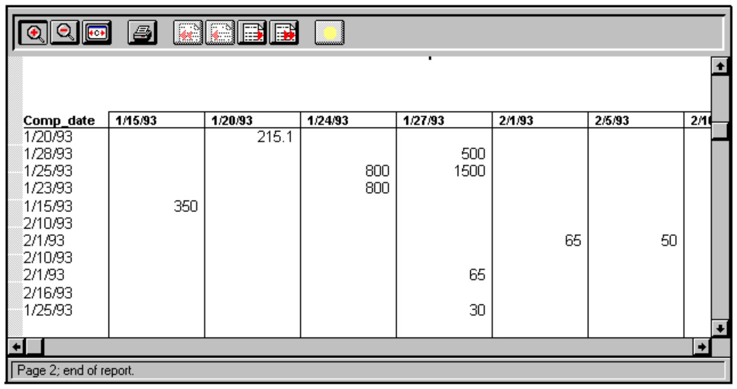
- A group footer section that contains the report data and row totals, and a report footer section that contains the column totals

Below is an example of a cross tabular report definition:



Comp_date	EXPTD_DATE	Total
COMP_DAT	sumval1	rowtot1
Total	gtcoltot1	gtcoltot2

Here is its data displayed in the CA-Report Viewer:



Comp_date	1/15/93	1/20/93	1/24/93	1/27/93	2/1/93	2/5/93	2/1
1/20/93		215.1					
1/28/93				500			
1/25/93			800	1500			
1/23/93			800				
1/15/93	350						
2/10/93							
2/1/93					65	50	
2/10/93							
2/1/93				65			
2/16/93							
1/25/93				30			

Page 2; end of report.

To allow column headings to be displayed on reports longer than one page, the CA-Report Writer turns on the Repeat Section On Page Break option in the Format Section dialog box for the report header.

You should change the generic page titles, and you will probably want to change the column headers as well. You can also change the format of the column category, date fields, and totals.

Changing the Column Category for the Report

To change a column category for your report:

1. Choose the Crosstab Definition command from the Edit menu.

   The Crosstab Report Definition dialog box appears again.

2. Select the specified field in the Column Category list box, and then click the Remove button.
3. Select a new field from the Fields list box and click the Add button.

   Note: Make sure you edit the field definitions of any aggregate fields that reference the column.

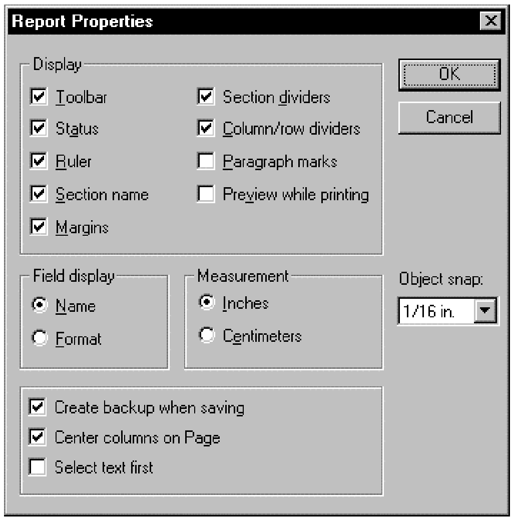
Setting Report Properties

You can set your own preferences for creating and editing report definitions. For example, you can hide or display margins and rulers, you can choose between inches and centimeters for measuring, and you can choose to create a backup of the report whenever you save it.

To set your report's properties:

1. Start the CA-Report Writer by opening an existing report or creating a new report (as described earlier in this chapter).
2. Choose the Properties command from the File menu.

   The Report Properties dialog box appears:



Display
- [✓] Toolbar
- [✓] Status
- [✓] Ruler
- [✓] Section name
- [✓] Margins
- [✓] Section dividers
- [✓] Column/row dividers
- [ ] Paragraph marks
- [ ] Preview while printing

Field display: [✓ Name] Format

Measurement: [✓ Inches] Centimeters

Object snap: 1/16 in.

- [✓] Create backup when saving
- [✓] Center columns on Page
- [ ] Select text first

3. Select the report properties you want to use.
4. Click OK.

Following are descriptions of the available options by category for the Report Properties dialog box.

Display Options

Use the Display group box options to control the display of various components in the CA-Report Writer window.

Field Display Options

Use Field Display group box options to control whether a field in the report definition appears as a field name or as a format string, such as decimal placement or number display.

Measurement Options

Use the Measurement group box options to choose inches or centimeters to use on rulers in the CA-Report Writer window and in the object snap feature.

Object Snap Option

Use the Object Snap option to specify the density of the snap grid. CA-Report Writer uses the grid to align objects you move in the report definition.

Other Options

Miscellaneous options appear at the bottom of the Report Properties dialog box as described below:

- Select Create Backup when Saving to back up the report definition
- Select Center Columns on Page to center table columns on the page
- Choose Select Text First to be able to select text

  Deselect this option for graphics.

Working with Queries in Reports

You can associate more than one query with a report definition, but you can use only one query at a time, however, as the main report query. You can work with the query in the CA-Report Writer window and perform the following tasks:

- Associate multiple queries with a report definition
- Rename a query
- Edit a query
- Export a query to a query window

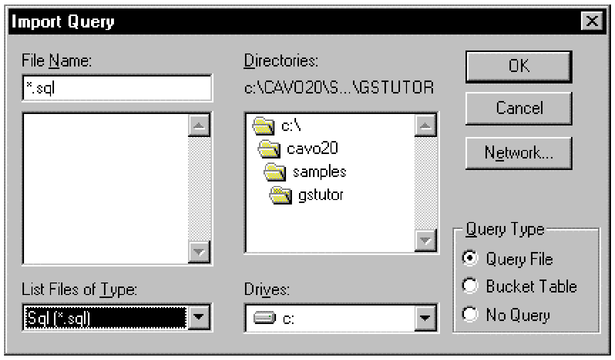
Associating Multiple Queries with a Report Definition

If you want to be able to create different reports from a single report definition, you can associate multiple queries with it. For example, you can create a report definition that has one query that accesses dBASE files, another query that accesses Btrieve files, and a third query that retrieves all records from a text file. By selecting which query to use before you print the report, you get different reports from the same report definition.

To associate multiple queries with a report definition:

1. Open or create a report definition as described earlier in this chapter.
2. Choose the Import Query command from the Query menu.

The Import Query dialog box appears:



3. Select a query from the Import Query dialog box, and then click OK.
4. If any of the column names in the new query do not match those in the report definition, map the fields in order to run the report successfully.

   For details, see Mapping Columns from a Query to a Report later in this section.

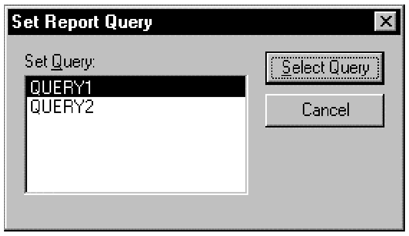
Selecting a Query to Use with a Report

You can use only one query at a time with a report definition as the main report query. If a report has multiple queries defined, you need to specify which query to use. A report continues using the same query until you set a different one.

To specify which query to use:

1. Open or create a report definition as described earlier in this chapter.
2. Choose the Set Report Query command from the Query menu.

   The Set Report Query dialog box appears:



3. Select the query you want to use, and then click the Select Query button.
4. If any of the column names in the new query do not match those in the report definition, map the fields in order to run the report successfully.

   For details, see Mapping Columns from a Query to a Report later in this section.

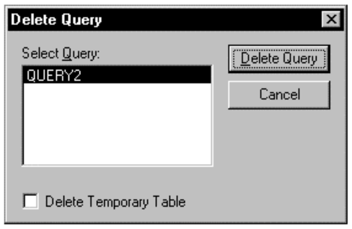
Deleting a Query

If you have multiple queries associated with a report definition, you can delete all but one of them. You must have at least one query associated with a report.

To delete a query:

1. In the CA-Report Writer window, choose the Delete Query command from the Query menu.

   The Delete Query dialog box appears:

2. Select the query you want to delete from the list box.
3. Optionally select Delete Temporary Table.

   This option, if selected, deletes the temporary table created when you create a report from the Report Browser window. See the CA-Report Writer online help system for detailed information.

4. Click the Delete Query button.

Mapping Columns from a Query to a Report

Mapping controls the connection between columns from a report query and fields in a report definition. Mapping lets you change the database column to which a report field refers. You may need to do this when you have multiple queries associated with a report and the column names in each of the queries are different.

Suppressing Fields

You can map fields when you want to suppress them from printing. For example, you could define an additional query for a report that inhibits certain internal or sensitive information, producing a non-proprietary version of the same report. You would eliminate the sensitive fields from the second query and match the corresponding report definition fields to a null value, or replace the database column with a computed field having a value such as “Information not available.”

Mapping when Adding New Queries

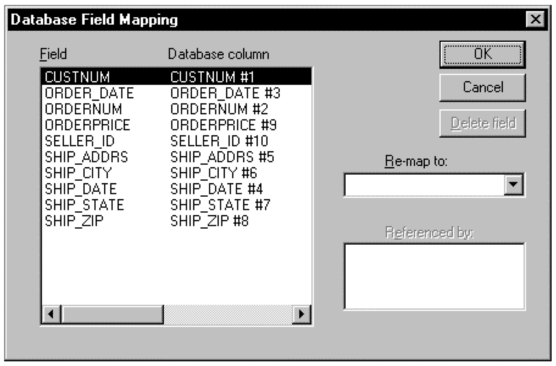
Whenever you create an additional query for a report that uses data from a different database, you may need to map the report definition fields to the database column names in the new query or set them to 0. This ensures that the new query data can be displayed on the report. If the database column names in the new query are exactly the same as those currently in the report definition, you do not need to map the fields.

If you create an additional query for a report that does not use one or more of the fields on the report, you need to map the report fields to 0 or null.

To map columns from a query to a report:

1. In the CA-Report Writer window, choose the Map Fields command from the Query menu.

   The Database Field Mapping dialog box appears, displaying the report fields that are already mapped:



Field	Database column
CUSTNUM	CUSTNUM #1
ORDER_DATE	ORDER_DATE #3
ORDERNUM	ORDERNUM #2
ORDERPRICE	ORDERPRICE #9
SELLER_ID	SELLER_ID #10
SHIP_ADDRS	SHIP_ADDRS #5
SHIP_CITY	SHIP_CITY #6
SHIP_DATE	SHIP_DATE #4
SHIP_STATE	SHIP_STATE #7
SHIP_ZIP	SHIP_ZIP #8

2. Click the field you want to map, and then click the Down arrow to display the Re-map To drop-down list box.
3. Select the database column to which you want to map the report field.

   If the field you are mapping is referenced by a computed field, the Referenced By list box shows all fields referenced.
4. Click OK.

Changing the Default Query Name

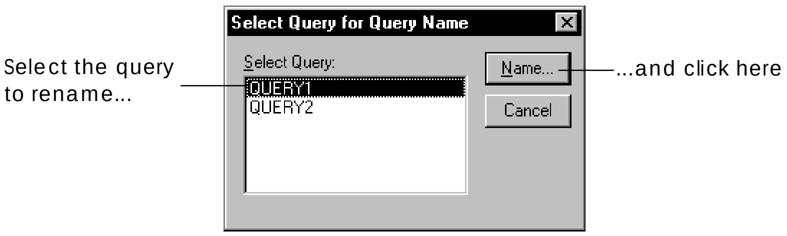
When a query is associated with a report definition, the CA-Report Writer gives the query a default name in the form QUERY1, QUERY2, and so on. You can change the default name to a more meaningful name and provide a description of the query. This is especially useful when you associate multiple queries with a single report definition.

Renaming a Query

To rename a query:

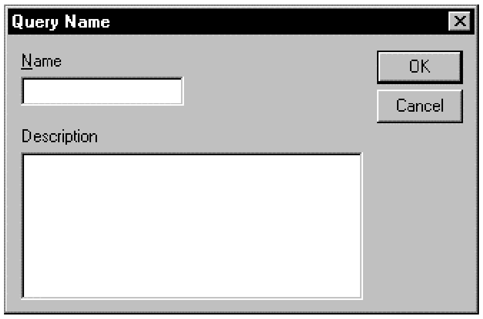
1. In the CA-Report Writer window, choose the Name command from the Query menu.

   The Select Query for Query Name dialog box appears:



2. If there is more than one query associated with the report, select the query you want to rename from the Select Query list box, and then click the Name button:

   The Query Name dialog box appears:



3. Enter the new name for the query in the Name edit control, and optionally enter a description.
4. Click OK.

Editing a Query

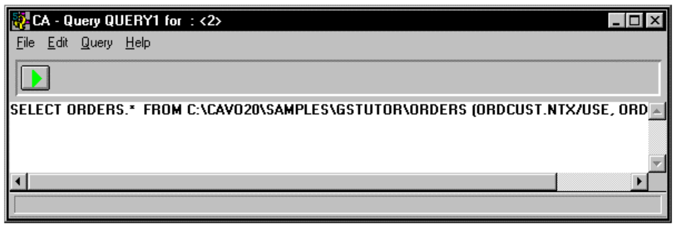
You can edit a report query when you need to add or delete columns or modify the query in any other way. When you edit a report query, you are working with a copy of the query, not the query file itself. Therefore, any changes you make to the report query are not reflected in the original file. You can export the report query to a query window if you want to save it as a separate query file. See Exporting a Report Query later in this chapter.

Editing a Report Query

To modify a query from within the CA-Report Writer window:

1. Choose the Edit Query command from the Query menu.

   If there is more than one query associated with the report, the CA-Report Writer displays the Edit Query dialog box for you to select the specified query for editing:

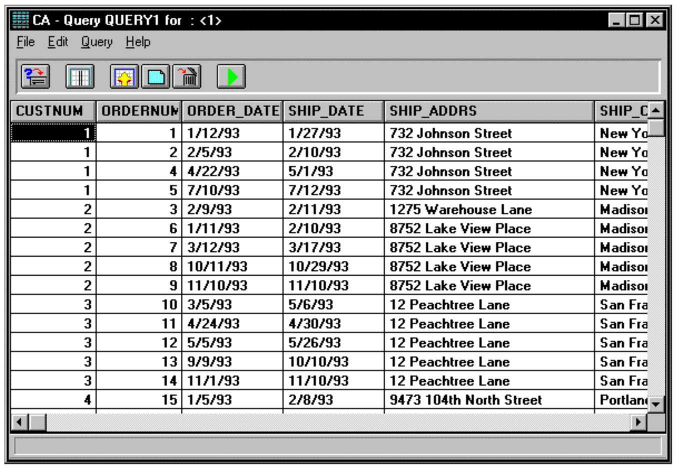


```
SELECT ORDERS.* FROM C:\CAVO20\SAMPLES\GSTUTOR\ORDERS (ORDCUST.NTX/USE, ORD
```

2. Choose the Run command from the Query menu to execute the query.

   Alternatively, click the Query dialog box's Execute tool bar button.

   The CA-Report Writer displays the Quick Query window where you can view and edit the query using various menu commands or toolbar buttons:



CUSTNUM	ORDERNUM	ORDER_DATE	SHIP_DATE	SHIP_ADDRS	SHIP_C
1	1	1/12/93	1/27/93	732 Johnson Street	New Yo
1	2	2/5/93	2/10/93	732 Johnson Street	New Yo
1	4	4/22/93	5/1/93	732 Johnson Street	New Yo
1	5	7/10/93	7/12/93	732 Johnson Street	New Yo
2	3	2/9/93	2/11/93	1275 Warehouse Lane	Madiso
2	6	1/11/93	2/10/93	8752 Lake View Place	Madiso
2	7	3/12/93	3/17/93	8752 Lake View Place	Madiso
2	8	10/11/93	10/29/93	8752 Lake View Place	Madiso
2	9	11/10/93	11/10/93	8752 Lake View Place	Madiso
3	10	3/5/93	5/6/93	12 Peachtree Lane	San Fra
3	11	4/24/93	4/30/93	12 Peachtree Lane	San Fra
3	12	5/5/93	5/26/93	12 Peachtree Lane	San Fra
3	13	9/9/93	10/10/93	12 Peachtree Lane	San Fra
3	14	11/1/93	11/10/93	12 Peachtree Lane	San Fra
4	15	1/5/93	2/8/93	9473 104th North Street	Portlan

3. Save your changes and close the Query window to return to the CA-Report Writer.

   The CA-Report Writer automatically applies any changes you made to the report definition file; however, you may need to modify the report definition itself.

   ■ If you added a column to the query, you must insert it in the report definition using the Insert Field menu command. (For more information, see Inserting, Moving, and Deleting Fields later in this chapter.)

   ■ If you deleted a column from the query, the CA-Report Writer displays a message telling you a column has been deleted. You can either set the field to 0, edit the query to include the column, or map the fields.

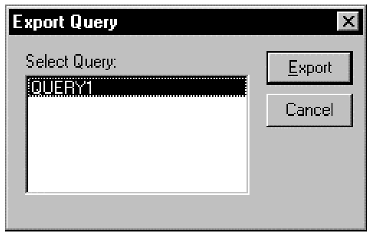
Exporting a Report Query

You can export a query from the CA-Report Writer to the appropriate query window if you want to save it in a separate query file. This is useful when the query used in the report definition does not exist in a separate query file—for example, if you did not save a query before using it in a report definition or you inadvertently deleted a query before saving it.

To export a query:

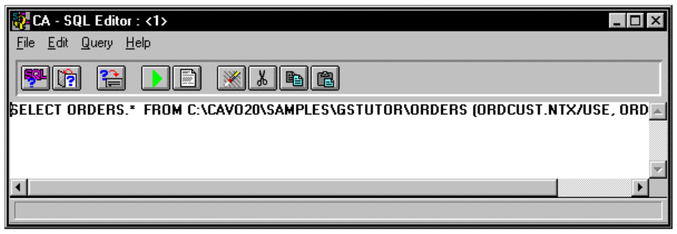
1. Choose the Export Query command from the Query menu.

   The Export Query dialog box appears:

2. Select the query to be exported from the list box.
3. Click Export.

   The CA-Report Writer exports the query and displays it in the appropriate query window. For example:

   SELECT ORDERS.* FROM C:\CAVO20\SAMPLES\GSTUTOR\ORDERS (ORDCUST.NTX/USE, ORD

4. Save it as a separate query file using the File Save As menu command.

Designing Reports

When you create a new report from scratch—or customize a default report created by CA-Report Writer—you can use any of the following report design techniques:

- Add, change, or delete report sections

  A report section is the basic unit of layout in a report definition. Each section can contain fields, text, and graphics. There are eight types of sections, which are described later in Working with Sections.

- Add report details, including text, fields, and graphics

  Text includes field labels and page and report headers and footers. (See Adding Report Details to Sections for detailed information.)

  A field is a reference in a report definition to an individual piece of data. It can be a database field, which displays information from one column in a query result or it can be a computed field, which displays a value computed from the data in a function or formula. (See Working with Tables for detailed information.)

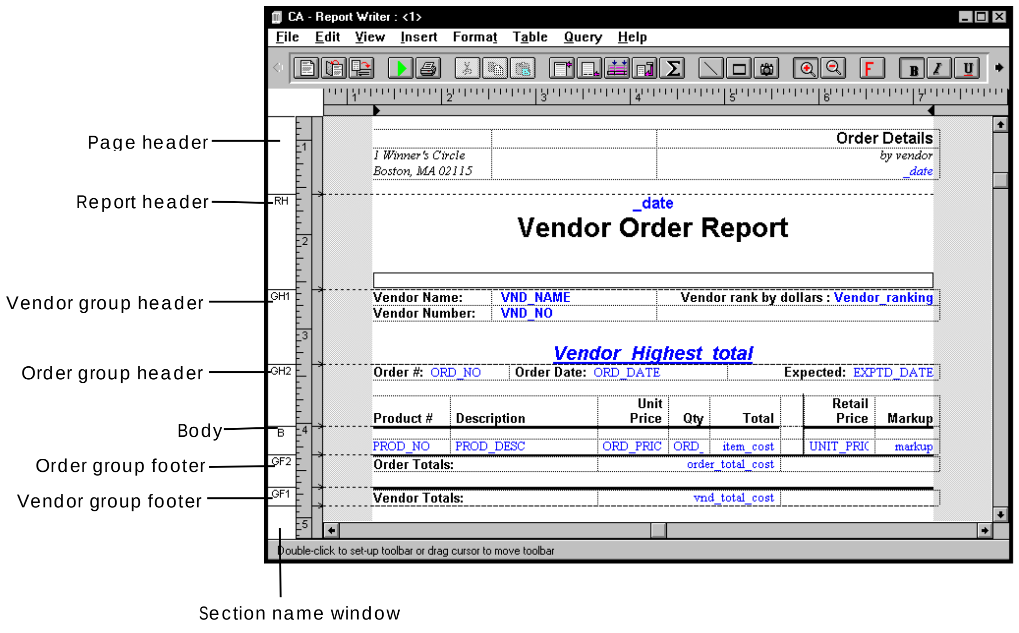
  Graphics—logos, designs, photos, or other pictures—can be imported into your report. You can also draw lines and rectangles. (See Adding Graphics for detailed information.)

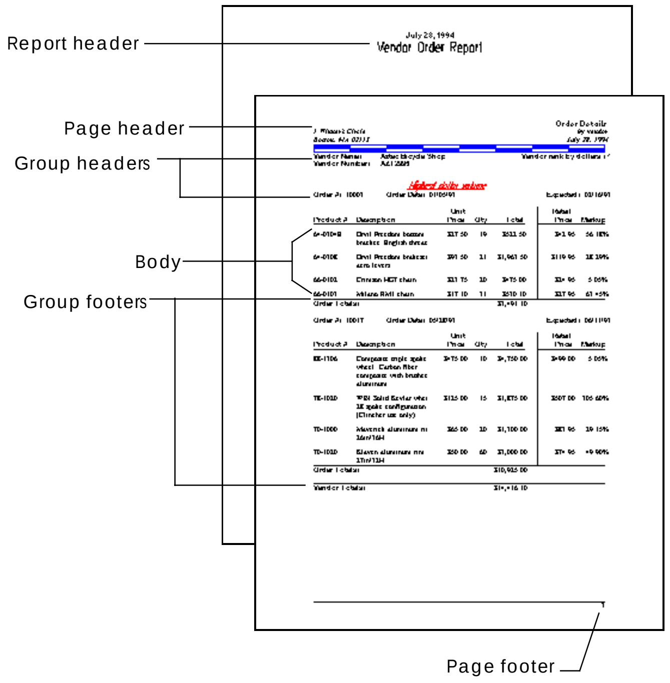
- Format the page display

  You can format report page margins; format the alignment of text, fields, and data on the report page, place borders around the report and apply fonts styles, and color to the report contents. (See Formatting a Report's Appearance for detailed information.)

Working with Sections

A *report section* is the basic unit of layout in a report definition. A report definition must have at least one section. The sections you use and the information you place in them depends on the type of report you are creating. The following illustrations show some of the section types you can use in CA-Report Writer reports:





Report Section Types

As stated earlier, you can use eight types of sections in a CA-Report Writer report:

Section Type	Description
Page Header (PH)	Appears at the top of each report page. You can use a page header section for letterheads, logos, subtitles, dates, and page numbers.
Report Header (RH)	Appears at the beginning of the report. You can use a report header section for report titles and dates. It can print as a separate report title page.
Body (B)	Contains the main body of the report's data. The entire body section prints for each row of data retrieved from your database.

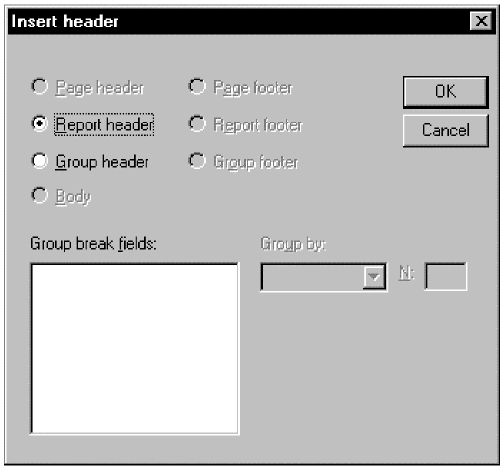
Section Type	**Description**
Group Header (GH)	Appears at the beginning of a group of sections. A group header section can identify a group of data that appears in another section following the group header. For example, a customer transaction report would need a group header for each customer name in the database, so that all transactions for a particular customer are grouped together in the body section of the report. For more information, see Grouping Data later in this chapter.
Group Footer (GF)	Appears at the end of a group of sections. You can use a group footer to display summary information, including group subtotals.
Report Footer (RF)	Appears at the end of the report. If there is a page footer, the report footer appears before it. You can use it to display report summary information, such as grand totals.
Page Footer (PF)	Appears at the bottom of each report page. You can use a page footer for page numbers and company information.
Background	Spans the background of the entire report page. A background section is independent of all other sections; you can use this type of section to display background graphics.

Adding a Section to a Report

To add a section to a report:

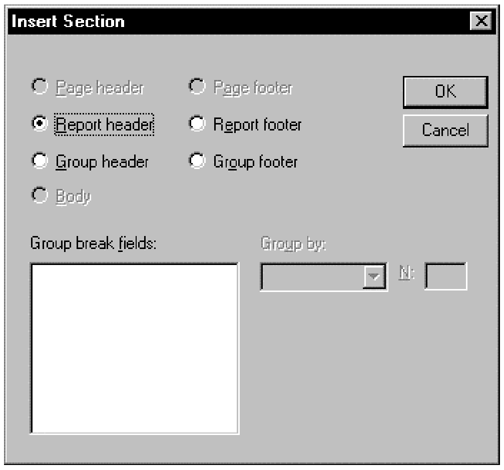
1. Click either the Insert Header or Insert Footer toolbar button.

   The Insert Header or Insert Footer dialog box, respectively, appears. For example:



   Alternatively, choose the Section command from the Insert menu.

   The Insert Section dialog box appears:



2. Depending on what you did in step 1, do one of the following:

   - In the Insert Header or Insert Footer dialog box, select the header or footer type you want to insert, and then click OK.

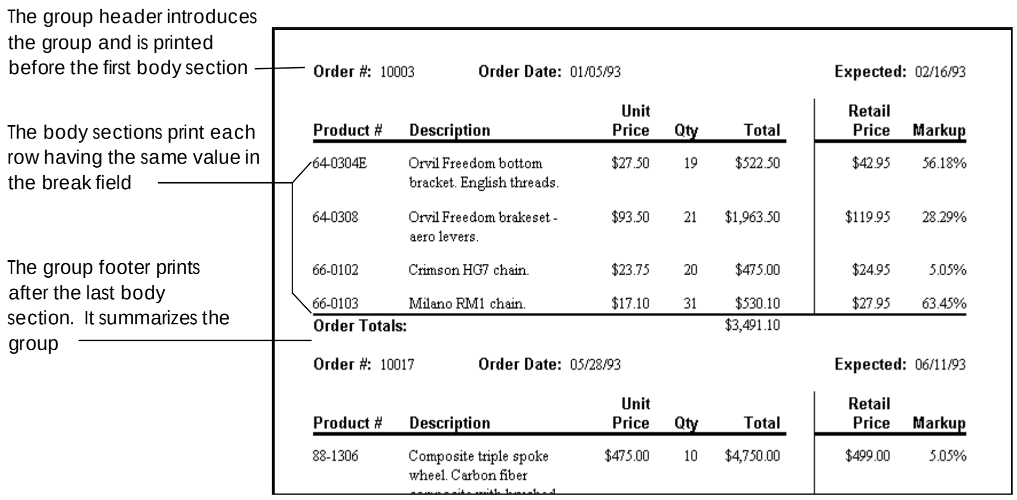
     Note: If you are adding a group header or group footer, you must specify a break field in the dialog box. See Working with Group Headers and Footers in the next section for details about specifying break fields.

- In the Insert Section dialog box, select the section type you want to insert, and then click OK.

The CA-Report Writer automatically includes a background section in each report definition.

Working with Group Headers and Footers

When you add a group header or group footer section, you must specify at least one *break field*. A break field is a field by which you want to group rows in a report. The CA-Report Writer prints the body section for each row that has an identical value in the break field. For example:



The group header introduces the group and is printed before the first body section

The body sections print each row having the same value in the break field

The group footer prints after the last body section. It summarizes the group

Order #: 10003 **Order Date:** 01/05/93 **Expected:** 02/16/93

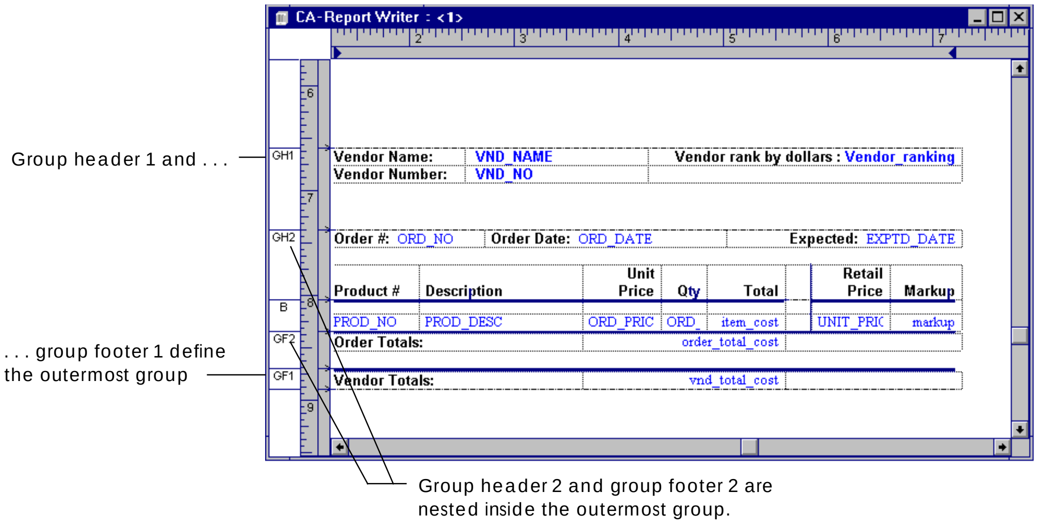
Product #	Description	Unit Price	Qty	Total	Retail Price	Markup
64-0304E	Orvil Freedom bottom bracket. English threads.	$27.50	19	$522.50	$42.95	56.18%
64-0308	Orvil Freedom brakeset - aero levers.	$93.50	21	$1,963.50	$119.95	28.29%
66-0102	Crimson HG7 chain.	$23.75	20	$475.00	$24.95	5.05%
66-0103	Milano RM1 chain.	$17.10	31	$530.10	$27.95	63.45%
Order Totals:				$3,491.10		

Order #: 10017 **Order Date:** 05/28/93 **Expected:** 06/11/93

Product #	Description	Unit Price	Qty	Total	Retail Price	Markup
88-1306	Composite triple spoke wheel. Carbon fiber	$475.00	10	$4,750.00	$499.00	5.05%

If you use more than one group, the CA-Report Writer nests the groups in the order in which you add them to the report. The first group header or footer you add is group level 1, the next is level 2, and so on.

Note: For a group break to make sense, the report data must be sorted in the same order as the group breaks. The query should contain a sort column that orders the data in the same order as the nested groups of the report. For example:


Group header 1 and . . .
. . . group footer 1 define the outermost group
Group header 2 and group footer 2 are nested inside the outermost group.
CA-Report Writer : <1>
GH1
GH2
B
GF2
GF1

Vendor Name: VND_NAME Vendor rank by dollars : Vendor_ranking
Vendor Number: VND_NO

Order #: ORD_NO Order Date: ORD_DATE Expected: EXPTD_DATE

Product #	Description	Unit Price	Qty	Total	Retail Price	Markup
PROD_NO	PROD_DESC	ORD_PRIC	ORD_	item_cost	UNIT_PRIC	markup

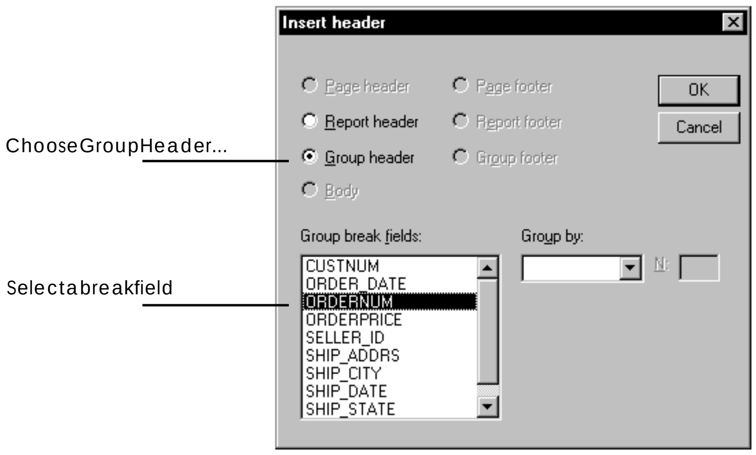
Order Totals: order_total_cost

Vendor Totals: vnd_total_cost

To add a group header or group footer to a report:

1. Click either the Insert Header or Insert Footer toolbar button.

   The Insert Header or Insert Footer dialog box, respectively, appears. For example:



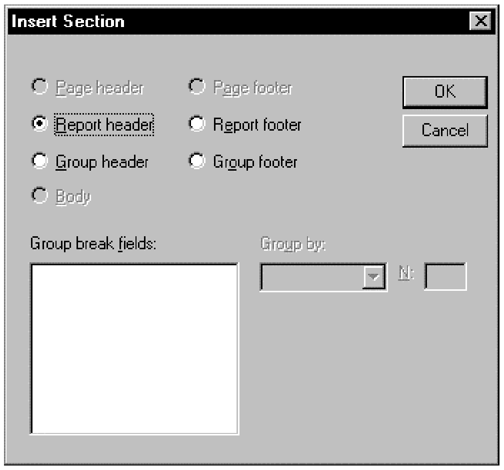
Page header / Page footer
Report header / Report footer
[✓ Group header] / Group footer
Body

Group break fields:
CUSTNUM
ORDER_DATE
ORDERNUM
ORDERPRICE
SELLER_ID
SHIP_ADDRS
SHIP_CITY
SHIP_DATE
SHIP_STATE

Group by:
N:

Alternatively, choose the Section command from the Insert menu.

The Insert Section dialog box appears:



2. Depending on what you did in step 1, do one of the following:
   - In the Insert Header or Insert Footer dialog box, select Group Header or Group Footer.
   - In the Insert Section dialog box, select Group Header or Group Footer.
3. Select the field you want to use as a break field from the Group Break Fields list box.
4. To specify a break field based on a partial change in the field, select a value from the Group By list box.

   Valid values are:
   - **Text field**

     First n characters by which to group fields
   - **Numeric field**

     Intervals by which to group fields
   - **Date field**

     Length of time by which to group fields
5. Click OK to add the section to the report definition.

After you add a group header or footer section, you can add text or fields to it. For example, you can label a group by inserting the break field in the group header section. You can print subtotals, group averages, or other summary information by adding computed fields. For more information about adding fields, see Using Report Fields later in this chapter.

Tip: Use care when placing data fields into group headers or footers to ensure that the result makes sense. If you insert a field that is not the break field, the value displayed in the report is taken from one of the rows in the group. The field's value may be merely a random example, rather than a characteristic value in common with all rows for that group.

Adding Report Details to Sections

You can add fields, text, and graphics to each section. Here are some guidelines and restrictions you should be aware of when adding details to a report section.

Page Headers and Footers

Page header and page footer sections can include page number functions, date and time fields, and database fields. They cannot include other functions or computed fields. A page footer section has a fixed length. It truncates any text or fields that extend beyond the bottom page margin.

Report Headers and Footers

If you want to print a title page for a report, you can format a report header to print before the page header. If you want to print a calculated field in the report header on each page, you can format a report header to print on subsequent pages.

Group Headers and Footers

When you insert a group header or footer into the report definition, you must specify one or more fields by which you want to group records in the report. See Grouping Data later in this chapter for more information.

Background

To insert information into the background section, select the View Page Background menu command. You can add page number functions, date and time fields, and database fields. You cannot add any other functions or computed fields. The CA-Report Writer prints information defined in the background section the same way on each page of the report.

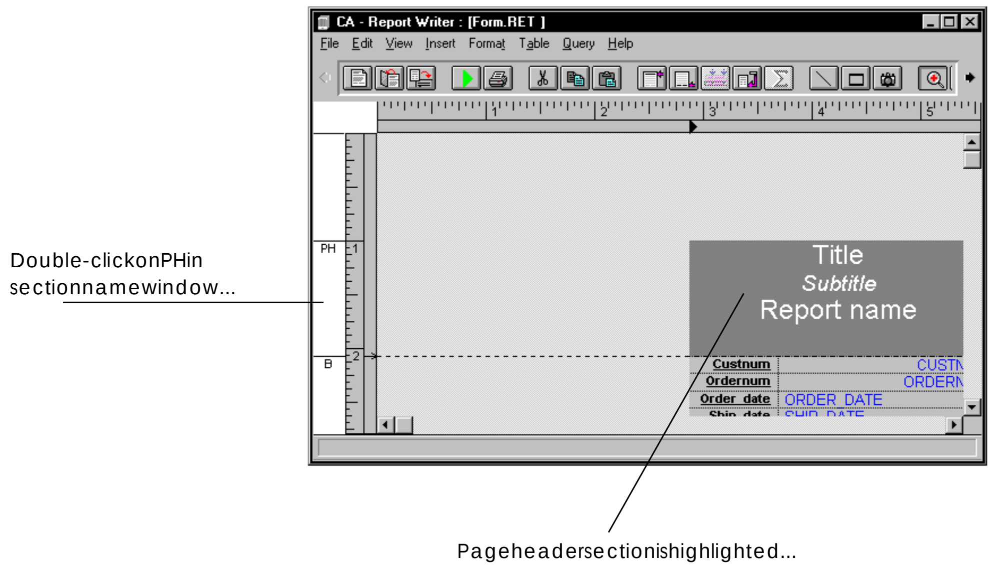
Formatting Sections

You can set attributes for a section in order to control how it prints and/or to improve its readability. There are three dialog boxes used for section formatting; the available options vary for each section type.

Basic Steps for Section Formatting

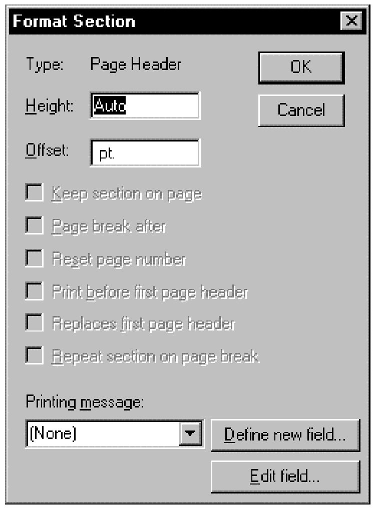
To format a section, follow these basic steps:

1. Double-click the section name in the section name window (for example, **PH**):


CA - Report Writer : [Form.RET]
File Edit View Insert Format Table Query Help
PH
B
Title
Subtitle
Report name
Custnum CUSTN
Ordernum ORDERN
Order date ORDER_DATE
Ship date SHIP_DATE
Double-clickonPHin sectionnamewindow...
Pageheadersectionishighlighted...

Alternatively, position the cursor in the desired section of the report definition, and then choose the Section command from the Format menu.

Depending on the section type you have selected, the Format Section, Format Group Section, or Format Body Section dialog box appears. For example:



Type: Page Header
Height: Auto
Offset: pt.
☐ Keep section on page
☐ Page break after
☐ Reset page number
☐ Print before first page header
☐ Replaces first page header
☐ Repeat section on page break
Printing message: (None)

2. Choose the desired options, which are described in the next section.

   Note: Some options are not available for certain section types.

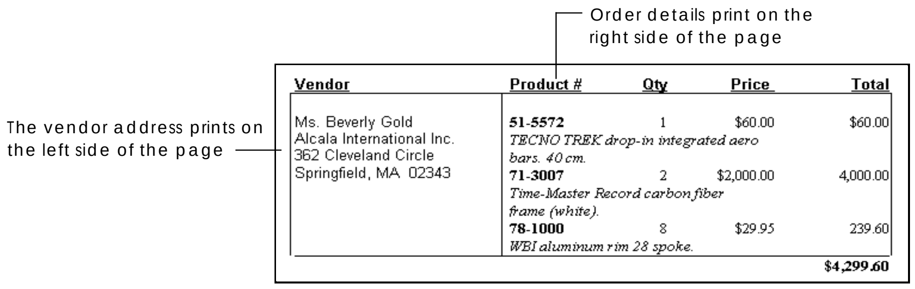
3. Click OK.

Section Formatting Options

Options for section formatting can include some or all of the following:

Height	Overrides the default Auto section height. Enter a value in inches (1.00, for example). **Note:** By default, the CA-Report Writer uses the minimum height necessary to display all the text, data, and graphics in a section. If you specify a height less than the minimum necessary to print the section, the CA-Report Writer adjusts the height when you specify the Auto option.

Offset Prints a group header and body section detail lines side by side by specifying an *offset* for the body section, moving the first line up the page. (Make sure the value you enter does not make sections overlap.)


Order details print on the right side of the page
The vendor address prints on the left side of the page

Vendor	Product #	Qty	Price	Total
Ms. Beverly Gold Alcala International Inc. 362 Cleveland Circle Springfield, MA 02343	**51-5572** *TECNO TREK drop-in integrated aero bars. 40 cm.*	1	$60.00	$60.00
	71-3007 *Time-Master Record carbon fiber frame (white).*	2	$2,000.00	4,000.00
	78-1000 *WBI aluminum rim 28 spoke.*	8	$29.95	239.60
				$4,299.60

Keep Section on Page Prevents the section from moving to another page in the report.

Keep with Next Section Keeps this section with the next section in the report.

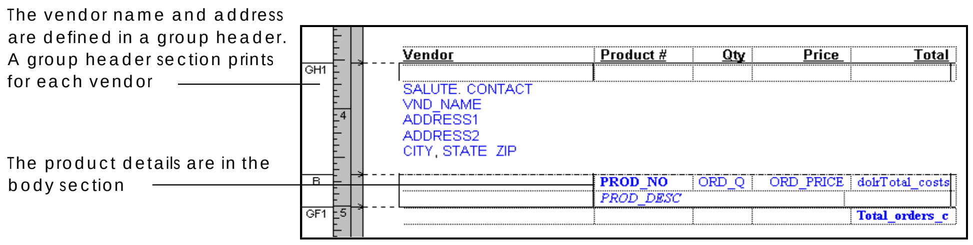
Page Break After Inserts a page break after this section.

Reset Page Number Resets the page number to 1 whenever report header or footer sections are processed. For example, you would want to reset the page number to 1 in an invoice report whenever the invoice recipient changes.

Repeat Section on Page Break Controls where group header sections appear by repeating a group header following a page break. When a page break occurs while printing a body section in a grouping, the group heading associated with the body section is printed at the top of a new page.

Underline Last Row Underlines the last body section in a group.

Replace Page Footer Controls where group footer sections appear by treating a group footer as a page footer. You can set this option for only one group footer—if a page footer section is also defined, the page footer prints only on those pages where no group footer prints.


The vendor name and address are defined in a group header. A group header section prints for each vendor
The product details are in the body section
GH1
4
B
GF1
5

Vendor Product # Qty Price Total
SALUTE CONTACT
VND_NAME
ADDRESS1
ADDRESS2
CITY, STATE ZIP
PROD_NO ORD_Q ORD_PRICE dolrTotal_costs
PROD_DESC
Total_orders_c

Insert Blank After Inserts a blank line after every *n*th body section when you enter a value in the text box.

Label Style Presents report data in column format (body sections only). This is used in label reports.

Printing Message	Specifies a printing message to use when printing a report.
Break Fields	Specifies the break field for a group section. When you change a break field in a report, you must also revise the sort order in the query used in the report.
Group Level	Changes the relative nesting level of a group section. For example, you can change a report that displays orders by product within vendor to one that displays orders by vendor within product. Nesting level 1 represents the outermost group; level 2 the first group, and so on. If there is a corresponding group header or footer, you must also change the nesting level for that section.
Group By	Specifies the group intervals for group sections.

Deleting a Section

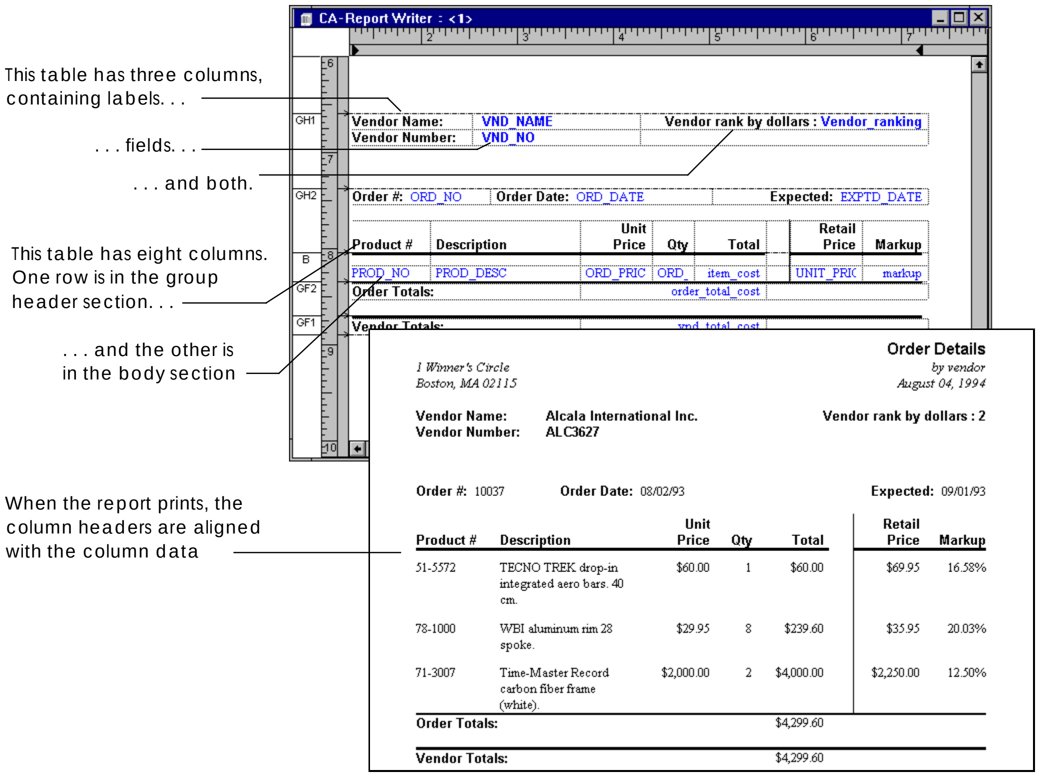
To delete a section:

1. Position the cursor on the appropriate section in the section name window or in the section area of the vertical ruler in the CA-Report Writer window.
2. Choose the Delete command from the Edit menu.

Working with Tables

You can align data, text, and graphics in different sections of a report by using a *table*, which contains rows (horizontal) and columns (vertical) of data.

Each table contains *cells*, which are the intersections of rows and columns. You can insert fields or text into one or more paragraphs in a cell, allowing you to align related information in different sections of a report. For example:


This table has three columns, containing labels. . .
. . . fields. . .
. . . and both.
This table has eight columns. One row is in the group header section. . .
. . . and the other is in the body section
When the report prints, the column headers are aligned with the column data
CA-Report Writer : <1>
GH1
GH2
B
GF2
GF1

Vendor Name:	VND_NAME	Vendor rank by dollars : Vendor_ranking
Vendor Number:	VND_NO	

Order #: ORD_NO | Order Date: ORD_DATE | Expected: EXPTD_DATE

Product #	Description	Unit Price	Qty	Total		Retail Price	Markup
PROD_NO	PROD_DESC	ORD_PRIC	ORD_	item_cost		UNIT_PRIC	markup
Order Totals:				order_total_cost			
Vendor Totals:				vnd_total_cost			

Order Details
by vendor
August 04, 1994

1 Winner's Circle
Boston, MA 02115

Vendor Name: Alcala International Inc.
Vendor Number: ALC3627
Vendor rank by dollars : 2

Order #: 10037 **Order Date:** 08/02/93 **Expected:** 09/01/93

Product #	Description	Unit Price	Qty	Total	Retail Price	Markup
51-5572	TECNO TREK drop-in integrated aero bars. 40 cm.	$60.00	1	$60.00	$69.95	16.58%
78-1000	WBI aluminum rim 28 spoke.	$29.95	8	$239.60	$35.95	20.03%
71-3007	Time-Master Record carbon fiber frame (white).	$2,000.00	2	$4,000.00	$2,250.00	12.50%
Order Totals:				$4,299.60		
Vendor Totals:				$4,299.60		

Defining Table Styles

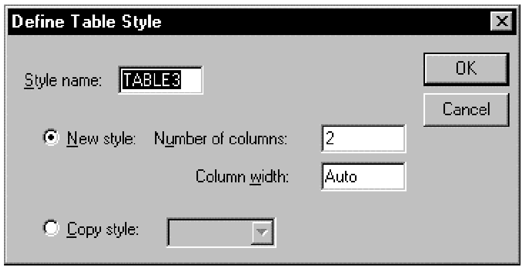
You can define a unique column (vertical) layout, called a *style*, for each table—rows in a table are aligned with the same style. You define a table style by creating a new style or copying an existing style. You can define up to 50 table styles for each report definition.

Defining a New Table Style

To define a new table style for your report definition:

1. Position the cursor where you want to insert a row using the new table style.
2. Choose the Define Table Style command from the Table menu.

   The Define Table Style dialog box appears:

   Style name: TABLE3

   ● New style: Number of columns: 2

   Column width: Auto

   ○ Copy style:

   Note that the New Style radio button is selected, by default.

3. Enter a name in the Style Name edit control for the new table style.

   The default is TABLE*n*.

4. Optionally, enter a different value in the Number of Columns edit control to override the default setting.
5. In the Column Width edit control, specify the width of each column in the table.

   Note: You can specify Auto to have the CA-Report Writer automatically size columns.

6. Click OK.

Copying an Existing Table Style

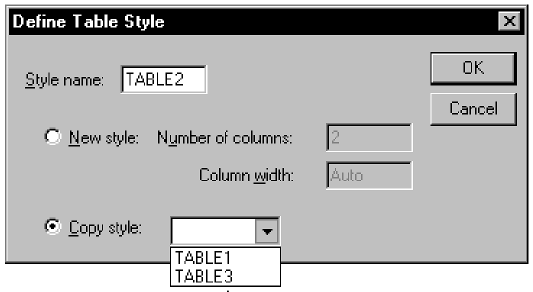
To copy an existing table style:

1. Position the cursor where you want to insert a row using the new table style.
2. Choose the Define Table Style command from the Table menu.

   The Define Table Style dialog box appears.

3. Enter a name in the Style Name edit control for the new table style definition.
4. Select the Copy Style radio button.

5. Select an existing table style from the drop-down list to be copied to the new table style definition:



Style name: TABLE2
Number of columns: 2
Column width: Auto
Copy style: TABLE1, TABLE3

Selecttabletobecopied

6. Click OK.

Deleting a Table Style

You can delete any table styles that are not currently being used. If a table style appears in a report definition, the Delete Table Style option is *not* available. (In such cases, you must first delete any rows defined in the table before you can delete the table style.)

To delete a table style:

1. Choose the Delete Table Style command from the Table menu..

   The Delete Table Style dialog box appears:

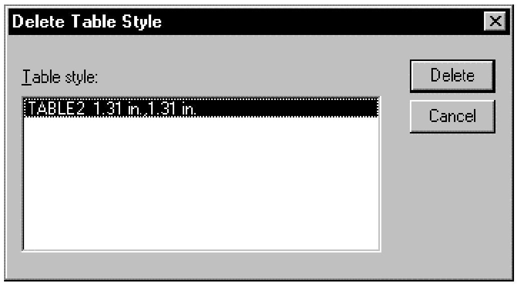


TABLE2 1.31 in. 1.31 in.

2. From the Table Style list box, select the style you want to delete.
3. Click the Delete button.

Using Columns in Tables

The columns in a table function like tab stops to align text. Text you enter in a report section normally wraps at the right page margin. In a column, text also wraps, starting a new line when the width of the text exceeds the space between the column margins. This section tells you how to add columns to a table style, format the columns, move columns, and delete columns.

The CA-Report Writer indicates where columns and rows appear in a report definition by displaying vertical and horizontal divider lines respectively. You can change the display of column and row dividers by choosing the Column/Row Dividers command from the Table menu.

Tip: Use your keyboard to move around in a table. The Tab key places the cursor at the start of the next cell in a row (or paragraph). To move the cursor to the start of the previous cell in a row (or paragraph), use Shift+Tab.

Adding Columns to a Table

To add a column to a table in your report definition:

1. Position the cursor in the column *after* which you want to insert a new column.
2. Choose the Insert Column command from the Table menu.

   The CA-Report Writer inserts a new column in all occurrences of the table style, including those in different sections.

Changing Column Width

Columns are separated by an area called the *gutter*. You can adjust the width of any column or column gutter in a report style. When you change a column width, it changes for all rows of the same table style in the report.

Note: Increasing the column width may cause other columns to extend beyond the page margins—you may need to decrease the width of these columns.

You can change the width of a column using either the Format Columns dialog box or the ruler method.

Dialog Box Method

To change column width in a report definition:

1. Position the cursor in the desired column.
2. Double-click between the column markers for the column in the ruler.

Alternatively, choose Format Columns from the Table menu.

The Format Columns dialog box appears:

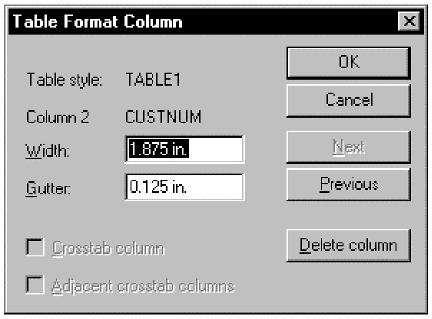


Table style: TABLE1

Column 2 CUSTNUM

Width: 1.875 in.

Gutter: 0.125 in.

3. Enter a new value in the Width edit control to change the column width.
4. Enter a value in the Gutter edit control to change the gutter width.
5. Click the Next or Previous button to move to and format the next or previous column in the table.
6. Click the Delete Column button if you want to delete this column from the table.

   Note: Use the Crosstab Column and Adjacent Crosstab Columns options to format columns in cross tabular reports. See Putting Columns of Like Data Together later in this chapter for information about crosstab columns in report tables.
7. Click OK to return to the report definition.

The CA-Report Writer shifts the columns according to your dialog box selections.

Ruler Method

To change column width in a report definition using the ruler:

1. Position the cursor in the desired column.
2. Drag either column marker in the ruler to the desired position.

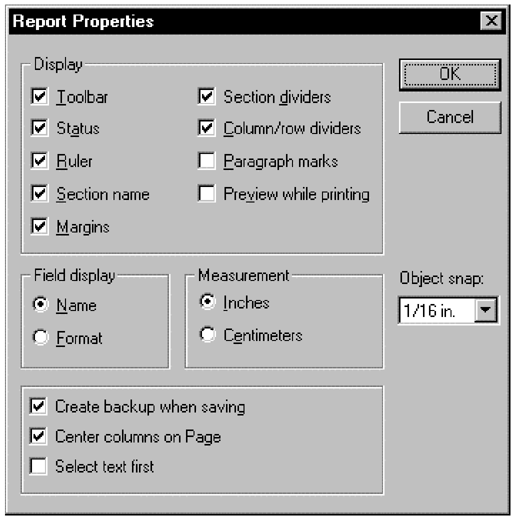
   Note: If you shrink a column, the CA-Report Writer shifts subsequent columns to the left. Expanding a column shifts subsequent columns to the right.

Centering Columns

To center a column within your report definition:

1. Position the cursor in the desired column.
2. Click the Center paragraph alignment toolbar button. The column is centered immediately within your report definition.
3. Alternatively, choose the Properties command from the File menu.

The Report Properties dialog box appears:


Report Properties

Display
- [✓] Toolbar
- [✓] Status
- [✓] Ruler
- [✓] Section name
- [✓] Margins
- [✓] Section dividers
- [✓] Column/row dividers
- [ ] Paragraph marks
- [ ] Preview while printing

Field display: [✓ Name] Format

Measurement: [✓ Inches] Centimeters

Object snap: 1/16 in.

- [✓] Create backup when saving
- [✓] Center columns on Page
- [ ] Select text first

4. Select the Center Columns On Page option.
5. Click OK.

Moving Columns

You can change the sequence of columns in a table style by moving columns. The content and format of the columns remains the same.

Follow these basic steps to move a column within a table in your report definition:

1. To display the column markers on the ruler, select the style you want to delete.
2. To select the column, click the ruler at the location of the column you want to move.
3. With the cursor on the ruler, drag the column across the report definition until its center is on the column that you want to position it in front of.

   The selected column is highlighted by a dashed outline while you drag it across the report.
4. To move a column to the beginning of the report, move the center of the selected column over the first column in the report.

Deleting Columns

You can delete a column from a table style, including any text or fields inside the column. The CA-Report Writer deletes the selected column from all rows of the table style in the report—any new rows will *not* contain the column.

To delete a column from a table in a report definition:

1. Position the cursor in the column you want to delete.

   Alternatively, click the column in the horizontal ruler (with the column markings for the table style displayed).

2. Choose the Delete Column command from the Table menu.

   The specified column is removed.

3. Alternatively, choose the Format Columns command from the Table menu.

   The Table Format Column dialog box:

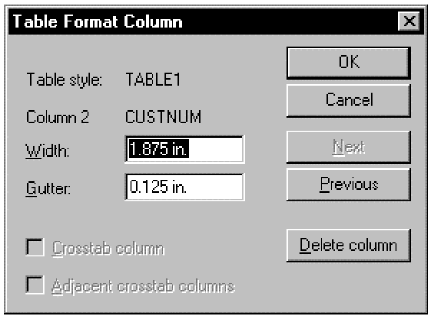


Table style: TABLE1
Column 2 CUSTNUM
Width: 1.875 in.
Gutter: 0.125 in.

4. Click the Delete Column button.
5. Click OK.

Defining an Adjacent Crosstab Column

If your cross tabular report contains more than one aggregate value, the CA-Report Writer normally displays all aggregates for the first column before going on to the next. If you want to put columns of like data next to each other, such as the quantity of a product ordered and the number of orders in a fiscal quarter, you can define each additional column in the report as an adjacent crosstab column. For example:

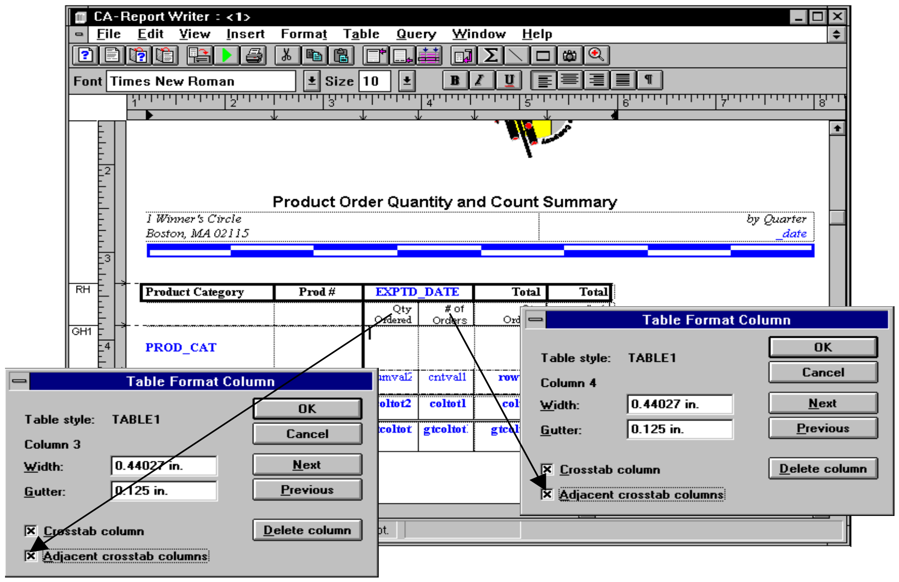


Table Format Column
Table style: TABLE1
Column 3
Width: 0.44027 in.
Gutter: 0.125 in.
[✓] Crosstab column
[✓] Adjacent crosstab columns

Table Format Column
Table style: TABLE1
Column 4
Width: 0.44027 in.
Gutter: 0.125 in.
[✓] Crosstab column
[✓] Adjacent crosstab columns

To define an adjacent crosstab column in a table within your report definition:

1. Place the cursor on the crosstab column.
2. Choose the Format Columns command from the Table menu.

   The Table Format Column dialog box appears.
3. Select the Adjacent Crosstab Columns option.
4. Click OK.

Working with Rows

Once a table style is defined, you can insert rows of the table into any section in the report definition. All rows in a table have the same column alignment, even if they are in different report sections. Each row can contain different text and fields. You add text and fields to the cells of a row the same way you add them to any area of a report definition. You do not have to place contents in every cell.

This section describes how to add, move, and delete rows, and how to convert rows to text and text to rows.

Adding Rows to Tables

To add a row to a table:

1. Position the cursor in the table where you want to insert a new row.
2. Click the Insert Row toolbar button.

   Alternatively, choose Insert Row from the Table menu.

   If there is more than one table style defined in the report, the CA-Report Writer displays the Insert Table Row dialog box. For example:

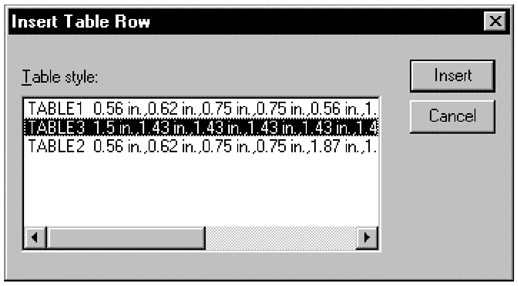
   

   Table style:

   TABLE1 0.56 in.,0.62 in.,0.75 in.,0.75 in.,0.56 in.,1.
   TABLE3 1.5 in.,1.43 in.,1.43 in.,1.43 in.,1.43 in.,1.4
   TABLE2 0.56 in.,0.62 in.,0.75 in.,0.75 in.,1.87 in.,1.

   All of the table styles defined for the report are listed in the Table Styles list box.
3. Select the table style you want to apply to the row, and then click the Insert button.

The CA-Report Writer inserts a new row of the selected table style.

Note: If there is only one table style, the new row is automatically inserted.

Moving Rows

When you move a row in a table, the CA-Report Writer moves the entire row, including text and fields, to the new location.

To move a row within a table in your report definition:

1. Position the cursor anywhere in the row.
2. Choose the Select Row command from the Table menu.
3. Click the Cut toolbar button.

   Alternatively, choose Cut from the Edit menu.
4. Move the cursor to the location where you want to insert the cut row.
5. Click the Paste toolbar button.

   Alternatively, choose Paste from the Edit menu.

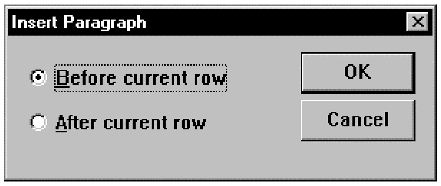
Note: If you paste the row onto another row, the CA-Report Writer merges the contents of each cell.

Inserting Blank Lines Between Rows

To insert a blank line, or *paragraph*, between rows in a table:

1. Position the cursor on either row you want to insert the blank line between.
2. Choose the Insert Paragraph command from the Table menu.

   The Insert Paragraph dialog box appears:

3. Select either the Before Current Row or After Current Row option to insert the new (blank) paragraph before or after the current row, respectively.
4. Click OK.

Deleting Rows

When you delete a row from a report definition, the CA-Report Writer deletes all row cells and cell contents.

To delete a row from a table in a report definition:

1. Position the cursor anywhere in the row.
2. Choose Delete Row from the Table menu.

Converting Text to Rows and Rows to Text

To move text from one area of the report definition into a table row, you can convert the text into a row. When you convert text to a row, each paragraph in the text is placed in a separate column.

You should select a table style that has the same number of columns as text paragraphs you are converting. Otherwise, you will have unused columns. For example, to convert two paragraphs of text to a row, place the cursor in front of the first paragraph; then select a table style that has two columns.

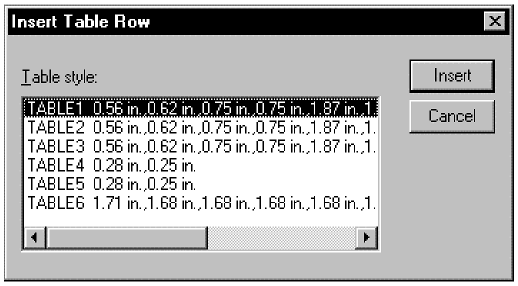
You can also move text out of a table and into another part of the report definition by converting the contents of a table row to page-aligned text. The row contents are placed in the new location as a series of paragraphs aligned with the page.

Text to Row

To convert text to a row within a report definition:

1. Position the cursor in front of the text you want to convert.
2. Choose Convert Text To Row from the Table menu.

   The Insert Table Row dialog box appears, if a table has been defined previously for this report:



```
TABLE1  0.56 in.,0.62 in.,0.75 in.,0.75 in.,1.87 in.,1
TABLE2  0.56 in.,0.62 in.,0.75 in.,0.75 in.,1.87 in.,1.
TABLE3  0.56 in.,0.62 in.,0.75 in.,0.75 in.,1.87 in.,1.
TABLE4  0.28 in.,0.25 in.
TABLE5  0.28 in.,0.25 in.
TABLE6  1.71 in.,1.68 in.,1.68 in.,1.68 in.,1.68 in.,1.
```

3. From the Table Style list box, select the table style you want to use for the text.
4. Click Insert.

   The CA-Report Writer moves each paragraph of the selected text into one column in the selected table style.

Row to Text

To convert a table row to text:

1. Position the cursor anywhere in the row you want to convert.
2. Choose the Convert Row To Text command from the Table menu.

   The CA-Report Writer deletes the row and aligns the text as a series of paragraphs within the normal page margins of the report.

Selecting a Table Cell for Editing

You can easily select the contents of a cell and format, move, copy, or delete its contents.

To select a table cell within a report definition:

1. Position the cursor in the cell.
2. Drag the cursor over the cell.

   Alternatively, choose Select Cell from the Table menu.

Once the cell is selected, then edit, copy, delete, or move the cell's contents.

Note: Pasting a selected cell onto another cell adds to the contents of that cell. You can also paste cell contents to a non-table area. However, if you paste more than one cell into a non-table area, the CA-Report Writer inserts a row with the contents of the selected cells copied.

Formatting a Report's Appearance

You can customize the display of your reports by:

- Aligning data
- Formatting the display of text on the page
- Adding borders around parts of a report
- Using conditional paragraphs
- Changing fonts and point sizes
- Applying bold, italics, and underlining
- Applying color
- Creating a report title page
- Controlling page breaks

When you format the appearance of the report, you are formatting paragraphs and characters.

Paragraph Formatting

A *paragraph* is any area in a report definition that ends with a paragraph mark (¶). For example, each column heading in a report header is a separate paragraph. A paragraph can be the contents of one cell or an entire section of a report. Pressing the Enter key inserts a paragraph mark. You can have many paragraphs within a block of text, and one cell can contain multiple paragraphs.

With paragraph formatting, you can do the following:

- Left-justify, right-justify, center, or justify the contents of a paragraph in a report
- Wrap, truncate, and specify line spacing for text displays in reports
- Place borders around selected paragraphs in reports

Tip: When formatting paragraphs, turn on the display of paragraph marks by clicking the Paragraph Mark toolbar button or by turning on the Paragraph Marks option in the Report Properties dialog box (Properties command in the File menu).

You can specify the format of a paragraph before you enter information in it. The CA-Report Writer applies the formatting as you enter text, and continues to use this formatting on new paragraphs until you reset it or reposition the cursor. For example, if you start a new paragraph, it has the same attributes as the previous one. Whenever you copy or move a paragraph, the CA-Report Writer also copies or moves the formatting.

Character Formatting

Character formatting determines how report text—including letters, numbers, punctuation, and symbols—appears on the screen and in print. Character formatting lets you choose:

- Fonts and point sizes
- Underlined, bold, or italic styles
- Color

You can select character formatting before you type, or selectively after you have typed text in your report. Select any number of characters, and then apply or remove character formatting. The current character formatting is displayed on the status bar (at the bottom of the CA-Report Writer window) and is highlighted on the toolbar.

Aligning Paragraph Text

You can align paragraph text with the left margin, the right margin, both margins simultaneously (justified), or you can center the text between the right and left margins. Column data aligns with the column margins. Data outside a column aligns with the page margins of the report.

To align text within the report definition:

1. Position the cursor in the paragraph you want to align.
2. Click one of the following toolbar buttons:

   Left　　Center

   Right　　Justify

Tip: Drag the cursor to select more than one paragraph.

Formatting Text Displays

Text display refers to the placement of text (or database fields containing text) that is too large to be displayed on a single line within the column or page margins of a report. The CA-Report Writer either wraps the text (continuing it on the next line), or truncates it at the column or page margin. Usually, wrapping applies to text fields, but not number or date fields.

When you preview a report, the CA-Report Writer displays numbers that are too large for a column as a sequence of pound (#) symbols.

Note: By default, the CA-Report Writer left-aligns and wraps text and text fields in reports.

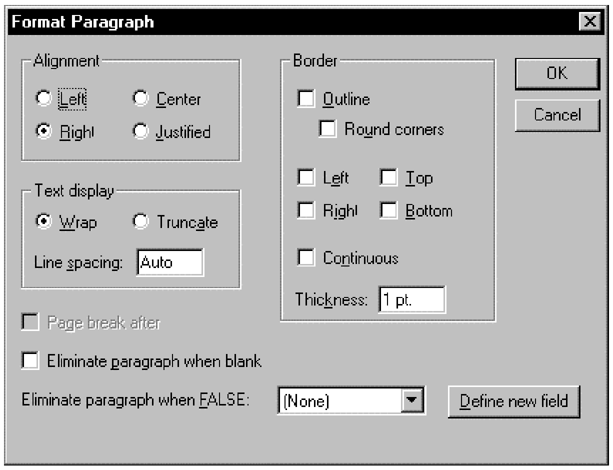
To format the display of text within your report definition:

1. Position the cursor in the paragraph you want to format.

   Note: Drag to select more than one paragraph. Alternatively, to select all paragraphs in a section or column, highlight the entire section or column.

2. Choose the Paragraph command from the Format menu.

   The Format Paragraph dialog box appears:



Format Paragraph

Alignment: Left Center [✓ Right] Justified

Text display: [✓ Wrap] Truncate

Line spacing: Auto

Border: Outline, Round corners, Left, Top, Right, Bottom, Continuous

Thickness: 1 pt.

Page break after

Eliminate paragraph when blank

Eliminate paragraph when FALSE: (None)

3. Select either the Wrap or Truncate option in the Text Display group box.

   Note: If you enter a value in the Line Spacing edit control, rather than using the default of Auto, the line spacing unit is in points.

4. Select any other paragraph formatting options you want to use.
5. Click OK.

Adding Borders to Paragraphs

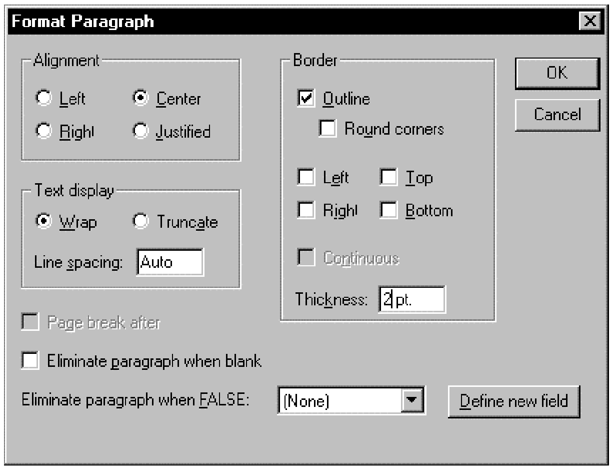
Paragraph borders are rectangular boxes around paragraphs. You can choose the width of the border and whether the top and bottom of a border end at column margins or extend to the center of the gutter (called *continuous borders*).

You can enclose part of a paragraph by specifying part of a border, such as its top or bottom; or you can place a single border around multiple paragraphs by turning on the Outline option in the Format Paragraph dialog box. The CA-Report Writer draws a border around all paragraphs until it reaches a paragraph for which the Outline option is not selected. You can also have a border with rounded corners.

To add a border to a paragraph within a report definition:

1. Position the cursor in the paragraph (or column or section) that you want to surround with a border.
2. Choose the Paragraph command from the Format menu.

   The Format Paragraph dialog box appears again.
3. Select the desired options in the Border group box. For example, select the Outline option and enter **2 pt.** in the Thickness edit control:



Alignment: ( ) Left [✓ Center] ( ) Right ( ) Justified
Text display: [✓ Wrap] ( ) Truncate
Line spacing: Auto
☐ Page break after
☐ Eliminate paragraph when blank
Eliminate paragraph when FALSE: (None)
Border: ☑ Outline ☐ Round corners ☐ Left ☐ Top ☐ Right ☐ Bottom ☐ Continuous
Thickness: 2 pt.

4. Click OK.

Changing Font, Font Size, Style, and Color

You can mix different fonts, font sizes, text styles, and text colors in reports.

Note: Point size refers to the height of the character. One point is approximately 1/72 of an inch.

To change your font settings for a report definition:

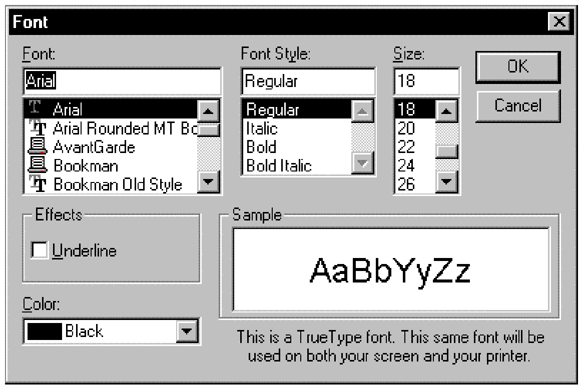
1. Select the characters you want to format.

   Alternatively, position the cursor at the point where you want to insert text using a new character format.

2. Click the Font toolbar button.

   Alternatively, choose the Character command from the Format menu.

   A standard Font dialog box appears:



Font

Font: Arial

Arial, Arial Rounded MT Bo, AvantGarde, Bookman, Bookman Old Style

Font Style: Regular

Regular, Italic, Bold, Bold Italic

Size: 18

18, 20, 22, 24, 26

Effects: Underline

Color: Black

Sample: AaBbYyZz

This is a TrueType font. This same font will be used on both your screen and your printer.

3. Select a font from the Font combo box.

4. Select a font style from the Font Style combo box.

   Alternatively, click the Bold, Italic, or Underline toolbar button to apply one of these styles to the selected text.

5. Select a font size from the Size combo box.

6. Select a color from the Color drop-down list box.

   Your selections are reflected in the Sample group box.

7. Click OK to return to the report definition.

Note: To print using a particular font, the font must be installed for the selected printer.

Creating a Report Title Page

You can print a separate report title page for a report.

Note: Your report must contain a report header section in order to print a title page.

To create a title page for your report:

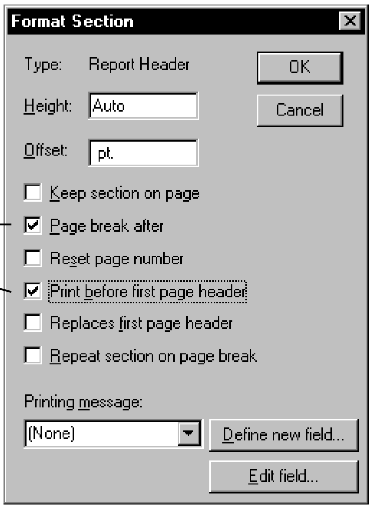
1. Position the cursor in the header section of the report definition.

   Alternatively, double-click the report header section in the section name window.

2. Choose the Section command from the Format menu.

   The Format Section dialog box appears.

3. Select the Page Break After and Print Before First Page Header options:



Type: Report Header

Height: Auto

Offset: pt.

- [ ] Keep section on page
- [x] Page break after
- [ ] Reset page number
- [x] Print before first page header
- [ ] Replaces first page header
- [ ] Repeat section on page break

Printing message: (None)

4. Optionally, select the Replaces First Page Header option *instead of* Print Before First Page Header if you want to define an alternate first page header.

5. Click OK to return to the report definition.

Controlling Page Breaks

You can control where page breaks occur in a report. For example, in a form letter, where the body of the report comprises the letter, you would want a page break after each body section.

You can control where page breaks occur by applying formatting to a section or by inserting a page break. You can also keep a section on the same page or keep sections together.

Page Break After a Section

To insert a page break after a body section within a report definition:

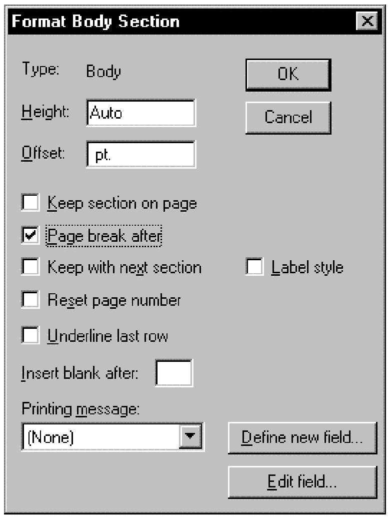
1. Position the cursor in the desired section of the report definition.

   Alternatively, double-click the section name window.

2. Choose the Section command from the Format menu.

   The Format Body Section dialog box appears.

3. Select the Page Break After option:


Format Body Section

Type: Body

Height: Auto

Offset: pt.

- [ ] Keep section on page
- [x] Page break after
- [ ] Keep with next section
- [ ] Label style
- [ ] Reset page number
- [ ] Underline last row

Insert blank after:

Printing message: (None)

4. Click OK.

Hard Page Breaks

To insert a hard page break within a report definition:

1. Position the cursor at the location where you want to insert a page break.
2. Choose the Insert Page Break command from the Insert menu.

   The page break appears as a dotted line on the screen.

Keeping a Section on a Page

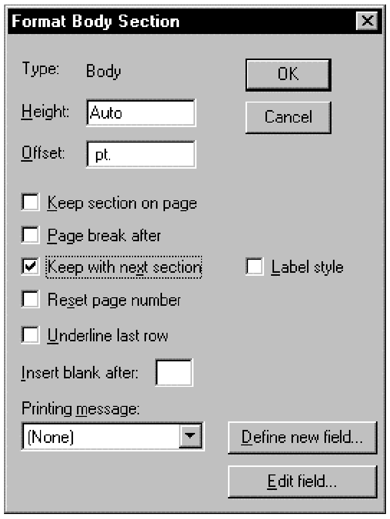
You can set an option to keep a section on the same page. If the entire section cannot fit on a page, the CA-Report Writer prints the section on a new page. (This attribute is always set for page header and page footer sections.)

To keep a section on a page:

1. Position the cursor in the desired section.

   Alternatively, double-click the section name window.
2. Choose the Section command from the Format menu.

   The Format Body Section dialog box appears again.
3. Select the Keep with Next Section option:



Height: Auto
Offset: pt.
☐ Keep section on page
☐ Page break after
☑ Keep with next section
☐ Label style
☐ Reset page number
☐ Underline last row
Insert blank after:
Printing message: (None)

4. Click OK.

Note: You cannot keep all body sections within a grouping together on the same page.

Keeping Sections Together

You can set an option to keep a group header section and at least one body section together in a report—this is useful when you want to keep a group heading with the body of the report. The CA-Report Writer prints a section on a new page if it cannot fit the following section on the current page. (Remember that every body section is a separate section.)

To keep sections together in your report:

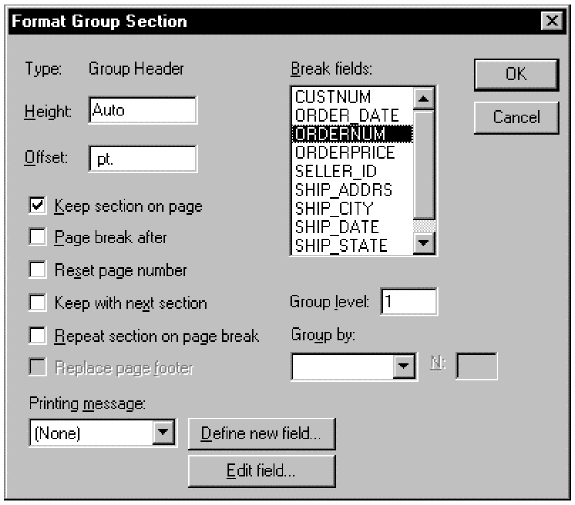
1. Position the cursor in a group header section in the report definition.

   Alternatively, double-click the group header section name window.

2. Choose Section from the Format menu.

   The Format Group Section dialog box appears.

3. Select the Keep Section on Page option:



4. Click OK.

Paginating Cross Tabular Reports

The CA-Report Writer determines the width (the number of columns) of a cross tabular report when it prints the report. The width of the report depends on the data. The CA-Report Writer uses horizontal pagination for reports that are wider than the defined page margins.

Using Conditions to Determine Report Data

You can include or exclude information from a report based on data coming from your database or computed *conditions*. For example, in an accounts receivable report, you may want to display a bold message next to any customer owing more than $10,000 or you may want to print a subreport detailing some information based on a set of conditions.

You can also have the CA-Report Writer eliminate individual fields or rows of a report if the value of one field is false.

To set report data conditions within a report definition:

1. Position the cursor in the paragraph to which you want to apply the condition.

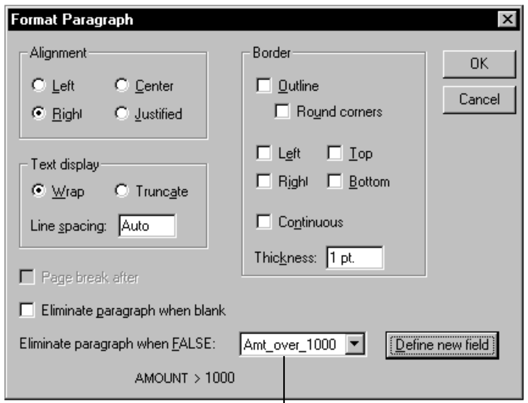
   Drag to select more than one paragraph.

   Alternatively, to select all the paragraphs in a column or section, highlight the column or section.

2. Choose Paragraph from the Format menu.

   The Format Paragraph dialog box appears.

3. Select a value from the Eliminate Paragraph When FALSE drop-down list box:



Format Paragraph

Alignment: ( ) Left ( ) Center (●) Right ( ) Justified

Text display: (●) Wrap ( ) Truncate

Line spacing: Auto

☐ Page break after

☐ Eliminate paragraph when blank

Eliminate paragraph when FALSE: Amt_over_1000

AMOUNT > 1000

Border: ☐ Outline ☐ Round corners ☐ Left ☐ Top ☐ Right ☐ Bottom ☐ Continuous

Thickness: 1 pt.

Report removes all rows with an amount value under 1000

4. Click OK.

   Note: To make the printing of an entire row conditional, you must make all paragraphs in a row conditional.

Using a Subreport in a Conditional Paragraph

You can insert a subreport in a conditional paragraph in a report. The subreport prints **only** if the condition in the paragraph is true, allowing you to use alternate subreports, depending on parameters or data in the main report.

See the online help system for details about using subreports.

Determining Page Layout

In the CA-Report Writer you can customize your report's page layout by:

- Setting page size and margins
- Snaking columns on a page
- Using horizontal pagination

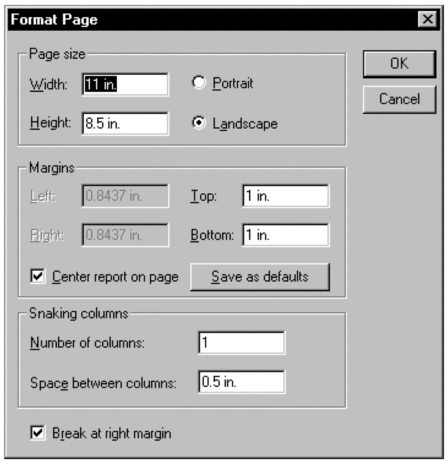
Setting Page Size, Orientation, and Margins

You can easily change the size of a report page, change the page orientation, and adjust the page margins.

To change the report page size and orientation:

1. Choose the Page command from the Format menu.

   The Format Page dialog box appears:

   Format Page

   Page size
   Width: 11 in.
   Height: 8.5 in.
   Portrait [✓ Landscape]

   Margins
   Left: 0.8437 in.
   Top: 1 in.
   Right: 0.8437 in.
   Bottom: 1 in.
   ☑ Center report on page

   Snaking columns
   Number of columns: 1
   Space between columns: 0.5 in.

   ☑ Break at right margin

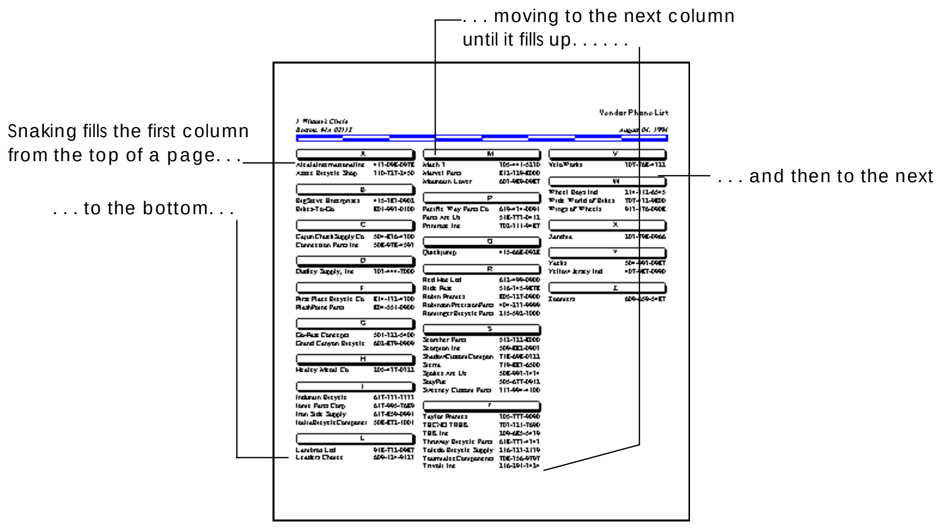
2. Enter values in the Width and Height edit controls.
3. Select either the Portrait or Landscape option to define the page orientation.
4. To change the page margins, enter values for Left, Right, Top, and Bottom in the Margins group box.

   Note: You can also change page margins for the report from the horizontal ruler by dragging the left or right margin markers (the solid black triangles) to the desired ruler location.

5. To save your page formatting selections as defaults, click the Save As Defaults button.
6. Click OK.

Snaking Columns on a Page

You can divide a page into a number of equal width columns and *snake* the contents of a report within those columns. Snaking means filling the first column from the top of a page to the bottom, moving to the next column until it fills up. The column contents are listed from top to bottom in each increment. This is particularly useful for a telephone directory, for example:


. . . moving to the next column until it fills up. . . . . .
Snaking fills the first column from the top of a page. . .
. . . to the bottom. . .
. . . and then to the next

To snake columns on a page in your report:

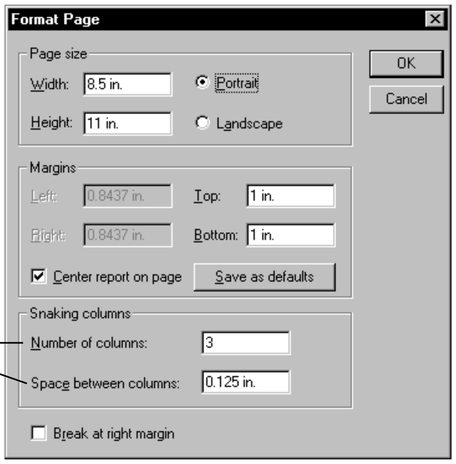
1. Choose the Page command from the Format menu.

   The Format Page dialog box appears.

2. Enter a value in the Number of Columns edit control to specify the desired number of columns into which you want to divide the report.

3. Enter a value in the Space Between Columns edit control to specify the amount of space that should be placed between columns. For example:

Thisreportisdivided intothreecolumns, 0.125in.apart



Page size
Width: 8.5 in.
Height: 11 in.
(●) Portrait
( ) Landscape

Margins
Left: 0.8437 in.
Top: 1 in.
Right: 0.8437 in.
Bottom: 1 in.
[✓] Center report on page

Snaking columns
Number of columns: 3
Space between columns: 0.125 in.

[ ] Break at right margin

4. Click OK.

Using Horizontal Pagination

If a report is wider than its page size, you can print the overflow information on subsequent pages using *horizontal pagination*. Horizontal pagination lets the report print across first and then down. It ensures that each cell fits on the page or it is forced to the next horizontal page. This feature is useful in cross tabular reports, where the width of the report can vary.

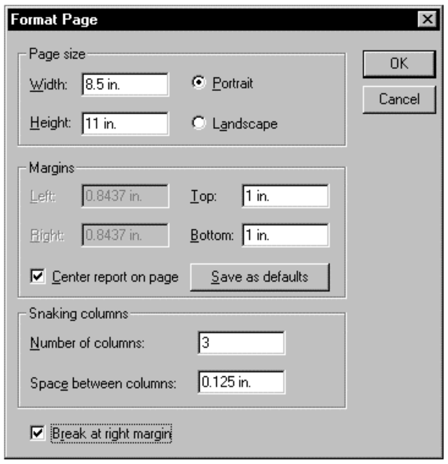
Note: If a report is wider than its page size and you do not define a horizontal page break, the columns are truncated.

To use the horizontal pagination feature:

1. Choose the Page command from the Format menu.

   The Format Page dialog box appears again.

2. Select the Break At Right Margin option:



Page size
Width: 8.5 in. [✓ Portrait]
Height: 11 in. Landscape

Margins
Left: 0.8437 in. Top: 1 in.
Right: 0.8437 in. Bottom: 1 in.
☑ Center report on page

Snaking columns
Number of columns: 3
Space between columns: 0.125 in.

☑ Break at right margin

3. Click OK.

The CA-Report Writer creates a horizontal page break when the report table columns in the report extend beyond the specified right margin.

Adding Graphics

You can add graphics to your reports by drawing lines and rectangles or by importing graphic images that were created in other applications.

Lines and Rectangles

Add lines and rectangles to a report to highlight report text. For example, you can draw a line after a group footer to print a line before each new group of data.

Lines and rectangles are useful when the report contents do not vary with the data. In other cases, you should use paragraph borders or text underlining, since they automatically adjust to the size of the paragraph or text.

Graphic Images

Insert files containing logos, designs, photos, or other pictures in a report definition to enhance report data. For example, you can create a letterhead by adding a logo to the page header. In addition, you can specify graphic images for data fields and computed fields so you can print a graphic image for every occurrence of a specific field in a report.

Note: You should place lines, rectangles, and graphic images within the boundaries of a single section only. If you want a graphic displayed across an entire report page, insert it into the background section.

Drawing Lines

To draw a line in any section of a report definition:

1. Select the Insert Line toolbar button.

   Alternatively, choose the Line command from the Insert menu.
2. Position the cursor where the line is to start and drag to create the line.

Note: The line must stay in one report section.

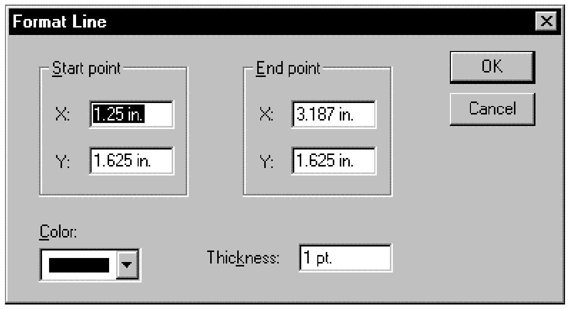
Changing a Line’s Color and Thickness

To modify a line in your report definition:

1. Double-click the desired line.

   Alternatively, choose the Line command from the Format menu.

   The Format Line dialog box appears.

   Start point
   X: 1.25 in.
   Y: 1.625 in.

   End point
   X: 3.187 in.
   Y: 1.625 in.

   Color: ■

   Thickness: 1 pt.

2. Select a color from the Color drop-down list box.
3. Enter a value in the Thickness edit control to change the line thickness.
4. Click OK.

Moving a Line

To modify a line in your report definition, use one of the following methods:

Drag Method	Position the cursor anywhere on the line except on a selection handle and drag the line to a new location.
Dialog Box Method	To use the dialog box method:

1. Double-click the desired line.

   Alternatively, choose the Line command from the Format menu.

   The Format Line dialog box appears again.

2. Enter values in the X and Y edit controls to specify the new position.
3. Click OK.

Resizing a Line

To resize a line in your report definition, use one of the following methods:

Drag Method

Drag a selection handle on one end of the line to a new position.

Dialog Box Method

To use the dialog box method:

1. Double-click the desired line.

   Alternatively, choose the Line command from the Format menu.

   The Format Line dialog box appears again.
2. Enter values in the X and Y edit controls in the Start Point and End Point group boxes.
3. Click OK.

Drawing Rectangles

You can draw rectangles in any section of a report definition. You can apply rounded corners and shadows to rectangles. You can also fill the background and foreground of a rectangle with various colors and patterns.

To draw a rectangle in your report definition:

1. Select the Insert Rectangle toolbar button.

   Alternatively, choose Rectangle from the Insert menu.
2. Position the cursor where you want the line to start and drag to create the rectangle.

Note: The rectangle must fit in one report section.

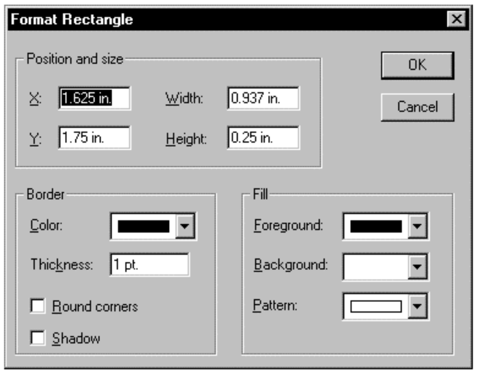
Changing a Rectangle's Color, Fill, and Border

To modify a rectangle within a report definition:

1. Double-click the desired rectangle.

   Alternatively, choose the Rectangle command from the Format menu.

   The Format Rectangle dialog box appears:



Format Rectangle

Position and size

X: 1.625 in.
Width: 0.937 in.
Y: 1.75 in.
Height: 0.25 in.

Border

Color:
Thickness: 1 pt.
Round corners
Shadow

Fill

Foreground:
Background:
Pattern:

2. Select a color from the Color drop-down list box.
3. Enter a value in the Thickness edit control to change the line thickness.
4. Select the Round Corners option to add rounded corners to the rectangle.
5. Select the Shadow option to add a shadow to the rectangle.
6. Select the desired foreground fill from the Foreground drop-down list box.
7. Select the desired background fill from the Background drop-down list box.
8. Select a fill pattern from the Pattern drop-down list.
9. Click OK.

Moving a Rectangle

To move a rectangle in your report definition, use one of the following methods:

Drag Method

Position the cursor anywhere on the rectangle except on a selection handle and drag the rectangle to a new location.

Dialog Box Method

To use the dialog box method:

1. Double-click the desired rectangle.

   Alternatively, choose the Rectangle command from the Format menu.

   The Format Rectangle dialog box appears again.

2. Enter values in the X and Y edit controls to specify the new position.
3. Click OK.

Resizing a Rectangle

To resize a rectangle in your report definition, use one of the following methods:

Drag Method

Drag a selection handle on a rectangle to a new position.

Dialog Box Method

To use the dialog box method:

1. Double-click the rectangle.

   Alternatively, choose the Rectangle command from the Format menu.

   The Format Rectangle dialog box appears again.

2. Enter values in the Height and Width edit controls.
3. Click OK.

Inserting Graphic Images in a Report

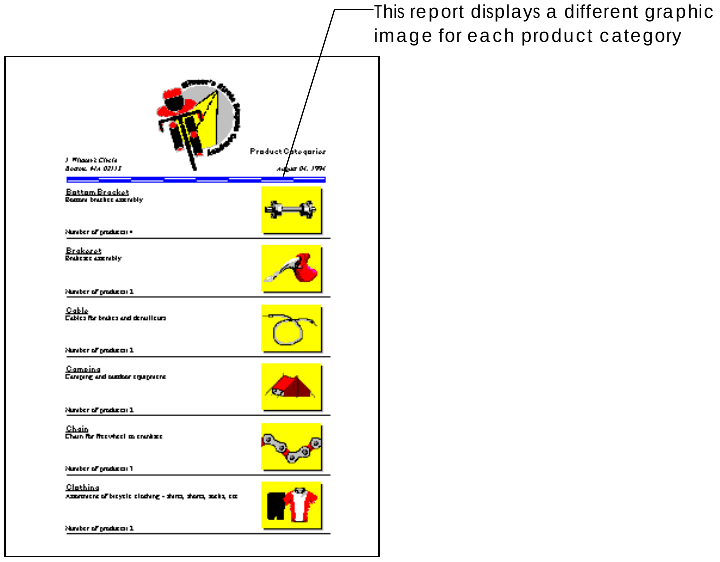
You can insert graphic images into any section of a report definition, but first you must specify the following:

- Where in the report you want the graphic image to appear
- Where the graphic image comes from
- How the graphic should be sized in the report

Graphics, like other report contents, appear in a report in accordance with section type. For example, a graphic inserted in the report header prints once at the top of the report, whereas a graphic inserted in the body section prints once for each row of data. Graphics inserted into the background section appear on each report page, creating a constant backdrop, such as on a form.

You can insert a graphic using a data field or computed field whose value is the file name of the graphic. Use this feature if you want to associate a graphic image with specific field values.

The following report illustrates this feature:


This report displays a different graphic image for each product category

Product Categories

1 Wheel Circle
Boston, MA 02111

August 04, 1994

Bottom Bracket
Bottom bracket assembly

Number of products: 4

Brakeset
Brakeset assembly

Number of products: 1

Cable
Cables for brakes and derailleurs

Number of products: 1

Camping
Camping and outdoor equipment

Number of products: 1

Chain
Chain for freewheel to crankset

Number of products: 1

Clothing
Assortment of bicycle clothing - shirts, shorts, socks, etc

Number of products: 1

You can insert a graphic in any section except a page header or page footer. You can import the following types of graphics files into a report:

- Windows bitmaps (.BMP)
- CompuServe (.GIF)

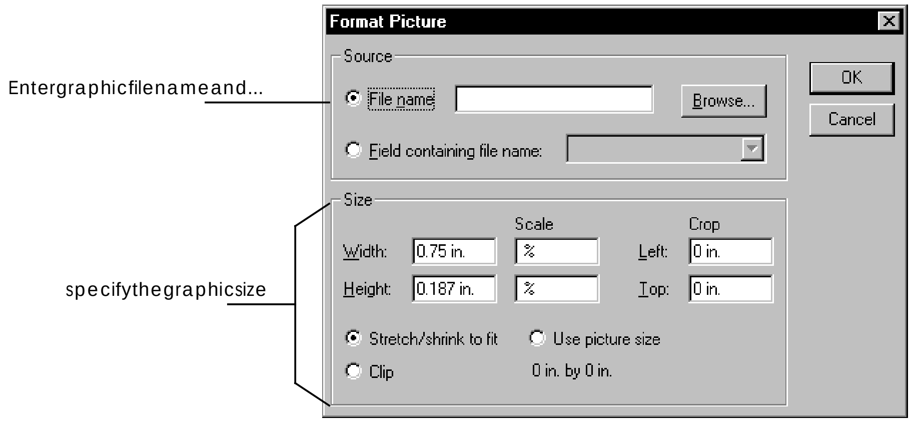
Tip: Use .BMP files for best performance results.

To insert a graphic in your report definition:

1. Select the Insert Picture toolbar button.

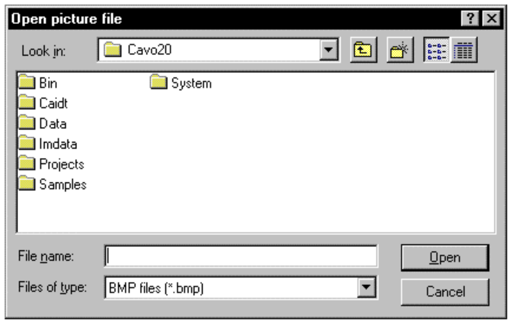
   Alternatively, choose the Picture command from the Insert menu.
2. Begin dragging in the report definition to specify where you want the start point (any corner of the graphic) to be located.
3. Drag until the rectangle (representing the location and dimensions of the graphic) is the size you want.

When you release the mouse button, the Format Picture dialog box appears:



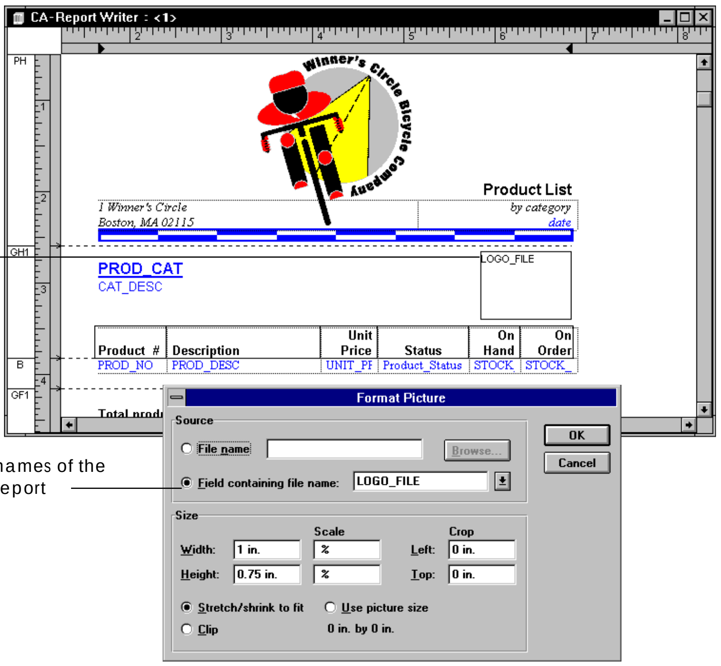
4. Do one of the following

   - Select the File Name radio button, and then enter the name and directory path of the graphics file to insert.
   - Choose Browse to display the Open Picture File dialog box from which you can locate and select the file to insert:



The CA-Report Writer displays the original dimensions of the graphic in the lower-right corner of the Format Picture dialog box.

- To associate a picture with a particular field, select the Field Containing File Name radio button, and then select the file name that contains the picture you want to insert from the drop-down list box. For example:



Format Picture

Source
- ○ File name:
- ◉ Field containing file name: LOGO_FILE

Size

		Scale		Crop
Width:	1 in.	%	Left:	0 in.
Height:	0.75 in.	%	Top:	0 in.

- ◉ Stretch/shrink to fit
- ○ Use picture size
- ○ Clip
- 0 in. by 0 in.

The group header contains an inserted graphic image

LOGO_FILE contains the file names of the graphics that will print in the report

5. Indicate how you want the graphic inserted into the report definition by selecting one of the following options:

   - Select Stretch / Shrink To Fit if you want the CA-Report Writer to expand or reduce the size of the graphic proportionally to fit in the space you specified for it.
   - Select Clip if you want the CA-Report Writer to crop the graphic to fit in the space you allotted for it.
   - Select Use Picture Size to use the original size of the graphic.

   Note: If you are using the Stretch / Shrink To Fit or Clip option, you can change the dimensions of the graphic by entering either the actual dimensions or the percentage by which to scale the picture using the Width and Height or the Scale edit controls in the dialog box. In conjunction with these options, you can also indicate the amount by which to crop the graphic from the left or top margin.

6. Click OK to return to the report definition.

Tip: For best results, consider sizing the image in your application and then choosing the Use Picture Size Option in the Format Picture dialog box.

Moving a Graphic

To move a graphic, simply drag the graphic to its new location in the report definition.

You can also move a graphic from one section to another using the Cut and Paste toolbar buttons or menu commands.

Resizing a Graphic

To resize a graphic, use one of the following methods:

Drag Method

Drag a selection handle at one corner of the picture.

Dialog Box Method

To use the dialog box method:

1. Double-click on a picture.

   Alternatively, choose the Picture command from the Format menu.

   The Format Picture dialog box appears again.

2. Enter new values in the Width and Height edit controls.

   Alternatively, enter new values in the Scale edit controls.

3. Click OK to return to the report definition.

Moving Graphics to the Background

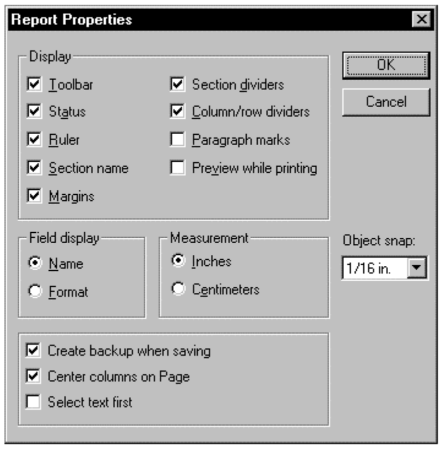
If you are editing an area of a report definition in which one or more graphics are overlaid on text, you can move the graphics to the background so that you can select and edit the text.

To move a graphic to the background:

1. Choose the Properties command from the File menu.

   The Report Properties dialog box appears.

2. Deselect the Select Text First option:



Display
- [✓] Toolbar
- [✓] Status
- [✓] Ruler
- [✓] Section name
- [✓] Margins
- [✓] Section dividers
- [✓] Column/row dividers
- [ ] Paragraph marks
- [ ] Preview while printing

Field display: [✓ Name] Format

Measurement: [✓ Inches] Centimeters

Object snap: 1/16 in.

- [✓] Create backup when saving
- [✓] Center columns on Page
- [ ] Select text first

Note: You cannot select the graphic in the background until you turn this option off.

3. Click OK.

Working with Overlapping Graphics

When a section contains more than one graphic object (line, rectangle, or image), you can use the Move To Back command (Format menu) to control the order in which the graphics are drawn relative to one another. The order is important *only* when graphics overlap.

The CA-Report Writer draws graphics from front to back. Each time you select a graphic and choose Move To Back, the selected graphic is moved one “step” backward in the order.

Note: Text is always drawn after all graphics. You cannot use the Move To Back option to change the drawing order relative to the text.

Deleting a Line, Rectangle, or Graphic

To delete graphic elements from your report definition:

1. Select a line, rectangle, or picture in the report definition.
2. Choose the Delete command from the Edit menu.

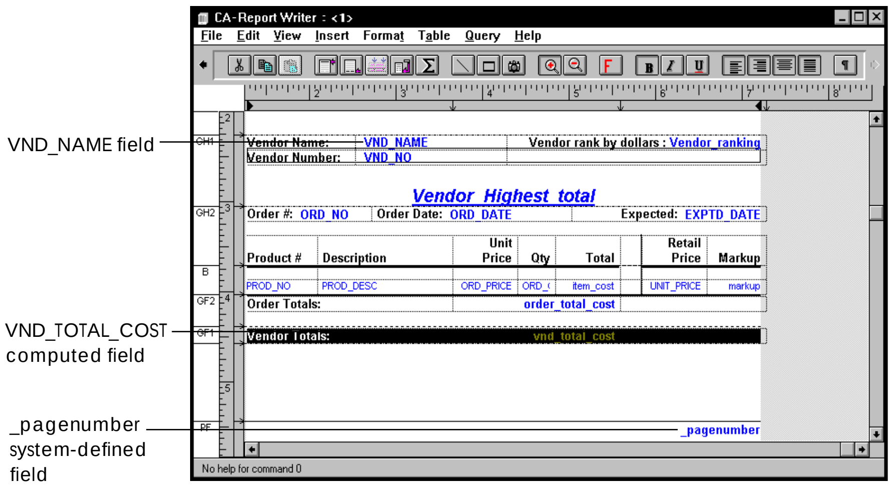
Using Report Fields

This section explains how to define and use fields in reports. To display data in a report, you insert report fields that define the data to the report definition. For example, inserting fields into the body section causes the report to print the field values once for each row in the query result.

Types of Fields

The CA-Report Writer recognizes four types of report fields—*data*, *computed*, *system-defined*, and *parameter*.

In the following illustration, the VND_NAME field is a data field in a query associated with the report definition; the VND_TOTAL_COST field is a computed field that totals the cost of each item in the order; and the _pagenumber field is a system-defined field that displays a page number in the footer of each report page:


VND_NAME field
VND_TOTAL_COST computed field
_pagenumber system-defined field
CA-Report Writer : <1>
File Edit View Insert Format Table Query Help
Vendor Name: VND_NAME
Vendor rank by dollars : Vendor_ranking
Vendor Number: VND_NO
Vendor_Highest_total
Order #: ORD_NO
Order Date: ORD_DATE
Expected: EXPTD_DATE
Product # Description Unit Price Qty Total Retail Price Markup
PROD_NO PROD_DESC ORD_PRICE ORD_(item_cost UNIT_PRICE markup
Order Totals: order_total_cost
Vendor Totals: vnd_total_cost
_pagenumber
No help for command 0

Data Fields

A *data field* maps to a field in the query result that you use to supply data for the report.

Computed Fields

A *computed field* uses functions and arithmetic operators to compute a value. You define a computed field by combining values in other report fields or by using functions. A common type of computed field is the Sum field, which totals the values of a specified report field.

System-Defined Fields

A *system-defined field* supplies system data such as the date and time (_date) or page number (_pagenumber).

Parameter Fields

A *parameter field* prompts the user to enter a value each time the report is run. You define the prompt and the data type for the parameter field. You can also define a parameter field in a subreport that you use to pass values from a main report or bucket table. Subreports and bucket tables are described fully in the online help system.

Steps for creating computed and parameter fields are described in detail later in this chapter.

Data Types

The CA-Report Writer recognizes and assigns three data types: *text*, *date*, and *number*. For data fields, it uses the data types of data in the database table to determine what data type to assign. For computed fields, the CA-Report Writer automatically assigns the data type based on the field definition.

The following table describes each data type:

Data Type	Description
Text	Character strings Character strings cannot contain null values; the maximum length is 65K.
Number	Decimal and integer numbers The range is 1.7e +/- 308 with 15 digits of accuracy (double floating point).
Date	Date and time values The acceptable range is Jan. 1, 0001–Dec. 31, 9999.

An example of each data type is shown in the following table:

Data Type	Field	Sample Data
Text	VND_NAME	Pacific Way Parts Co.
Date	ORD_DATE	8/4/93
Number	ORD_PRICE	89.95

A variety of standard format strings are available for each data type. You can choose one of these or create a format using format characters. For more information, see Formatting Fields later on in this chapter.

Viewing and Editing Field Definitions

Before you define computed fields or parameter fields for your report definition, your report contains data fields from the query associated with the report and system-defined fields, such as _date and _pagenumber.

If a field is a data field, you can view its definition and edit its appearance in the report; if a field is any other type of field, you can also edit its definition.

To view or edit a field definition, do one of the following:

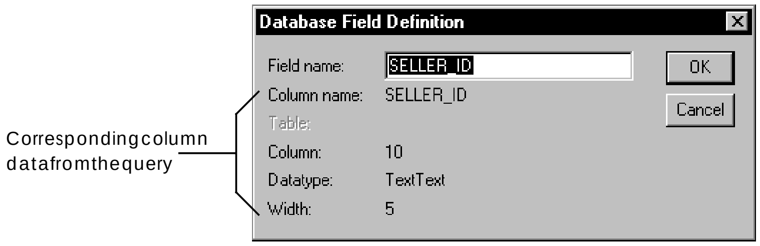
- Choose the Field Definitions command from the Edit menu, and then select a field.
- Double-click on a field in the report definition.
- Position the cursor on a field in the report definition and choose the Field command from the Format menu.

The CA-Report Writer responds in one of several ways depending on the type of field you selected:

Data Fields

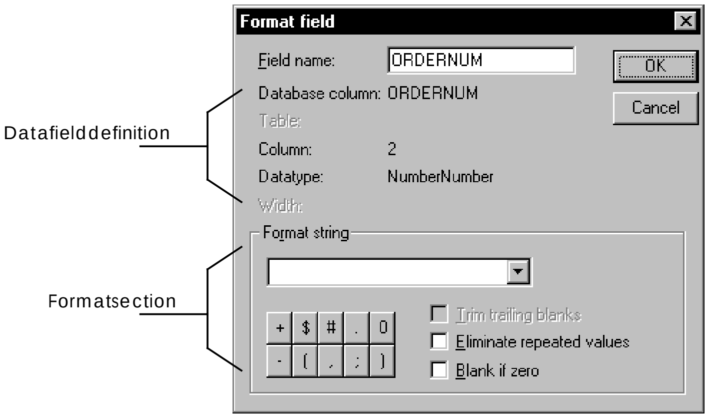
If you chose Field Definitions from the Edit menu and selected a data field to edit, the CA-Report Writer displays the Database Field Definition dialog box where you can view the definition.

This dialog box displays the column name, table, column, data type, and width from the corresponding column in the query:


Correspondingcolumn datafromthequery
Database Field Definition
OK
Cancel

Field name: SELLER_ID
Column name: SELLER_ID
Table:
Column: 10
Datatype: TextText
Width: 5

If you double-clicked a data field or chose Field from the Format menu, the CA-Report Writer displays the Format Field dialog box where you can view the definition and edit its format:



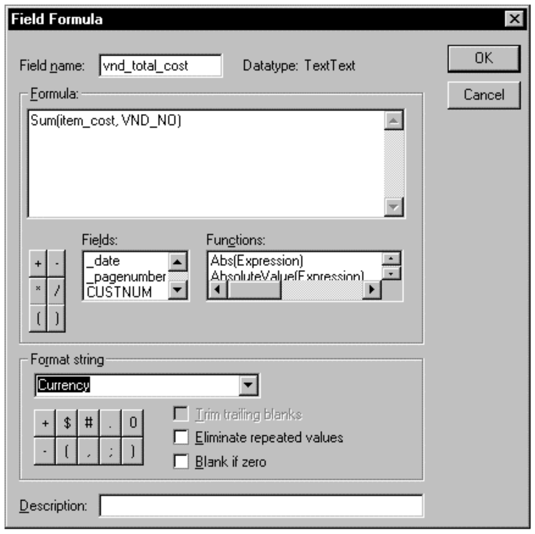
Field name: ORDERNUM
Database column: ORDERNUM
Table:
Column: 2
Datatype: NumberNumber
Width:

Computed and System-Defined Fields

If you selected a computed field or system-defined field, the CA-Report Writer displays the Field Formula dialog box, shown below.

This dialog box displays the formula that defines the selected field and its formatting options. Use the dialog box options to edit a field's formula, apply a format to the field, and assign other display options.

In the following example, the VND_TOTAL_COST field totals the cost of each item carried by individual vendors:



Field name: vnd_total_cost Datatype: TextText
Formula: Sum(item_cost, VND_NO)
Format string: Currency
Description:

You can edit the field's formula in one of two ways:

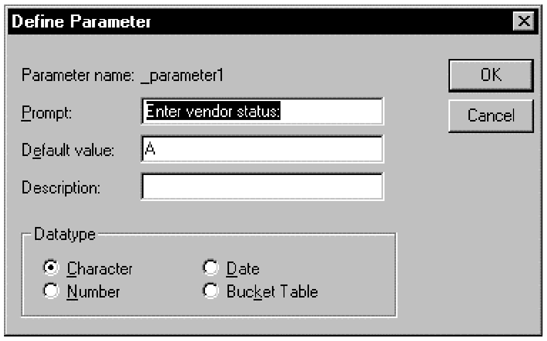
- Edit the formula manually in the Formula multi-line edit control.
- Select the field from the Fields list box, and then double-click a function in the Functions list box.

You can optionally enter a field description in the Description edit control.

For more information about formatting fields in reports and using field formulas, see Formatting Fields and Defining Computed Fields later in this chapter.

Parameter Fields

If you select a parameter field, the CA-Report Writer displays the Define Parameter dialog box, shown below. This dialog box displays the prompt displayed to the user when CA-Report Writer runs the report, the parameter's default value, a description (if any), and the data type.



Parameter name: _parameter1
Prompt: Enter vendor status:
Default value: A
Description:
Datatype: (● Character) ○ Date ○ Number ○ Bucket Table

Tip: You can also edit the definition of a parameter field by selecting the Edit Report Parameters menu command.

For more information about defining field parameters in reports, see Using Parameter Fields later in this chapter.

Inserting, Moving, and Deleting Fields

With some restrictions, you can insert fields into any section in a report definition. If you decide later that you want to move a field or delete it, you can use commands in the Edit menu or, if a field does not appear in a report definition, use other procedures described in this section.

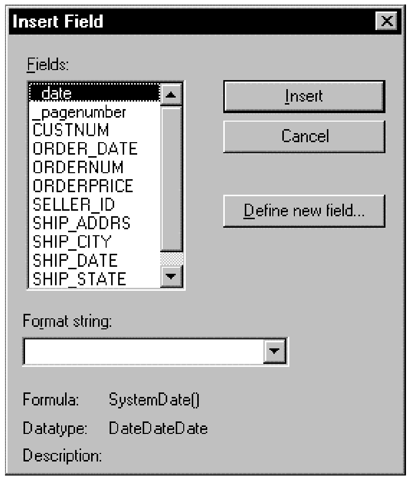
Inserting a Field in a Report Definition

To insert a field in your report definition:

1. Move the cursor to the location where you want to insert a new field.
2. Click the Field toolbar button.

   Alternatively, choose the Field command from the Insert menu.

The Insert Field dialog box appears:



_date
_pagenumber
CUSTNUM
ORDER_DATE
ORDERNUM
ORDERPRICE
SELLER_ID
SHIP_ADDRS
SHIP_CITY
SHIP_DATE
SHIP_STATE

Formula: SystemDate()
Datatype: DateDateDate
Description:

3. From the Fields list box, select a field type name.
4. Optionally, choose a format string from the Format String drop-down list box.
5. Click the Insert button to display the new field in the report definition.

Moving a Field

To move a field within your report definition:

1. Select the field you want to move.
2. Click the Cut toolbar button.

   Alternatively, choose the Cut command from the Edit menu.
3. Move the cursor to the location where you want to place the cut field.
4. Click the Paste toolbar button.

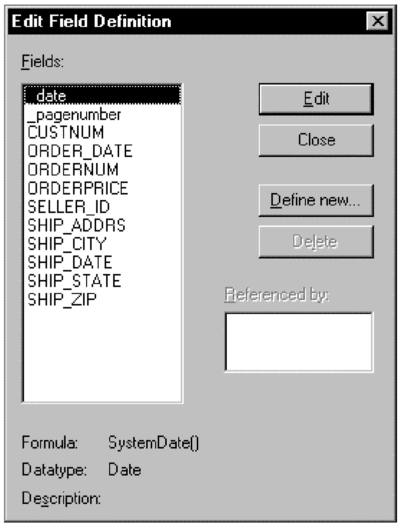
   Alternatively, choose the Paste command from the Edit menu.

Deleting a Field

When you delete a field from a report definition, you must also delete the field definition. To do so, place the cursor on the field you want to delete and do one of the following:

- Choose Field Definitions from the Edit menu.
- Alternatively, choose the Delete command from the Edit menu.

The Edit Field Definition dialog box appears:



_date
_pagenumber
CUSTNUM
ORDER_DATE
ORDERNUM
ORDERPRICE
SELLER_ID
SHIP_ADDRS
SHIP_CITY
SHIP_DATE
SHIP_STATE
SHIP_ZIP

Formula: SystemDate()
Datatype: Date
Description:

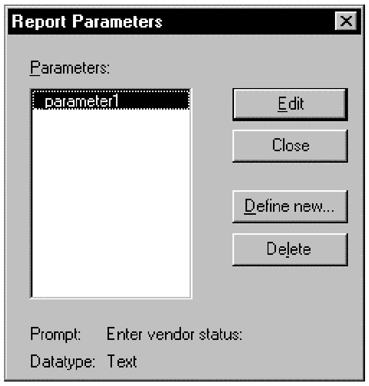
Select the name of the field you just deleted from the report definition, click Delete, and then click Close.

Deleting Parameter Fields

To delete a parameter field, do one of the following:

- Choose Report Parameters from the Edit menu.

  The Report Parameters dialog box appears:



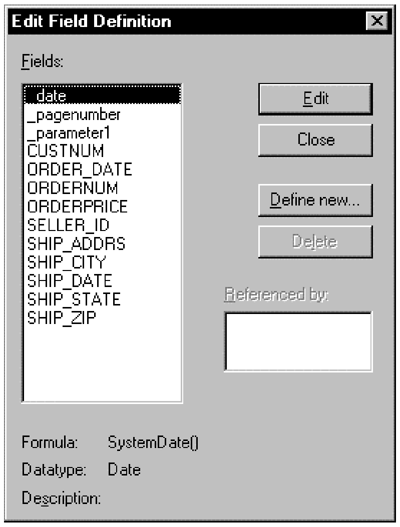
parameter1

Prompt: Enter vendor status:
Datatype: Text

  Select a parameter from the Parameters list box, click Delete, and then click Close.

- Choose Field Definitions from the Edit menu.

The Edit Field Definition dialog box appears:



_date
_pagenumber
_parameter1
CUSTNUM
ORDER_DATE
ORDERNUM
ORDERPRICE
SELLER_ID
SHIP_ADDRS
SHIP_CITY
SHIP_DATE
SHIP_STATE
SHIP_ZIP

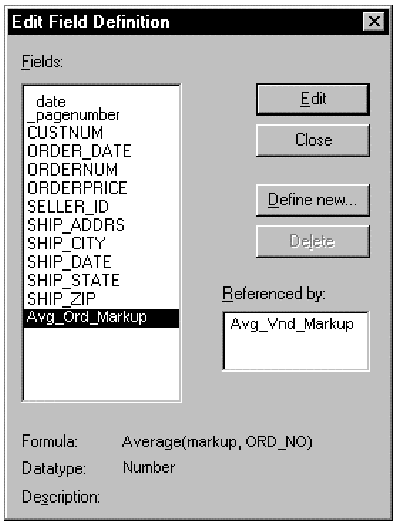
Formula: SystemDate()
Datatype: Date
Description:

Select a field from the Fields list box, click Delete, and then click Close.

Restrictions

You cannot delete a computed field if it is referenced by any other computed field, as displayed in the Referenced By list box in the Edit Field definitions dialog box. Similarly, you cannot delete a parameter field if it maps to a field in a query. The CA-Report Writer dims the Delete button if the field cannot be deleted.

For example, the Delete button in the following illustration is dimmed because the selected field, Avg_Ord_Markup, is used in the field definition of Avg_Vnd_Markup:



_date
_pagenumber
CUSTNUM
ORDER_DATE
ORDERNUM
ORDERPRICE
SELLER_ID
SHIP_ADDRS
SHIP_CITY
SHIP_DATE
SHIP_STATE
SHIP_ZIP
Avg_Ord_Markup

Referenced by: Avg_Vnd_Markup

Formula: Average(markup, ORD_NO)
Datatype: Number
Description:

Defining Computed Fields

Computed fields are user-defined fields that you can create using other fields, arithmetic operators, and functions to compute values. For example, the definition of a simple computed field, UNIT_SALES, is the product of two other fields: PRICE*QUANTITY.

Field Names

By default, CA-Report Writer supplies generic field names (FIELD1, FIELD2, etc.) for computed fields, which you can replace with something more meaningful. Field names must begin with a letter; subsequent characters in the field name can be any combination of upper and lowercase letters, numbers, and/or the underscore (_) character.

Arithmetic Operators

The CA-Report Writer uses the operators shown in the following table to combine field values arithmetically:

Operator	Function	Example
+	Addition	PRICE+10
-	Subtraction	QUANTITY-4
*	Multiplication	PRICE*QUANTITY
/	Division	Total/QUANTITY

CA-Report Writer observes the standard rules of precedence:

- Operations involving multiplication and division are computed first, going from left to right.
- Operations involving addition and subtraction are computed after multiplication and division operations.

For example, the result of the following formula is 15:

```
3 + 4 / 2 * 6
```

You can use parentheses in an expression to override the rules of precedence and to add clarity to the formula's display.
CA-Report Writer computes the operations in the innermost parentheses first and works outward.

For example, the result of this formula is 3 1/3:

```
3 + (4 / (2 * 6))
```

Defining a Simple Computed Field

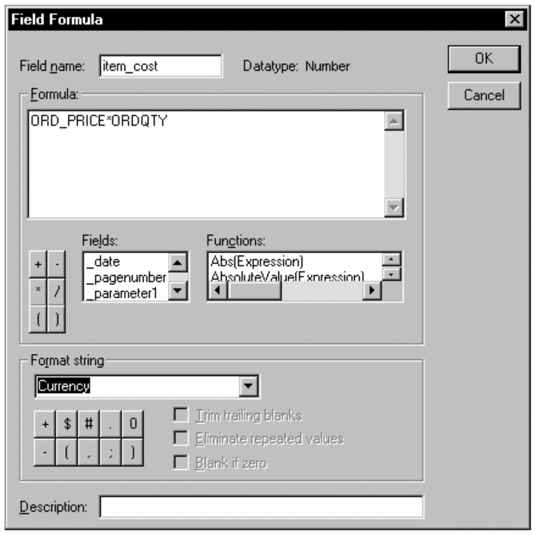
You define computed fields by either entering the field formula from your keyboard or using a point-and-click technique to build the field definition.

To define a computed field for your report definition:

1. Click the Insert Field toolbar button, or choose Field from the Insert menu and then click the Define New Field button in the Insert Field dialog box that appears.

   Alternatively, choose Field Definitions from the Edit menu and then click the Define New button in the Edit Field Definition dialog box that appears.

   In all cases, the Field Formula dialog box appears.
2. Optionally, enter a field name in the Field Name edit control (or accept the default).
3. Position the cursor in the Formula text box, and then do one of the following:
   - Define the field by entering a formula in the Formula multi-line edit control. For example:

     Field Formula

     Field name: item_cost

     Datatype: Number

     Formula: ORD_PRICE*ORDQTY

     Fields: _date, _pagenumber, _parameter1

     Functions: Abs(Expression), AbsoluteValue(Expression)

     Format string: Currency

     Description:

   - Double-click a field in the Fields list box and select the desired arithmetic operator buttons to build the formula.
4. Optionally, format the field’s appearance using the Format String group box options.
5. Optionally, enter a description for the field in the Description edit control.
6. Click OK.

Using Functions to Define a Computed Field

The CA-Report Writer supplies over 80 functions that you can use to create computed fields. The functions perform sophisticated calculations. For example, you can use functions to rank field values or to return the future value of an investment.

When you use a function in a computed field, you must specify certain variables on which the function operates. For example, to use the function Sum(*expression*, *section*), you must specify the expression (a data field or computed field) and the section over which to compute the sum (the group section or the entire report). The VND_TOTAL_COST field in ORD.RET uses this sum function:

```
Sum(item_cost,VND_NO)
```

where ITEM_COST is the field to total within a group that has VND_NO defined as its break field.

Types of Functions

The following information is a summary of the functions you can use in the CA-Report Writer. For detailed information about these functions, see the CA-Report Writer online help system.

Aggregate Functions

Aggregate functions perform calculations (for example, counting, summing, and averaging) on grouped data or data for the entire report. Aggregate functions ignore Null values in their computations. For example, the Avg() function does not include the Null value when it computes the average.

Generally, you insert a computed field containing an aggregate function in a group header or footer or report header or footer. The placement of the field determines the value returned by the function.

For example, if you want to count the number of items per order, place the Count aggregate function in a group section defined by the ORD_NO break field. If, instead, you want to count the number of items in the entire report, insert the field in a report header or footer section. See the online help system for the individual aggregate functions and for details on where to place them in a report definition.

Note: If your report contains a large amount of data, you can improve performance and memory usage by placing aggregate functions in either report footers or group footers. Placing an aggregate function in a report header requires the most time and memory to evaluate.

Arithmetic Functions

Arithmetic functions perform common mathematical operations on data defined in the report definition, such as raising a value to a power or returning an absolute value.

Conditional Functions

Conditional functions perform common relational operations on data defined in the report definition. Conditional functions take relational expressions as arguments that return a true (1) or false (0) result. For example, you can test if a value falls within a range or is null.

Conversion Functions

Conversion functions convert a field or value of one data type to a value of another data type. For example, CA-Report Writer provides functions that convert date values to number values or text values, number values to date value or text values, and so on.

Date and Time Functions

Date and Time functions calculate dates and times using serial numbers. For example, you can use a function to add a number of weeks to a date and get the future date back.

Financial Functions

Financial functions calculate common financial formulas. For example, you can use financial functions to calculate the payment on a loan or the future value of an investment.

Financial functions usually contain parameters such as a payment amount, a number of payment periods, and an interest rate. Be sure that each number you supply is based on the same unit of time (for example, months or years).

For example, if the payment period is monthly, but you have an annual interest rate, divide the interest rate by twelve, as shown in the example below where the annual interest rate is 8%:

```
pmt(0.08/12,30,10000,0,0)
```

Lookup Functions

Lookup functions search tables of data (that is, data arranged in multiple columns and rows) to locate a specified value. They help you locate specific information, which is provided as part of a large package of data (for example, tax tables, mortgage payment tables, zip code tables, pay scale tables, and so on).

String Manipulation Functions

String manipulation functions manipulate text strings and fields with a text data type in the report definition. For example, text functions let you compare strings, insert and extract characters within strings, and convert lowercase strings to uppercase strings.

Defining a Computed Field Using a Function

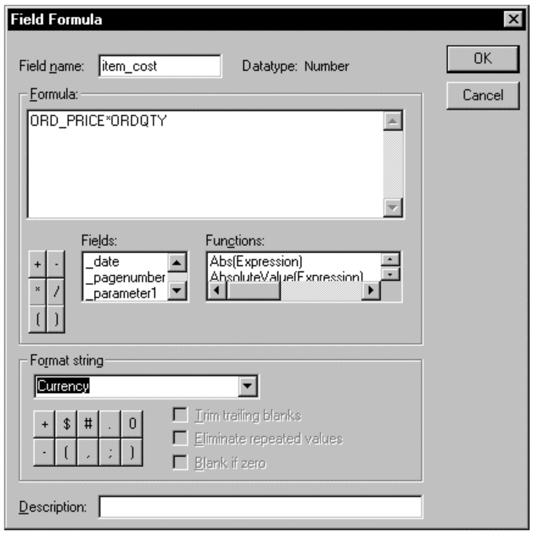
When you define a computed field that uses a function, the function performs a calculation and returns a value to the computed field. You can use a function by itself or embed it in a more complex expression.

To define a computed field for your report definition using a function:

1. Click the Insert Field toolbar button, or choose Field from the Insert menu and then click the Define New Field button in the Insert Field dialog box that appears.

   Alternatively, choose Field Definitions from the Edit menu and then click the Define New button in the Edit Field Definition dialog box that appears.

   In all cases, the Field Formula dialog box appears:



Field name: item_cost Datatype: Number

Formula: ORD_PRICE*ORDQTY

Fields: _date, _pagenumber, _parameter1

Functions: Abs(Expression), AbsoluteValue(Expression)

Format string: Currency

Trim trailing blanks
Eliminate repeated values
Blank if zero

Description:

2. Optionally, enter a field name in the Field Name edit control (or accept the default).
3. Position the cursor in the Formula multi-line edit control, and then double-click a function in the Functions list box.

   The Function Editor dialog box appears.
4. After completing the function definition in the Function Editor (described in the following section), you are returned to the Field Formula dialog box.
5. Optionally, format the field's appearance using the Format String group box options.
6. Optionally, enter a description for the field in the Description edit control.
7. Click OK.

Using the Function Editor Dialog Box

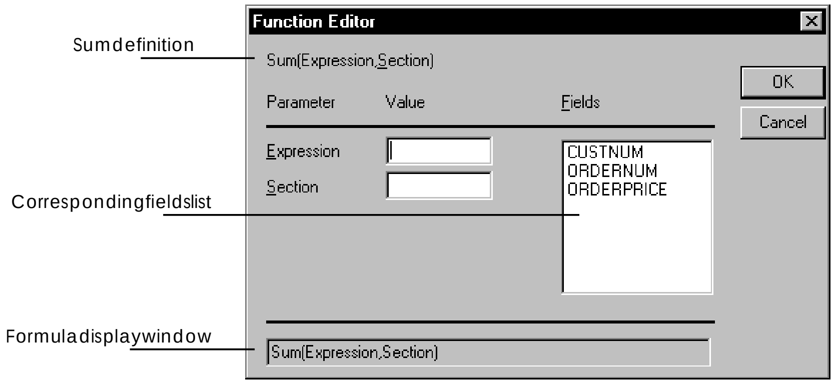
The Function Editor dialog box is displayed whenever you double-click a function name in the Functions list box of the Field Formula dialog box. It prompts you to choose the correct fields to enter for a function, displaying the syntax for the selected function, value boxes for the parameters the function requires, and a scroll box of fields that you can use for each function parameter.

Formulating a Function

For example, suppose you want to total the cost of all items for an order:

1. Double-click Sum in the Functions list box of the Field Formula dialog box.

   The Function Editor dialog box appears:



Sum(Expression,Section)

CUSTNUM
ORDERNUM
ORDERPRICE

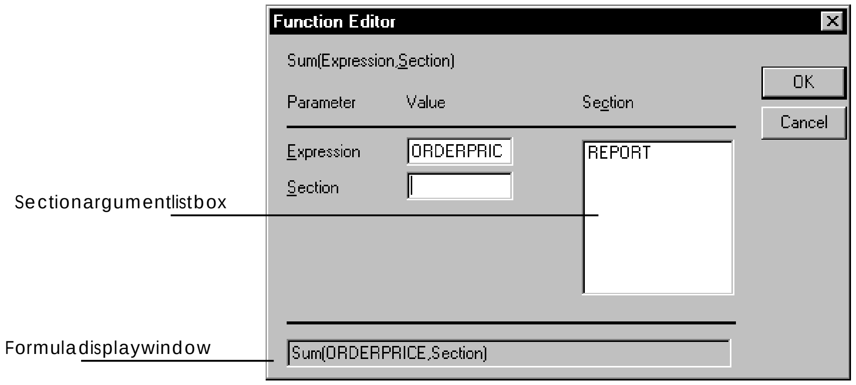
Sum(Expression,Section)

Notice that this dialog box displays:

- The sum definition for the Sum function that was double-clicked in the Field Formula dialog box.
- The fields that correspond to the expression of the Sum definition.
- The field the expression will act upon appears in the field formula display window.

2. Click the Section edit control.

   The Fields list box changes to the Section list box for section arguments:



Sum(Expression,Section)

Expression: ORDERPRIC

REPORT

Sum(ORDERPRICE,Section)

Note: When you click in the Section edit control, the Function Editor only displays the fields that make sense for the section argument in the Fields list box.

3. Double-click the desired field in the Section list box.

   Notice that the formula changes in the formula display window.

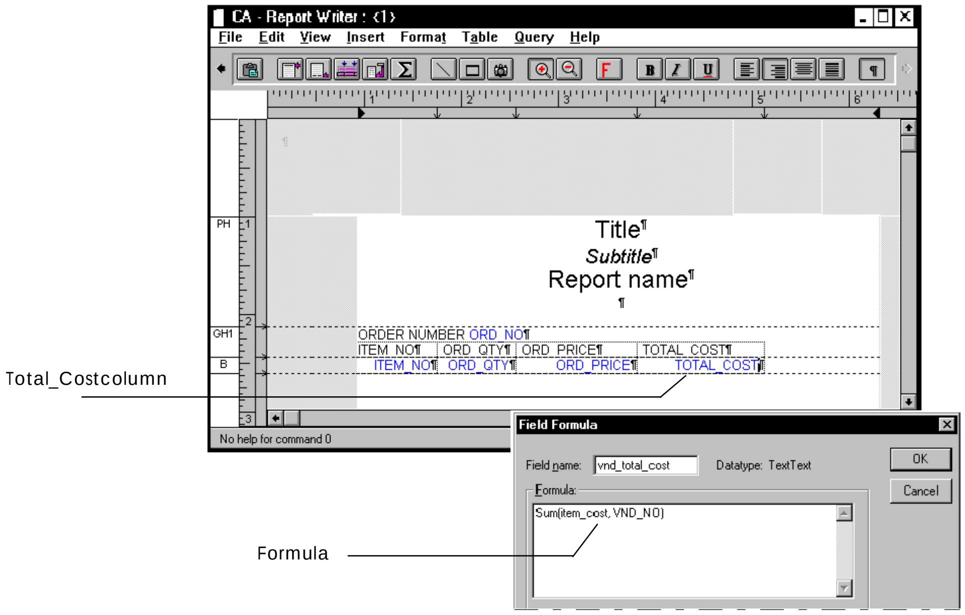
4. Click OK to return to the Field Formula dialog box.

Defining a Sum Field with the Sum Button

CA-Report Writer provides a quick method for creating Sum fields using the Sum toolbar button. To define a Sum field within your report definition, use this method:

1. Position the cursor in a column of numeric values that you want to sum, and then click the Sum toolbar button.

   CA-Report Writer displays the completed formula in the Field Formula dialog box. For example:



Field name: vnd_total_cost   Datatype: TextText

Formula:

Sum(item_cost, VND_NO)

2. Complete the field definition (for example, replace the generic field name).

3. Click OK.

Note: If you click the Sum button in a table column in a group footer that uses the same table style as the body section, CA-Report Writer automatically creates a Sum field using the column name and group break field for the function arguments.

Using Parameter Fields

A *parameter field* is a field whose value is passed to something else—a report, a main report from a subreport, or a bucket table. You define the prompt, data type, and default value for the parameter field.

Parameter Fields for Reports

Defining a parameter field for a report causes the CA-Report Writer to prompt the user to enter a value each time the report is run. For example, suppose you have a payroll report that lists all employees and their salary histories in employee ID order. To run a report on a single employee's salary history, you can define a parameter field based on employee ID. Each time the report is run, the user is prompted for an employee ID, the parameter value is passed to the report query, and the query is executed.

Parameter Fields for Subreports

You can also define a parameter field in a subreport that is used to pass values from a main report to the subreport. When the main report is run, the parameter value is passed to the report query defined in the subreport and the query is executed.

Parameter Fields for Bucket Tables

Parameter fields can also be used to pass values from bucket tables. When you create a bucket table, the parameter field is *not* mapped to a query.

For more information about parameter fields, subreports, and bucket tables, refer to the online help system.

Formatting Fields

Fields inserted into a report have an associated format string that determines how the data is displayed on the report. The *format string* is a mask for the field, designating the exact punctuation and placement of data. You can use predefined format strings or you can create your own. You can also assign formatting options to fields that suppress trailing blanks and repeated values.

Using Format Strings

CA-Report Writer provides various format strings you can use to format numeric and date-time report fields. You can also create your own format string using the formatting string symbols in the Field Formula dialog box. The symbols in the format strings determine the way the values are formatted.

Note: Some of the symbols refer to strings specified in the International section of the Control Panel. You can change these strings in the Windows or Windows NT Control Panel (click the International icon to see and change these strings).

Applying Formatting to a Field

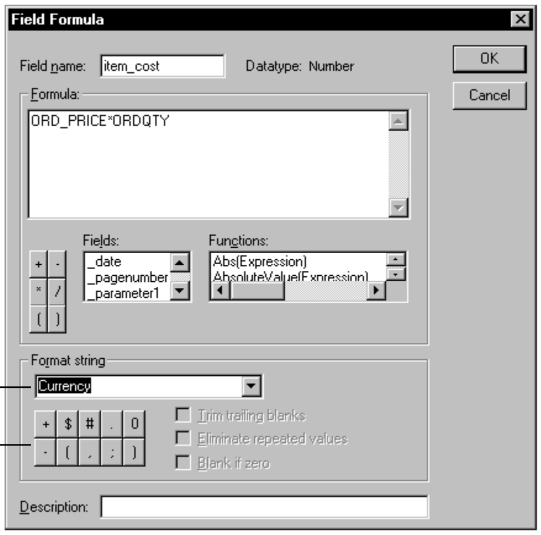
You can apply formatting to a field in both the Format Field and Field Formula dialog boxes. To do so:

1. Click the Insert Field toolbar button, or choose Field from the Insert menu and then click the Define New Field button in the Insert Field dialog box that appears.

   Alternatively, choose Field Definitions from the Edit menu and then click the Define New button in the Edit Field Definition dialog box that appears.

   In all cases, the Field Formula dialog box appears.

2. Select a predefined format from the Format String drop-down list box:



Field name: item_cost
Datatype: Number
Formula: ORD_PRICE*ORDQTY
Format string: Currency
Description:

   Alternatively, use the Format String buttons to create your own format string.

   Note: The Field Formula dialog box will be empty if you use the Insert Field method for accessing it.

3. Select the Trim Trailing Blanks option to eliminate trailing blanks in text values.
4. Select the Eliminate Repeated Values option to eliminate repeating values when the same value is repeated in a group or selected rows.
5. Select the Blank if Zero option to display a blank when the field value is zero.
6. Click OK to return to the report definition.

Editing the Format of a Field

To edit a field's format:

1. Double-click the field you want to format to access the Format Field dialog box.

   Alternatively, position the cursor on the specified field and choose the Field command from the Format menu to access the Field Formula dialog box.
2. Make your selections in the appropriate dialog box.
3. Click OK to return to the report definition.

Note: Both dialog boxes offer the same formatting options. For numeric or date fields, select a string from the Format String combo box, or enter a string or select from the string characters displayed. Optionally, you can select options to print a blank when a field value is 0, print only the first value in a group of repeated field values, or eliminate trailing blanks in string values. Note that the Blank If Zero option applies only to numeric fields that are not formatted with a decimal point.

Formatting Numbers

Use numeric format strings to format numeric values with dollar signs, thousandths separators, scientific notation, percentages, and so on. Numeric format strings can have one or two sections, separated by a semicolon:

- If the format string has one section, positive and negative values use the same format and negative numbers use a minus sign prefix.
- If the format string has two sections, you can format positive numbers and negative numbers differently. The first section is the format for positive numbers; the second is for negative numbers.

If you do not specify a format string, CA-Report Writer rounds to 10 significant digits: 1234567.1234 prints as 1234567.123; 1234567891234567 prints as 1.23e15 (this is the same as the General format string).

The following table provides examples of numeric format strings:

Format String	Value	Formatted Value
0.00	100.5	100.50
	-145.10	-145.10
0.00;(0.00)	100.5	100.50
	-145.10	(145.10)
$#,##0.00	100.5	$100.50
	0	$0.00
	2500.25	$2,500.25
	-145.10337	-$145.10
$#,##0.00;($#,##0.00)	100.50365	$100.50
	-145.10	($145.10)
$#,##0.00"CR";$#,##0.00"DB"	1125.9	$1,125.90CR
	-2500	$2,500.00DB
0[S / 1000]	12375	12
	199	0
General	147	147
	1.875	1.875

Formatting Dates

Date-time format strings allow you to control how dates and times appear in reports. The following table shows some examples of these strings:

Format String	Value	Formatted Value
mm/dd/yy	Jan 15, 1999	01/15/99
mm/dd/yyyy	Jan 9, 1999	01/09/1999
m/d/yy	Jan 9, 1999	1/9/99
dd.mm.yy	Jan 9, 1999	09.01.99
Mmm d, yyyy	Jan 9, 1999	Jan 9, 1999
dd-MMM-yy	Jan 9, 1999	09-JAN-99
Mmmm d, yyyy	Jan 9, 1999	January 9, 1999
hh:mm:ss	4:53:10 PM	16:53:10
hh:mm:ss AM/PM	4:53:10 PM	04:53:10 PM
mm/dd/yy hh:mm:ss	Jan 9, 1999 9:43	01/09/99 09:43:00

Saving Reports

When you create a new report, you should save the report and then save any additional edits on a frequent basis. To initially save a new report, click the Report Writer's Save toolbar button. To save any subsequent changes, just click the Save toolbar button again.

After saving your report, you can then continue to define other reports or queries, switch to another tool or window in the IDE (using the mouse or the Window menu), or exit the Report Editor.

Once you close the Report Editor, you will be prompted to store any changed entities. The binary report entity will then be stored in the repository.

Printing a Report

This section tells you how to print reports, how to preview a report as it prints, and how to create print status messages.

You can print a report from the CA-Report Writer window or the CA-Report Viewer window using one of the following methods:

- Click the Print toolbar button in the CA-Report Writer window.
- Choose Print from the File menu in the CA-Report Writer window.

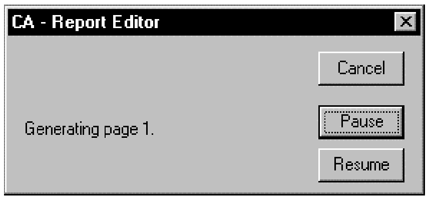
CA-Report Writer displays a standard Print dialog box where you can select a print range, print quality, and the number of copies to print.

Tip: If you want to select a different printer or change the page orientation, click Setup to access the Print Setup dialog box.

Pausing Printing

To temporarily pause the printing process:

1. Click the Pause button in the CA-Report Editor message box that displays while the report is printing:



Generating page 1.

2. Click Resume to start printing again.

For more information about this dialog box, see Adding Print Status Messages.

Previewing a Report as It Prints

You can set up the CA-Report Writer so you can see each page of a report online as it prints. To do so:

1. Choose Properties from the File menu to access the Report Properties dialog box.
2. Select the Preview While Printing option in the Display group box.
3. Click OK.

Note: If you have the Preview While Printing option selected, the CA-Report Editor message box is not displayed (see the next section). To interrupt the printing of a report, press the Esc key.

Adding Print Status Messages

The CA-Report Writer allows you to change the text that appears in the CA-Report Editor message box that displays as you are printing a report. For example, if you are printing a report that lists orders by vendor, you can create a message to indicate each order number as it prints and each vendor name as it changes.

If you do not define print messages, the CA-Report Writer uses the default Generating page *x* and Printing page *x* messages.

You can display a message using data from one or more fields on a report definition by adding a print message to a report section. If you have more than one message, attach each to a different section. Any section can have a print message. If you place the message in the report header section, it displays when the report starts printing. If you place it in the page header] section, it is re-evaluated at the beginning of each page. Messages placed in the body section are updated as each record is processed.

Example

To add a print status message:

1. Position the cursor in the section of the report where you want to place the print message.
2. Choose the Section command from the Format menu.

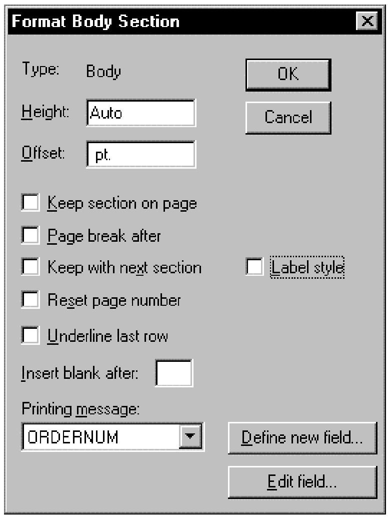
   If the section is not defined as a group, either the Format Section or Format Body Section dialog box appears.

   If, however, the section is defined as a group, the Format Group Section dialog box, shown earlier, appears instead.

   Note: For more information about defining a section as a group, see Grouping Data earlier in this chapter.

Using an Existing Message

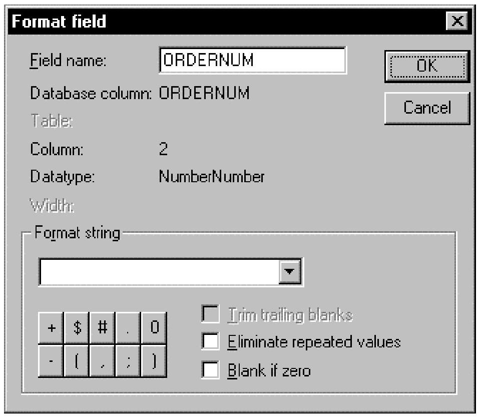
3. To use one field for the print message, select it from the Printing Message drop-down list box. For example:



Type: Body
Height: Auto
Offset: pt.
Insert blank after:
Printing message: ORDERNUM

Note: If you want to define a new message using one or more fields, skip to step 5.

4. Optionally, to change the name of this field or edit its format string, click the Edit Field button to access the Format Field dialog box:



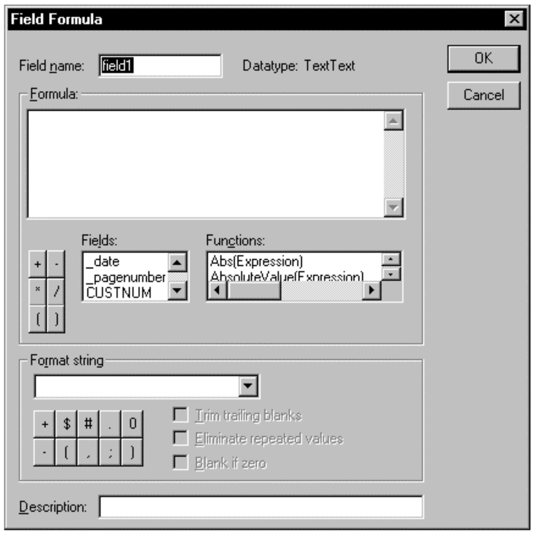
Field name: ORDERNUM
Database column: ORDERNUM
Table:
Column: 2
Datatype: NumberNumber
Width:

Defining a New Message

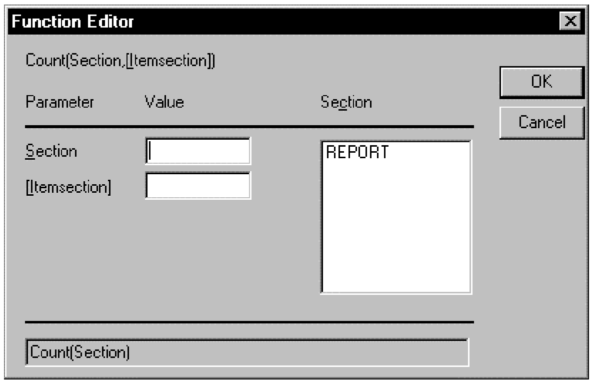
5. Alternatively, to define a new message using one or more fields, do one of the following:

- Click the Define New Field button

The Field Formula dialog box appears:



- To add a field to the formula, double-click the specified field in the Fields list box.
- To add a function to the formula, double-click the specified function in the Functions list box to access the Function Editor dialog box:

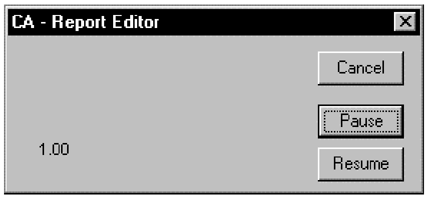


- You can enter a field in the Section edit control under Value, and then click OK to add the formula.

  (For more information about creating formulas, see Defining Computed Fields earlier in this chapter.)

6. Click OK in the appropriate Format Section dialog box to associate the message with the section.

7. Click the Print Report toolbar button to print the report and display the message you created. For example:



Note: If you have the Preview While Printing option selected in the Report Properties dialog box, the CA-Report Writer does not display the new message. Deselect this option to view your message.

Note: The CA-Report Writer does not display a print message until it computes all field values referenced by the section. This means that if you include a report total (or any non-literal column) in a message used in a page header, the CA-Report Writer cannot process the report. Instead it displays the following message: “Field . . . illegally references a field or function which cannot appear in the ... section.”

Using the Report in an Application

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

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

Associating a report with either of these events is easy. Simply use the report entity name in the ButtonClick event property of the push button or the Event Name property of the menu item. Then, when you select the push button or menu item, the report will be displayed. See **Chapter 5: Using the Menu** Editor for more information on associating a report with a menu selection.

Exporting a Report to a File

You can export reports to DOS files in one of two formats:

- ASCII text (unformatted)
- Microsoft's Rich Text Format (RTF)

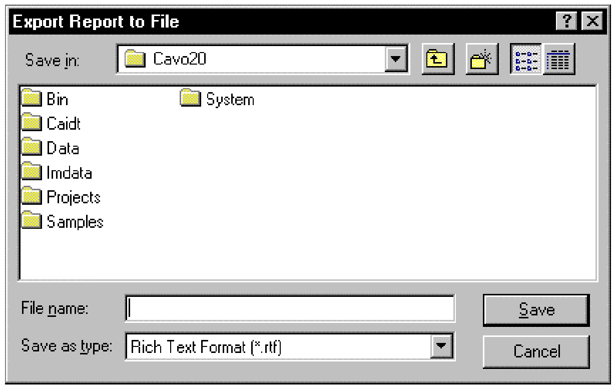
  RTF preserves fonts, paragraph formatting, and alignment characteristics used in the report; and it is supported by several Windows word processors.

When you export reports to text files, each paragraph ends with a new line, with tabs separating report column values.

To export a report:

1. Open the report in the CA-Report Writer window.
2. Choose the Export Report To File command from the File menu.

   The Export Report to File dialog box appears:

3. Select the drive, path, and file name for the exported file and select the type of file in which you want to export the report.

   The CA-Report Writer displays a dialog box as it exports the report. Click Pause to pause the export; click Cancel to stop the export.

Chapter

10 Using the Image Editor

You can use the CA-Visual Objects Image Editor to create custom images, such as icons, cursors, bitmaps, and ribbons, for your applications. It provides a drag-and-drop interface that allows you to:

- Create icon, cursor, bitmap image, and ribbon entities (and classes) that can be used as mouse pointers, application icons, window icons, and window control icons for push button, radio button, check box, and fixed icon controls
- Load existing icon (.ICO), cursor (.CUR), or bitmap (.BMP) files
- Modify the entity name, related file name, and image of the icon, cursor, bitmap or ribbon; and save your changes
- Use drawing tools, including a color palette, lines, ellipses, rectangles, and rotation tools, to create new images or to modify existing images
- Edit each pixel of an image while simultaneously viewing how it will appear in your application
- Use temporary buffers to store up to six images; use drag-and-drop to transfer images to and from the temporary buffers or to delete images via the “trash can”
- Manipulate images using the Windows Clipboard

Starting the Image Editor

Simply double-click on an icon, cursor, bitmap, or ribbon entity in the Repository Explorer’s list view pane to open an Image Editor window for that image. You can also access the Image Editor using the Tools menu or the New Entity toolbar button. (You can open as many individual Image Editor windows as you wish.)

Workspace Overview

The following illustration shows the various features of the Image Editor:

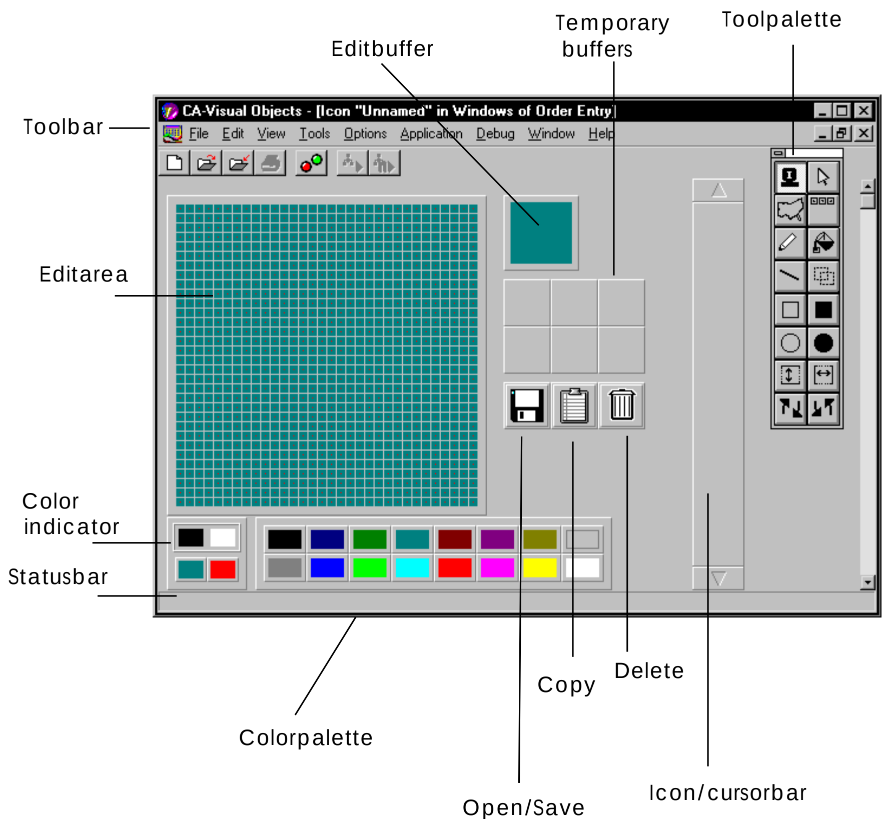
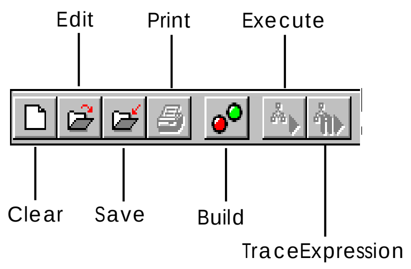


Image Editor Components

The Image Editor window contains it own menu bar, toolbar, tool palette, and other unique components:

Toolbar

The Image Editor toolbar contains buttons for frequently used menu commands:



The Clear, Edit, and Save buttons are described in this chapter. See **Chapter 2: Working in the** Desktop and **Chapter 11: Debugging Your Applications** for information about the other toolbar buttons.

Note: The Print toolbar button is disabled because printing from the Image Editor is not supported.

Edit Area

This is the area where you edit an image using the tools on the tool palette. It can be a destination for a drag-and-drop operation.

Refer to Using the Edit Area in this chapter for more information.

Edit Buffer

This area shows the image in the same size in which it will appear on the screen. It can be a source or destination for a drag-and-drop operation.

Temporary Buffers

Each of these six areas holds an image whose type depends on the current mode. They are useful for temporary storage of images you are modifying and to transfer images between files. Each temporary buffer can be a source or destination for a drag-and-drop operation.

Tool Palette

This set of buttons contains the Mode buttons described below and a set of drawing tool buttons. Refer to Using the Tool Palette in this chapter for specific information on the various drawing tools.

Mode Buttons

The four buttons at the top of the tool palette allow you to toggle between several modes—Icon, Cursor, Bitmap, and Ribbon—depending on which type of image object you wish to edit. The following table describes each mode and its corresponding image object type.

Mode	Button	Description
Icon	I	Icons can be variable in size and can use a 2-, 16-, or 256-color palette.
Cursor	↖	Cursors are small bitmaps which can use a 2-, 16-, or 256-color palette. Each cursor also has a hot spot, which is a single pixel that determines where the mouse action is to take place; for example, the hot spot on a standard arrow cursor is normally located at the tip of the arrow.
Bitmap		Bitmaps can be variable in size (16x16, 32x32, 48x48, or 64x64) and can use a 2-, 16-, or 256-color palette.
Ribbon	▣▣▣	Ribbons are composed of a series of bitmaps that can be variable in size (16x16, 32x32, 48x48, or 64x64), and can use a 2-, 16-, or 256-color palette.

Icon/Cursor Bar

This bar shows the images contained in a ribbon. Click the arrows to scroll through the images; use the left mouse button to advance one image at a time, and the right mouse button to advance to the next or previous page of images. Each item shown on the bar can be a source or destination for a drag-and-drop operation.

Open/Save Button

Drag an image from the edit buffer, a temporary buffer, or the icon/cursor bar and drop it on this button to invoke File Save Image File. Click this button and drag the disk icon to the edit area or edit buffer to invoke File Open Image File.

Copy Button

Drag an image from the edit buffer, a temporary buffer, or the icon/cursor bar and drop it on this button to invoke Edit Copy, transferring the image to the Windows Clipboard. Double-click this button to view the contents of the Windows Clipboard.

Delete Button

Drag an image from the edit buffer or a temporary buffer and drop it on this button to remove the image from the buffer.

Important! *You will not receive any confirmation before the image is removed, even if there are unsaved changes in the buffer.*

Color Indicator	The four color rectangles in this box show the active colors for the drawing tools and screen in the edit area and edit buffer. The top two colors correspond to the primary and secondary solid colors. The bottom two colors correspond to the primary and secondary transparent colors. Refer to the Using the Color Indicator and the Editing Icons and Cursors sections in this chapter for specific information on changing these colors.
Color Palette	This box allows you to change the solid and transparent colors in the color indicator box. You can choose a 2-color palette (black and white), a 16-color palette or a 256-color palette for icons, cursors, bitmaps, and ribbons. Refer to Editing Icons and Cursors in this chapter for specific information on using the color palette.
Status Bar	The status bar provides information on the cursor position within the edit area, the location of the hot spot pixel on a cursor object, the size of an area you are outlining, and the function of buttons on the tool palette.

Customizing the Image Editor

You can set a number of default options for the Image Editor—such as image type and size, grid color, and color palette—by choosing the Properties command from the Options menu. The Image Editor Options dialog box appears:

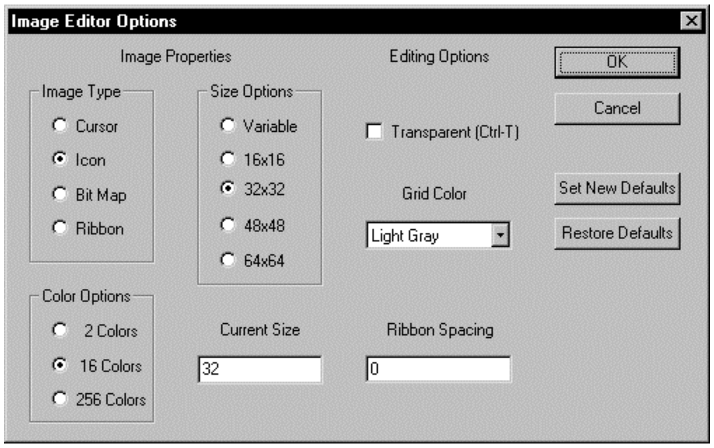


Image Editor Options

Image Properties

Image Type: ( ) Cursor [✓ Icon] ( ) Bit Map ( ) Ribbon

Size Options: ( ) Variable ( ) 16x16 [✓ 32x32] ( ) 48x48 ( ) 64x64

Color Options: ( ) 2 Colors [✓ 16 Colors] ( ) 256 Colors

Current Size: 32

Editing Options

[ ] Transparent (Ctrl-T)

Grid Color: Light Gray

Ribbon Spacing: 0

Typically, you will want to select the most commonly used settings as permanent system defaults; and then, if necessary, override them on an application-by-application basis using the Set New Defaults button. Of course, you can restore the default settings at anytime by selecting the Restore Defaults button.

The Image Editor Options dialog box provides the following options:

Image Type	From the Image Type radio button group, select a default image type for new images. Valid values are: Cursor, Icon, Bit Map (single icon or bitmap), and Ribbon (series of bitmaps). The default is Icon.
Color Options	Select a default color palette from the Color Options radio button group. Valid values are 2 Colors, 16 Colors (for 16x16 and 32x32 pixel images), and 256 Colors (for 48x48 and 64x64 pixel images). The default is 16 Colors.
Size Options	From the Size Options radio button group, select the default pixel size for images. Valid values are Variable, 16x16, 32x32, 32x32, 48x48, and 64x64. The default is 32x32.
Current Size	The size of the currently loaded image is displayed in the Current Size edit control. Optionally, enter a new value.
Transparent Option	Select the Transparent Option if you want the background color of the application or window to appear as the background color for the specified image at runtime. (See Transparent Colors for more detailed information about this option.)
Grid Color	From the Grid Color drop-down list box, choose a color for the grid: Valid choices are: None (no grid), White, Light Gray, Gray, Dark Gray, and Black. The default is Light Gray.
Ribbon Spacing	If the current image is a ribbon bitmap, specify the spacing between each image within the ribbon.

Loading Images

You can open existing icon (.ICO), cursor (.CUR), and bitmap (.BMP) files in the Image Editor in order to view them in the icon/cursor bar and transfer them to the edit area or a temporary buffer.

To load a single icon, cursor, bitmap, or ribbon image object:

1. Start the Image Editor.

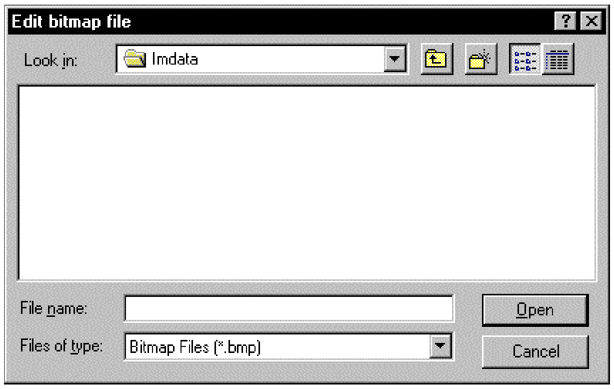
   Like all CA-Visual Objects tools, the Image Editor is accessed using the Tools menu or the New Entity toolbar button.

2. In the Image Editor, click one of the Mode buttons to specify the type of image you wish to load.

3. Do one of the following:

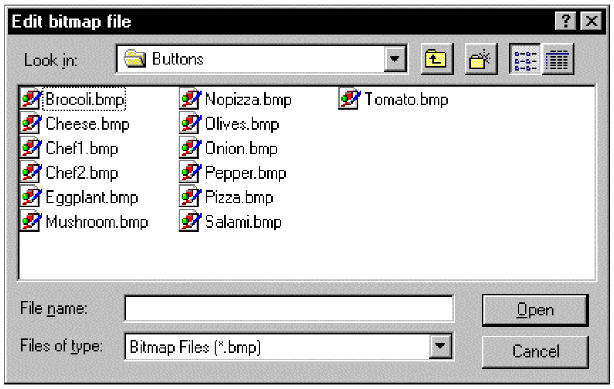
- Click the Edit tool bar button
- Choose the Open Image File command
- Click the Open/Save button and drag the disk icon to the edit buffer

In all cases, the appropriate dialog box—Edit Icon File, Edit Bitmap File, Edit Cursor File, or Initialize Ribbon from Bitmap File—appears. For example:



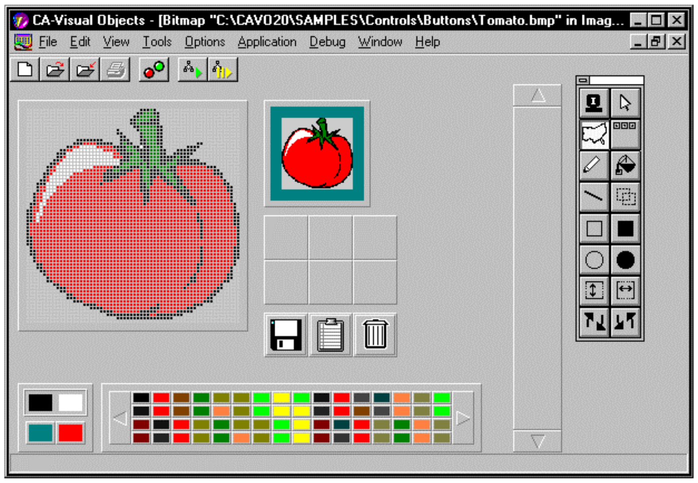
4. Select a directory folder from the Look In combo box.

Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory. For example, below is the BUTTONS folder in CAVO27/SAMPLES/CONTROLS:



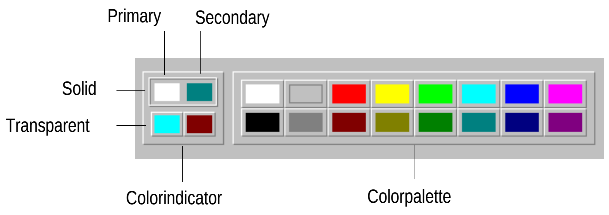
5. Choose the appropriate .BMP, .ICO, or .CUR file you want to load (for example, TOMATO.BMP in the BUTTONS folder shown above).

CA-Visual Objects loads the image defined in the specified file into the edit area:


CA-Visual Objects - [Bitmap "C:\CAVO20\SAMPLES\Controls\Buttons\Tomato.bmp" in Imag...
File Edit View Tools Options Application Debug Window Help

Using the Color Indicator

The Image Editor displays the four current colors in the 2x2 color indicator. The top row displays the two solid (drawing) colors and the bottom row displays the two transparent (background) colors. The left color is called the primary color, and the right color is called the secondary color. You can draw with the primary color by using the left mouse button, while the right button controls the secondary color.



Solid Colors

Solid Colors is the default option setting for the color indicator; these colors define the icon or cursor image. You will spend most of your time using solid colors. The color palette displays 2, 16, or 256 solid colors for icons, cursors, bitmaps, and ribbons—depending on your default settings for the Image Editor. At runtime, only pixels you have drawn with solid colors are displayed, while the host window background fills in for any empty pixels.

Note: To erase solid colors, switch to transparent colors by selecting the Transparent Colors command from the Options menu, and then use the Pencil or Fill tool with the left mouse button.

Transparent Colors

You can also choose the Transparent Colors option for the color indicator; use these colors to simulate the way the icon will look when displayed at runtime on windows with different background colors. Once you choose the primary transparent color, the secondary color will be automatically determined. The transparent colors are temporary (appearing only in the Image Editor) and may not appear at runtime if the host window has a different background color.

You will see the primary transparent color displayed as a frame surrounding the icon in the edit buffer. When the transparent colors are active, you can change the background frame color by clicking on a different color with the left mouse button. You can see what the icon will look like at runtime, with the standard gray window background colors, by dragging it from the edit buffer to a temporary buffer.

When the transparent colors are active, you can draw just like you can in the solid color mode. If you draw a transparent color with the left mouse button, you will overwrite any solid color pixels. Then at runtime (or in the temporary buffer), these pixels will be empty, so the window background colors will appear instead of solid pixels. You can also use the secondary transparent color to allow a different background color to display, based on the color of the host window. It is advisable to not use either of the transparent colors as solid colors to avoid confusing effects.

Using Drag-and-Drop

The table below shows how you can use drag-and-drop techniques to perform certain tasks more efficiently. There are six possible locations where you can begin or end a drag-and-drop operation. The “From” locations are indicated by the row headings, while the “To” locations are indicated by the column headings.

To / From	Edit Buffer	Temporary Buffer	Open/ Save Button	Clipboard Button	Delete Button	Icon/ Cursor Bar
Edit Buffer	–	Yes	Save file	Yes	Yes	Yes
Temporary Buffer	Yes	–	Save file	Yes	Yes	Yes
Open/Save Button	Open file	No	–	No	Delete file	No
Clipboard Button	No	No	No	–	Delete from Clipboard	No
Delete Button	No	No	No	No	–	No
Icon/Cursor Bar	Yes	Yes	Save file	Yes	Yes	–

Using the Edit Area

The edit area is where you make changes to an image. As you modify the pixels in the image, you can immediately see in the edit buffer how your changes affect the icon, cursor, bitmap, or ribbon. Choose commands on the Options menu to control the appearance of the edit area. Click buttons on the tool palette or choose commands on the Edit Select from Palette menu to activate various tools to modify the image. Use the color palette to control the colors used by the various drawing tools and the background color of the edit area.

Setting Colors

Within the edit area you can change the drawing colors, the background colors, and the grid color.

Drawing Colors

To change the drawing colors used by all drawing tools:

1. Choose Solid Colors from the Options menu.

   The highlight rectangle in the color indicator box moves to the top to indicate that drawing colors are now active.

2. Click with the left mouse button on a color in the color palette to set the drawing color for the left mouse button.
3. Click with the right mouse button on a color in the color palette to set the drawing color for the right mouse button.

Background Colors

To change the background colors, which are displayed only in the Image Editor and not at runtime:

1. Choose Transparent Colors from the Options menu.

   The highlight rectangle in the color indicator box moves to the bottom to indicate that background colors are now active.

2. Click with the left mouse button on a color in the color palette to change the primary background color. This color is visible only in the edit area and edit buffer, not in the temporary buffers.

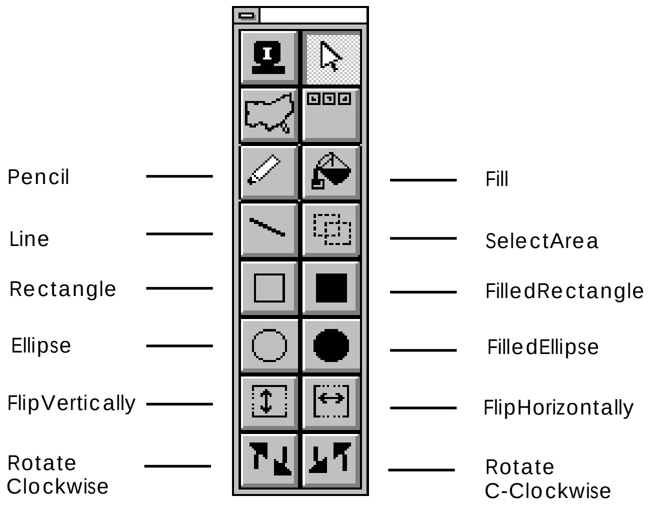
Note: The secondary background color automatically changes when you change the primary color. Unlike the primary transparent color, pixels set to the secondary background color do affect the way the background color is displayed at runtime. Move the icon or cursor image to a temporary buffer to see how the secondary background color changes for areas in which you have used it. The color in which pixels set to the secondary background color appear in another area or window is controlled by the background color of that area or window. For this reason, we recommend that you not set pixels to the secondary background color, as the results can be hard to predict. Also, you should avoid using either of the two transparent colors as solid colors, since you cannot distinguish between a solid and transparent pixel of the same color by looking at the edit area.

3. When you have set the desired background color, choose the Options Solid Colors menu command to highlight the drawing colors at the top of the color indicator. Note that a check mark appears on the Options menu next to the colors that are active.

Tip: Choose the Options Transparent Toggle menu command to be able to switch easily between solid colors and transparent Colors.

Using the Tool Palette

The buttons on the floating tool palette provide a variety of tools to modify the icon or cursor image in the edit area. You can click and drag the tool palette to move it to any location you want. Use the drawing tools to modify individual pixels and draw lines, rectangles, and ellipses. Use the rotation tools to flip or rotate the entire image or a selected area. This section provides a description of how to use each tool on the tool palette:



Pencil

Use the Pencil to color individual pixels, for freehand drawing, or to set the hot spot location for a cursor. To use the Pencil tool:

1. Click the Pencil button or choose Pencil from the Edit Select from Palette menu.
2. Click with either mouse button to change the color of an individual pixel, or click and drag to change multiple pixels.

To set a hot spot in a cursor object only:

1. Click the Pencil button or choose Pencil from the Edit Select from Palette menu.
2. While holding down the Ctrl key, click with either mouse button on the pixel that you want to designate as the hot spot for the current cursor object.

An outline appears around the pixel to identify it as the hot spot. The hot spot is the point on the cursor that identifies the area to apply an action to when you click a mouse button. For example, the hot spot in the cursor for the Pencil tool itself is located at the point of the pencil, so the color of the pixel under the point changes when you click with a mouse button.

You can also use the Pencil tool to erase solid colors. To do this:

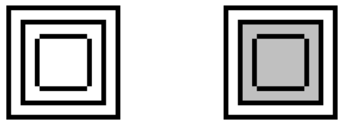
1. Switch to transparent color mode using the Options Transparent Colors menu command.
2. Click the Pencil button or choose Pencil from the Edit Select from Palette menu.
3. Click with the left mouse button to erase an individual pixel, or click and drag to change multiple pixels.

Fill

Use Fill to change the color of pixels inside an existing shape or to change the color of all pixels that use the primary background color.

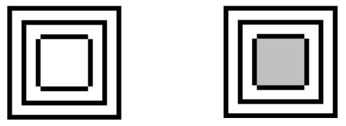
The basic operation of this tool fills all pixels in a bounded area to a selected color. This Fill option "spills through" diagonal gaps, as shown in the example below. To use this option:

1. Click the Fill button or choose Fill from the Edit Select from Palette menu.
2. Click with the left or right mouse button inside an area bounded by pixels set to a drawing color.



To fill *without* spilling through diagonal gaps:

1. Click the Fill button or choose Fill from the Edit Select from Palette menu.
2. While holding down the Shift key, click with the left or right mouse button inside an area bounded by pixels set to a drawing color.



To change all pixels in the edit area that are set to the primary background color to the selected color:

1. Click the Fill button or choose Fill from the Edit Select from Palette menu.
2. While holding down the Shift key, click with the left or right mouse button anywhere in the edit area.

You can also use the Fill tool to erase solid colors. To do this:

1. Switch to transparent color mode using the Options Transparent Colors menu command.
2. Click the Fill button or choose Fill from the Edit Select from Palette menu.
3. Click with the left mouse button inside an area bounded by pixels erase the solid colors.

Line

Use the Line tool to draw a line between two pixels in the edit area. To use the Line tool:

1. Click the Line button or choose Line from the Edit Select from Palette menu.
2. Click with the left or right mouse button on the pixel you want to be one endpoint of the line, then drag to the pixel you want to be the other endpoint.
3. Release the mouse button to draw the line.

Note on the status bar that the initial pixel is denoted by 1,1; as you drag, the coordinates change to indicate the length and angle of the line.

Select Area

Use the Select Area tool to identify an area of pixels that you want to flip or rotate. To use the Select Area tool:

1. Click the Select Area button or choose Select from the Edit Select from Palette menu.
2. Click with the left or right mouse button on the pixel at one corner of the desired area, then drag to the pixel at the opposite corner.
3. Release the mouse button to select the area and identify it with a rectangle.

Note on the status bar that the initial pixel is denoted by 1,1; as you drag, the coordinates change to specify the size of the selection area. To use the rotation tools, you must select a square area of pixels.

Rectangle

Use the Rectangle tool to draw a rectangle in the edit area. To use the Rectangle tool:

1. Click the Rectangle button or choose Rectangle from the Edit Select from Palette menu.
2. Click with the left or right mouse button on the pixel at one corner, then drag to the pixel at the opposite corner of the rectangle.
3. Release the mouse button to draw the rectangle, which will be outlined with the appropriate drawing color based on the mouse button you used.

Note on the status bar that the initial pixel is denoted by 1,1; as you drag, the coordinates change to specify the size of the rectangle.

Filled Rectangle

Use the Filled Rectangle tool to draw a rectangle in the edit area. To use the Filled Rectangle tool:

1. Click the Filled Rectangle button or choose Filled Rectangle from the Edit Select from Palette menu.
2. Click with the left or right mouse button on the pixel at one corner, then drag to the pixel at the opposite corner of the rectangle.
3. Release the mouse button to draw the rectangle, which will be filled with the appropriate drawing color based on the mouse button you used.

Note on the status bar that the initial pixel is denoted by 1,1; as you drag, the coordinates change to specify the size of the rectangle.

Ellipse

Use the Ellipse tool to draw an ellipse in the edit area. To use the Ellipse tool:

1. Click the Ellipse button or choose Ellipse from the Edit Select from Palette menu.
2. Click with the left or right mouse button on a pixel at the outside edge of the desired area, then drag to a pixel at the opposite edge.
3. Release the mouse button to draw the ellipse, which will be outlined with the appropriate drawing color based on the mouse button you used.

Note on the status bar that the initial pixel is denoted by 1,1; as you drag, the coordinates change to specify the size of the ellipse.

Filled Ellipse

Use the Filled Ellipse tool to draw an ellipse in the edit area. To use the Filled Ellipse tool:

1. Click the Filled Ellipse button or choose Filled Ellipse from the Edit Select from Palette menu.

2. Click with the left or right mouse button on a pixel at the outside edge of the desired area, then drag to a pixel at the opposite edge.
3. Release the mouse button to draw the ellipse, which will be filled with the appropriate drawing color based on the mouse button you used.

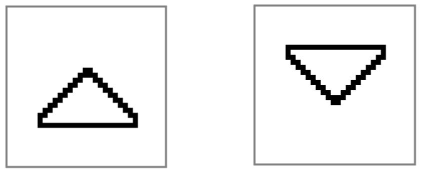
Note on the status bar that the initial pixel is denoted by 1,1; as you drag, the coordinates change to specify the size of the ellipse.

Flip Vertically

Use the Flip Vertically tool to flip the entire edit area or a portion you have selected with the Select Area tool. To use the Flip Vertically tool:

1. If desired, select an area of pixels to flip using the Select Area tool (refer to the instructions earlier in this chapter).
2. Click the Flip Vertically button or choose Edit Flip Vertically.

All pixels in the image or the selected area move to the exact opposite location around an imaginary horizontal line at the halfway point of the edit area. Refer to the picture below for an example of this tool.



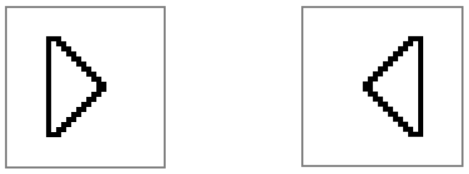
Before vertical flip After vertical flip

Flip Horizontally

Use the Flip Horizontally tool to flip the entire edit area or a portion you have selected with the Select Area tool. To use the Flip Horizontally tool:

1. If desired, select an area of pixels to flip using the Select Area tool (refer to the instructions earlier in this chapter).
2. Click the Flip Horizontally button or choose Edit Flip Horizontally.

All pixels in the image or the selected area move to the exact opposite location around an imaginary vertical line at the halfway point of the edit area. Refer to the picture below for an example of this tool.



Before horizontal flip After horizontal flip

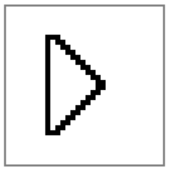
Rotate Clockwise

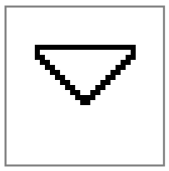
Use the Rotate Clockwise tool to rotate the entire edit area or a square portion you have selected with the Select Area tool. To use the Rotate Clockwise tool:

1. If desired, select a square area of pixels to rotate using the Select Area tool (refer to the instructions earlier in this chapter).

2. Click the Rotate Clockwise button or choose Edit Rotate Clockwise.

The area rotates 90 degrees in a clockwise direction. Refer to the picture below for an example of this tool.


Before clockwise rotation

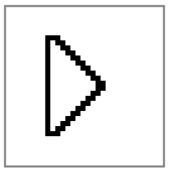

After clockwise rotation

Rotate Counterclockwise

Use the Rotate Counterclockwise tool to rotate the entire edit area or a square portion you have selected with the Select Area tool. To use the Rotate Counterclockwise tool:

1. If desired, select a square area of pixels to rotate using the Select Area tool (refer to the instructions earlier in this chapter).
2. Click the Rotate Counterclockwise button or choose Edit Rotate C-Clockwise.

The area rotates 90 degrees in a counterclockwise direction. Refer to the picture below for an example of this tool.


Before counterclockwise rotation

After counterclockwise rotation

Clearing the Edit Area

To remove the current image from the edit area and edit buffer:

Click the Clear button on the toolbar or choose the File New menu command. The Image Editor prompts you to save any changes to the image.

Note: You can also click and drag the image from the edit buffer to the Delete (trash can) icon; however in this case, Image Editor does not prompt you to save any unsaved changes.

Creating a Ribbon

Suppose you wanted to adapt and expand the Order Entry sample application for a client who happens to be a greengrocer. You could add some “glitz” to the application by creating a ribbon image object to be associated with its menu or one of its window controls. You could use some of the vegetable bitmaps in the BUTTONS folder in the CAVO27\SAMPLES\CONTROLS directory for this purpose.

To create a ribbon entity and bitmap file from existing bitmap files:

1. Start the Image Editor.

   Like all CA-Visual Objects tools, the Image Editor is accessed using the Tools menu or the New Entity toolbar button.

2. In the Image Editor, click the Bitmap mode button.

3. Click the Open/Save button and drag the disk icon to the edit buffer.

   The Edit Bitmap File dialog box appears.

4. Choose the appropriate .BMP file you want to load (for example, TOMATO.BMP).

   CA-Visual Objects loads the image defined in the specified file into the edit area and the edit buffer.

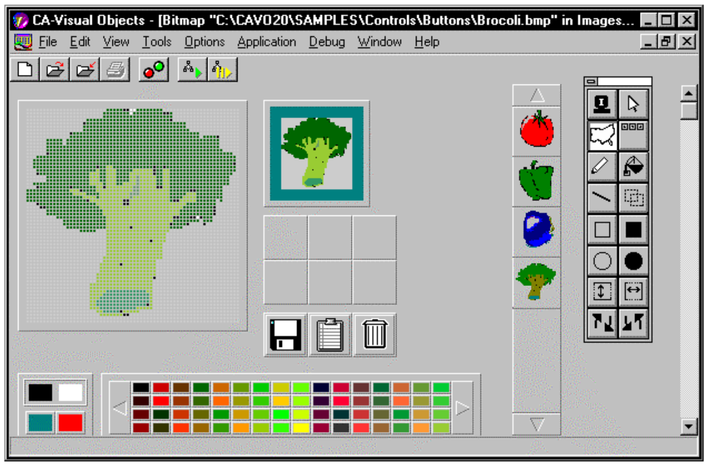
5. Switch to Ribbon mode.

6. Copy the image from the edit buffer to the icon/cursor bar using the drag-and-drop technique.

   Note that after dropping the image on the icon/cursor bar, you are automatically switched back to Bitmap mode.

7. Repeat steps 3–6, adding PEPPER.BMP, EGGPLANT.BMP, and BROCOLI.BMP to the icon/cursor bar.

The Image Editor should now look something like this:


CA-Visual Objects - [Bitmap "C:\CAVO20\SAMPLES\Controls\Buttons\Brocoli.bmp" in Images...
File Edit View Tools Options Application Debug Window Help

8. Save your ribbon entity as **Grocer** and its corresponding bitmap file as **GROCER. BMP**. (See Saving Images below for details.)

See Using Images in Applications for more information about incorporating image objects in your application designs.

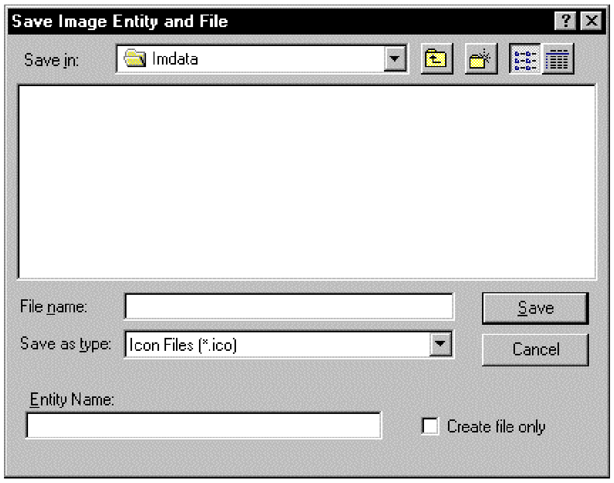
Saving Images

To save the image object in the edit area as an entity in the current module of your CA-Visual Objects application:

1. Do one of the following:

- Click the Save toolbar button
- Choose the Save Image File command
- Drag the image from the edit buffer to the Open / Save (disk) icon

In all cases, the Save Image Entity and File dialog box appears:



Save Image Entity and File

Save in: Imdata

File name:

Save as type: Icon Files (*.ico)

Entity Name:

Note: This dialog box has been updated slightly in CA-Visual Objects 2.7 and now includes a Network push button. For detailed information, see the online help.

2. Select a directory folder from the Look In combo box.

   Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory.

   Use the New Folder toolbar button to create a new directory folder, if necessary.

3. If you are saving the image for the first time, type a name in the Entity Name edit control.

   You will receive a warning if the entity already exists; in that case, specify a different entity name. If there is already an entity name in the Entity Name edit control, you may leave that name, or type over it to save the image under another entity name.

4. Click Save to save the icon, cursor, or bitmap file and create or make changes to the entity.

Note: All of the bitmap images used to create a Ribbon entity are saved as a single .BMP file, which you may use subsequently for creating a toolbar for a window in the Window Editor.

Generating Code

In addition to saving the image you designed as an icon (*.ICO), cursor (*.CUR), or bitmap (*.BMP) file, this action also causes the Image Editor to generate the following code using the file and entity names you specified in the save dialog box:

- An icon, cursor, bitmap image, or ribbon entity

  You can double-click on this entity in Repository Explorer's list view pane to begin editing it with the Image Editor.
- An appropriate subclass—Icon subclass for icons, Pointer subclass for cursors, or Bitmap subclass for bitmaps and ribbons—using the name of the menu
- An Init() method to instantiate the Icon subclass
- A resource entity identifying the generated .ICO, .CUR, or .BMP file name

Using Images in Applications

Once you have created an icon, cursor, bitmap image, or ribbon entity, you can use it in designing applications. The table below describes where image objects can be used when you design an application:

Object Type	Cursor	Icon, Bitmap, or Ribbon
Application	Not applicable	Application Properties dialog box, Icon button; Menu Editor (Menu Properties window)
Shell window	Mouse Pointer	Icon Name
Data window	Mouse Pointer	Icon Name
Datadialog window	Mouse Pointer	Icon Name
Dialog window	Mouse Pointer	Not applicable
Window controls (fixed icon, push button, radio button, and check box)	Not applicable	Caption (for fixed icons only); Image (for other window controls)

In the case of choosing an icon for the application, you must click the Icon button in the Application Properties dialog box. Choose one of the available icons, and click OK to close the dialog box. The new icon will be displayed in the Repository Explorer, as well as when the application is minimized.

In all other cases, you indicate the image object using a value cell in the appropriate Properties window. In some cases, such as with the fixed icon control, you must enter the name directly. In other cases, you can select the name from a drop-down list of all cursor, icon, bitmap image, and ribbon entities in your application. The appropriate source code will be generated by the Window Editor to link entity and file to the window.

In addition, once the Image Editor creates the related class, Init() method, and resource entity, you can utilize your images directly in the source code you create. CA-Visual Objects also contains several system icons and system cursors which you can utilize. See the Icon and Pointer class descriptions in the online help system for a listing of the available system icons and cursors. The system icons and cursors cannot be utilized as described above; instead you must use them by writing custom source code.

Chapter 11 Debugging Your Applications

With a CA-Visual Objects application, there are three distinct types of error that can occur: *compiler errors*, *runtime errors* that are trapped by the error handling system, and *logic errors* in the application that cannot be trapped as errors.

Compiler Errors

When you attempt to build an application, you may receive compiler errors for such things as incorrect syntax usage. 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 Undeclared Variables option.

The resolution of compiler errors 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, CA-Visual Objects provides two tools: the Error Browser and the visual indicators in the Repository Explorer.

Runtime Errors

After all compiler errors are resolved, you will get a successful build for your application and can run it to see what happens. However, 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.

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.

Tip: You can minimize runtime errors by strongly scoping and typing variables (and function and procedure declarations) and by *deselecting* the Undeclared Variables compiler option. Doing this will trap any errors involving the misuse of a variable or function call at compile time. See "Variables, Constants, and Declarations" for more information on strong typing and scoping of variables and "Functions and Procedures" for more information on calling conventions for functions and procedures—both of these chapters are located in the *Programmer's Guide.*

Logic Errors

You may also encounter errors in your application at runtime, known as logic errors, that manifest themselves in less obvious ways than displaying error messages.

This type of error resolution 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 of this nature, CA-Visual Objects provides you with the Debugger, a comprehensive tool designed for just this purpose.

This chapter discusses how to use the various tools and features of CA-Visual Objects to detect and resolve all types of errors. In this chapter, you will learn how to:

- Use the Error Browser and the visual indicators in the various browsers to pinpoint and resolve compiler errors
- Set debugging options at the application, module, and entity levels
- Recognize and resolve runtime errors using both the comprehensive runtime error reporting system that is built into every CA-Visual Objects application and the Debugger for less obvious runtime errors
- Use the Debugger to track down and correct logic errors in your application

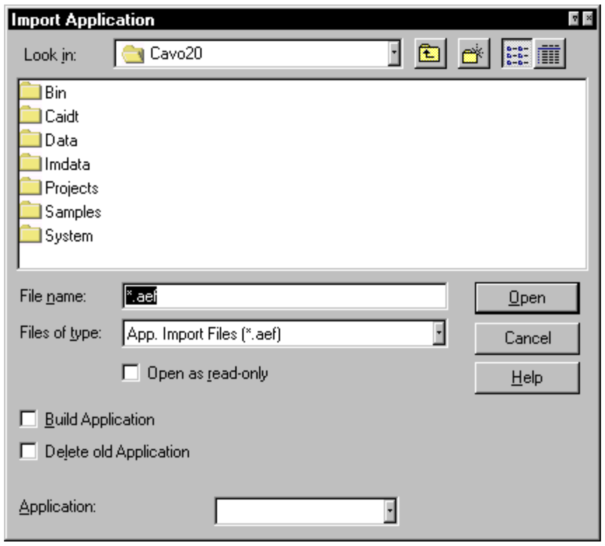
Note: In CA-Visual Objects 2.7, there are several new debugging features and options, including AutoStart debugging and DLL debugging, as well as minor changes to the Error Browser window and the Evaluate Expression dialog box. For details, see the online help.

A Sample Debugging Application

Before you begin working through this chapter, you must import the sample application and add a new module:

1. From the Repository Explorer, choose the File Import menu command.

   The Import Application dialog box appears:

2. Select the CA-Visual Objects \SAMPLES\DEBUG directory, and double-click on the file named SALES.AEF.

   CA-Visual Objects imports the Sales Totals application and returns you to the Repository Explorer.

   Note: If you installed CA-Visual Objects to a drive or directory other than the default directory, then the Application Options dialog box displays. Here you can change the path of the application. (You will also need to change the path for SALES.DBF in the Sales DB Server entity.)

3. To view the modules for the Sales Totals application expand the applications tree by clicking on the + button to the left of the application.

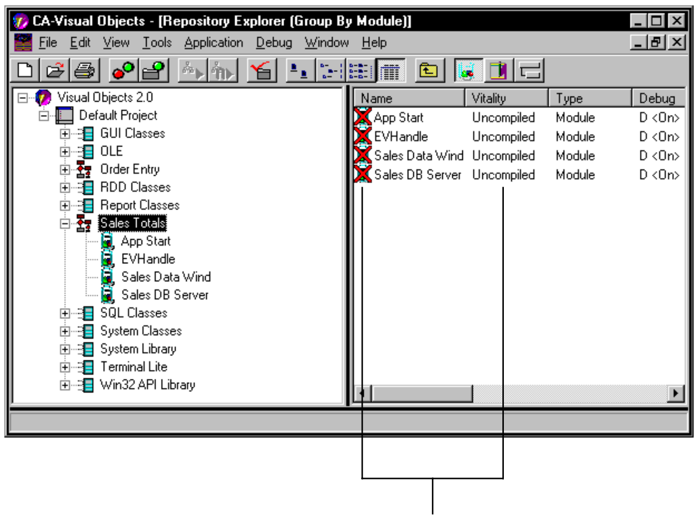
In the Repository Explorer's tree view pane, you will see four modules: App Start, EVHandle, Sales Data Wind, and Sales DB Server.

The source code in the EVHandle module contains a single method, CellDoubleClick(), that is an over-simplified example designed with intentional errors to demonstrate the tools available for debugging an application in CA-Visual Objects. It is used as an example and refined throughout the rest of this chapter as you walk through the various stages of debugging.

Resolving Compiler Errors

The Repository Explorer has indicators to show the compilation status of applications, modules, and entities, as has already been discussed in the “Working in the Desktop” and “Using the Repository Explorer” chapters earlier in this guide.

The Repository Explorer’s list view pane shows that the modules of the Sales Totals application need to be compiled, as indicated by a red X through each of the module’s icons and the “Uncompiled” status in the Vitality column:



Name	Vitality	Type	Debug
App Start	Uncompiled	Module	D <On>
EVHandle	Uncompiled	Module	D <On>
Sales Data Wind	Uncompiled	Module	D <On>
Sales DB Server	Uncompiled	Module	D <On>

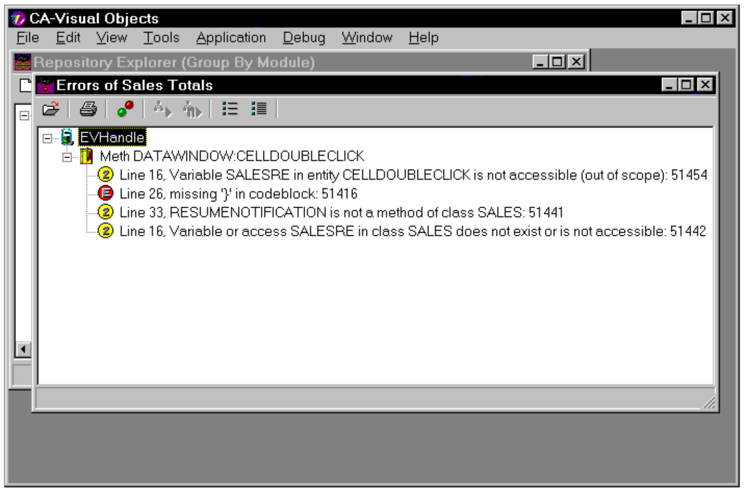
Once you build the application, these indicators change based on the success or failure of the build, on an entity-by-entity basis. If there are errors, the indicators show you the general location of the errors, while the Error Browser allows you to pinpoint their exact location.

The Error Browser is automatically displayed after an unsuccessful build. However, if you close it before resolving all the errors, you can reopen it at anytime by choosing the Error Browser command from the Tools menu.

Building the Application

For example, click the Build toolbar button now to compile the Sales Totals application.

You are launched immediately into the Error Browser, indicating compiler errors and / or warnings in lines 16, 26, and 33 of the CellDoubleClick() method, located in the EVHandle module:


CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Errors of Sales Totals

```
EVHandle
  Meth DATAWINDOW:CELLDOUBLECLICK
    Line 16, Variable SALESRE in entity CELLDOUBLECLICK is not accessible (out of scope): 51454
    Line 26, missing '}' in codeblock: 51416
    Line 33, RESUMENOTIFICATION is not a method of class SALES: 51441
    Line 16, Variable or access SALESRE in class SALES does not exist or is not accessible: 51442
```

Using the Error Browser

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 or warning message. This structure is a convenient mechanism for zeroing in on the error you want to correct.

An error is indicated by a red circle with an “E” inside it. Similarly, warnings and their corresponding severity levels are indicated as follows:

Icon Color	Severity Indicator	Description
Dark yellow circle	“1”	Level 1 warning (most critical)
Yellow circle	“2”	Level 2 warning
Light yellow circle	“3”	Level 3 warning
White circle	“4”	Level 4 warning (least critical)

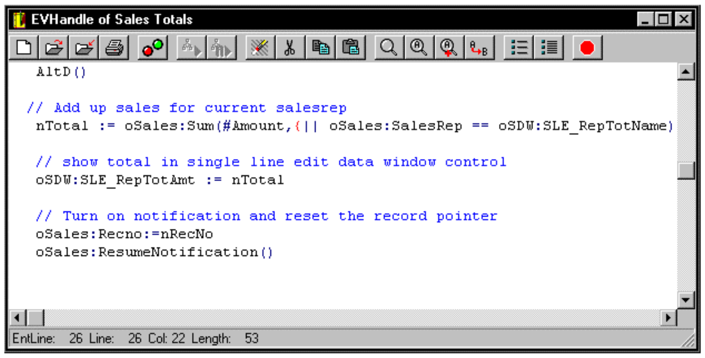
Similar to the Repository Explorer, you can manipulate the tree by clicking on the – / + buttons to collapse or expand, respectively, a particular level (or branch), and the Expand All or Collapse All toolbar button to expand or collapse the entire tree.

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.

Correcting the Errors

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:

1. Double-click on the second error message in your current Error Browser to open the Source Code Editor as follows:



```
  AltD()

// Add up sales for current salesrep
  nTotal := oSales:Sum(#Amount,{|| oSales:SalesRep == oSDW:SLE_RepTotName)

  // show total in single line edit data window control
  oSDW:SLE_RepTotAmt := nTotal

  // Turn on notification and reset the record pointer
  oSales:Recno:=nRecNo
  oSales:ResumeNotification()
```

EntLine: 26 Line: 26 Col: 22 Length: 53

2. Insert **}** (a right brace) just before the right parenthesis at the end of the line to correct this error.

3. Close the Source Code Editor and save your changes.

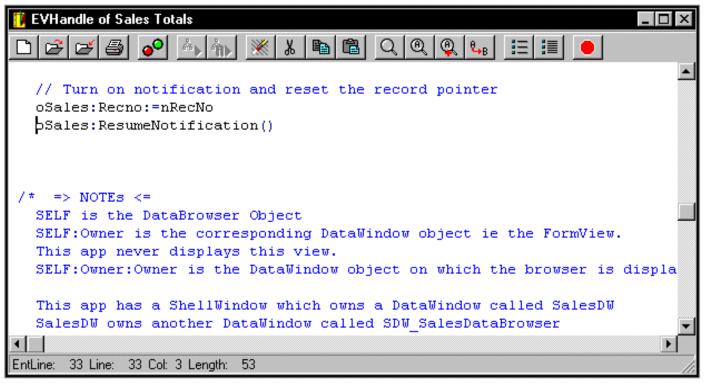
To go to the next item:

1. Switch back to the Error Browser.

   Note: To do this, choose the Error Browser from the list of open windows in the Windows menu or press Ctrl+Tab until you reach the Error Browser.

2. Double-click on the message for line 33.

Your Source Code Editor window should now be positioned at line 33, as shown here:



EVHandle of Sales Totals

```
  // Turn on notification and reset the record pointer
  oSales:Recno:=nRecNo
  oSales:ResumeNotification()



/*  => NOTEs <=
  SELF is the DataBrowser Object
  SELF:Owner is the corresponding DataWindow object ie the FormView.
  This app never displays this view.
  SELF:Owner:Owner is the DataWindow object on which the browser is displa

  This app has a ShellWindow which owns a DataWindow called SalesDW
  SalesDW owns another DataWindow called SDW_SalesDataBrowser
```

EntLine: 33 Line: 33 Col: 3 Length: 53

This is actually a warning, not an error, and it occurred because ResumeNotification() is not a valid method name for the Sales class. Sales inherits most of its methods, including this one, from the DBServer class.

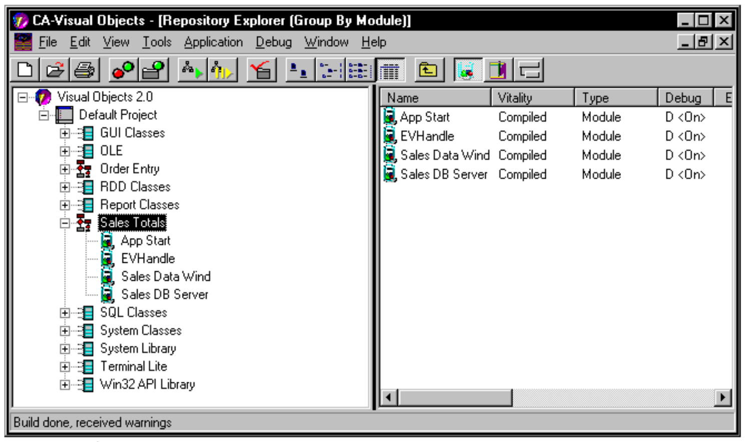
To correct the error:

1. Change the method name from ResumeNotification() to ResetNotification().
2. Close the Source Code Editor and save your changes.

We still have not dealt with the item pertaining to line 16, but this is a warning message that we will ignore for now.

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.

Close the Error Browser and build the application again—this time, the status bar will indicate that there are warnings:



Name	Vitality	Type	Debug
App Start	Compiled	Module	D <On>
EVHandle	Compiled	Module	D <On>
Sales Data Wind	Compiled	Module	D <On>
Sales DB Server	Compiled	Module	D <On>

Build done, received warnings

Messageindicateswarnings

If your application raises *only* warnings, the Error Browser is not launched automatically, but it can be opened manually using the Tools Error Browser menu command. If you open it now, you will see the warning message for line 16, which we have not yet resolved.

Note: You can set up your system so that warnings are displayed as errors. See Setting Compiler Options in the “Working in the Desktop” chapter for detailed information.

Unlike errors, however, warnings do not prevent the application from compiling and may not present any runtime implications either. You can see in the Repository Explorer that the Vitality column now displays “Compiled” and no X appears through the icons.

You have seen how the CA-Visual Object debugging tools allow you to quickly pinpoint and resolve compiler errors, and you will soon see the successfully compiled application in action.

Setting Debugging Options

Before running the application, however, let's take a moment to examine the debugging indicators in the various browsers. In CA-Visual Objects, you can set debugging options at any level—application, module, or entity—and override the current default setting at the next lower level. This feature enables you to debug a new application piece-meal by setting the application-level debug flag on and selectively turning off the debug flags for modules and entities that you are not currently interested in debugging.

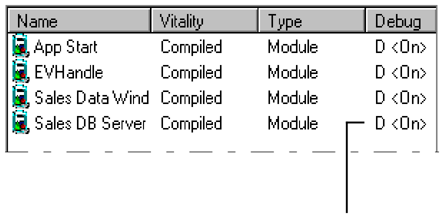
The Repository Explorer has a Debug column, which displays the debug flag status for each level in the application's hierarchy. D<On> indicates that debugging is turned on for the current level, D<Off> indicates that debugging is turned off for the current level, and D<Auto> indicates that debugging at another level will apply.

Important! *Whenever you change a debug setting, you must rebuild the application to have it take effect. Because the system only rebuilds what you have changed, the rebuild is usually instantaneous.*

Application Level

As discussed earlier in this guide, when you initially create an application, you can turn its debug flag on or off. Each module within the application then automatically inherits the application's debug flag setting.

For example, our sample Sales Totals application was created with the debug flag turned on, so all modules in that application that are created subsequently will have a "D<On>" indicator displayed in the Debug column:



Name	Vitality	Type	Debug
App Start	Compiled	Module	D <On>
EVHandle	Compiled	Module	D <On>
Sales Data Wind	Compiled	Module	D <On>
Sales DB Server	Compiled	Module	D <On>

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 and, by default, debug any entity defined in the application.

If, however, an application's debug flag is not turned on, all modules will have debugging turned off and their debugging indicators will be *Auto*. Furthermore, unless you have debugging turned on for a particular entity, 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.

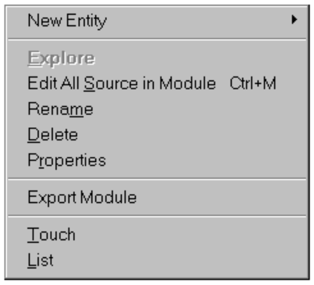
Note: At any point during the lifetime of an application in the repository, you have the option of resetting the debug flag using the Properties dialog box (accessed via the Application Properties toolbar button or menu command). See Modifying Your Application's Properties in "Using the Repository Explorer" for more information about setting application options.

Module Level

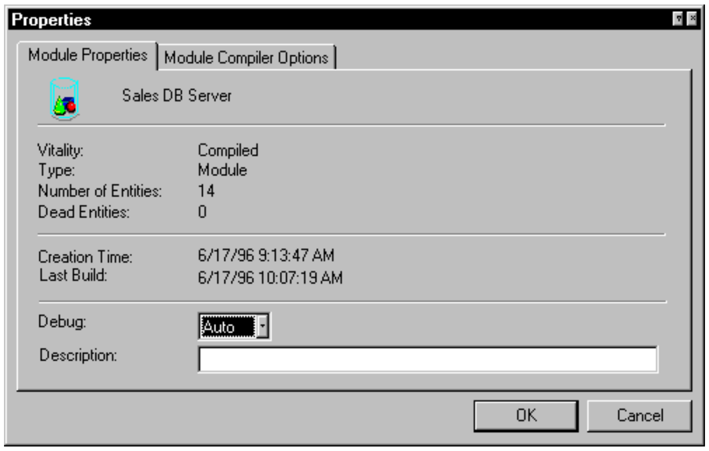
Since you may at times want to override the default debug setting for a particular module, the system allows you to change the debug setting at the module level. For instance, if you are debugging an application and are not interested in looking at entities in a particular module, you can turn the module debug flag off.

In the Sales Totals application, for example, the EVHandle module is the only one in question, so you could disable debugging for the other modules, as shown here:

1. Highlight the Sales DB Server module.
2. Click the right mouse button and choose Properties from the local pop-up menu that displays:



The Properties (Module) dialog box appears:



Sales DB Server

Vitality: Compiled
Type: Module
Number of Entities: 14
Dead Entities: 0

Creation Time: 6/17/96 9:13:47 AM
Last Build: 6/17/96 10:07:19 AM

Debug: Auto
Description:

Note that the Debug drop-down list is set to Auto, indicating that the selected module is currently using the debug flag set at the application level.

Note: This dialog box has also been changed slightly in CA-Visual Objects 2.7. The Debug drop-down list box has been replaced by three radio buttons: Auto, On, and Off. Additionally, it indicates whether source control is available or not and, if so, the source control status of the specified module. For detailed information about the new Source Code Control Interface, refer to the online help.

3. Select the Off setting to turn debugging off for this module, and then click OK.

Entity Level

CA-Visual Objects allows you to set the debug flag at the entity level in a similar manner.

To change the debug flag for an individual entity:

1. Click on the specified module to display its entities, and then highlight the specified entity (for example, Sales).
2. Click the right mouse button, and then choose Properties from the local pop-up menu that displays.

   The Properties (Entity) dialog box, which is almost identical to the Properties (Modules) dialog box, appears.
3. Choose any one of the three debugging options: Auto, On, or Off.

Important! *For the sample application, the debug flag for the EVHandle module and all of its entities must be set to On or Auto, as should the debug flag for the App:Start() entity. Make sure this is the case before continuing with the instructions in this chapter.*

Resolving Runtime Errors

Now, let's move into the next stage of debugging by running the application and looking for runtime and logic errors.

If you have changed any of the debug flags according to the instructions in the previous section, you will need to rebuild the Sales Totals application now. Then, run the application by clicking the Run toolbar button.

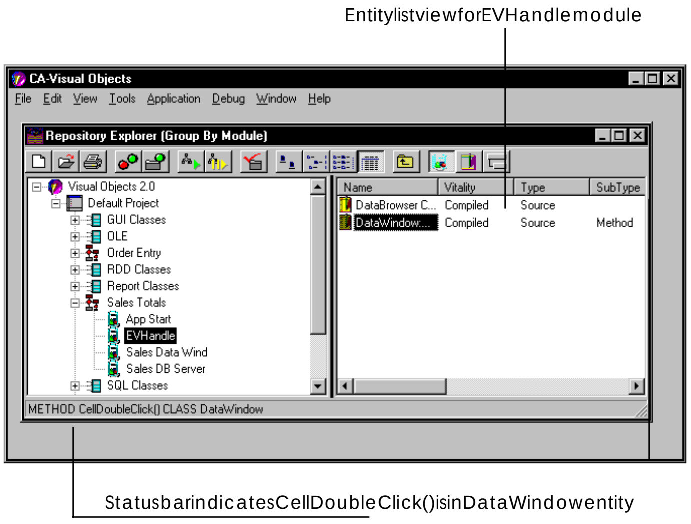
The application executes, displaying a scrolling list box containing sales figures by sales representative. The application instructs you to double-click on a sales representative to show a total amount, but when you do this, the edit control opens in edit mode.

This is because there is an error in the logic of the program preventing the double-click event from being processed correctly. This type of error is not easy to track down because there are no error messages or visual indicators in the IDE or the application to give you clues.

Using the Online Help Reference to Isolate an Error

The first thing you want to consider is the action itself. In this case, the action is a double-click. A good place to start is the CellDoubleClick() method:

1. Close the Sales Totals application by clicking on the Close button in the title bar of its shell window.
2. Bring up the entity list view for the EVHandle module by clicking on the module.
3. Find the CellDoubleClick() method, and note that it is defined for the DataWindow class:


EntitylistviewforEVHandlemodule
CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Visual Objects 2.0
Default Project
GUI Classes
OLE
Order Entry
RDD Classes
Report Classes
Sales Totals
App Start
EVHandle
Sales Data Wind
Sales DB Server
SQL Classes
StatusbarindicatesCellDoubleClick()isinDataWindowentity

Name	Vitality	Type	SubType
DataBrowser C...	Compiled	Source	
DataWindow:...	Compiled	Source	Method

METHOD CellDoubleClick() CLASS DataWindow

4. Choose the Help Index menu command to display the Index tab control in the Help window.
5. Type **CellDoubleClick** in the edit control.

   You will see in the Topics Found list box that this method is defined for the DataBrowser class, not DataWindow.
6. Choose Cancel to close the Help window and return to the Repository Explorer.

Correcting an Error Using the Source Code Editor

Now, you can easily correct this error using the Source Code Editor. To do so:

1. Double-click on the CellDoubleClick() method to show the source code for the method.
2. Change the first statement reading

   ```
   METHOD CellDoubleClick() CLASS DataWindow
   ```

   to:

   ```
   METHOD CellDoubleClick() CLASS DataBrowser
   ```

3. Choose the Save toolbar button to save your changes, and then close the Source Code Editor.
4. Click on the Build toolbar button to build the application. In this case, no information was available to help you resolve the runtime error. In some cases, however, a runtime error message dialog box will give you clues as to the source of the runtime error.

The Error Dialog Box

The level of detail in CA-Visual Objects runtime error message dialog box allows you to pinpoint the exact location of the error in your source code. It tells you the class name, entity name, and line number, as well as a description of the error.

Important! *You should record this information—this is the only place where the exact location of a runtime error is pinpointed.*

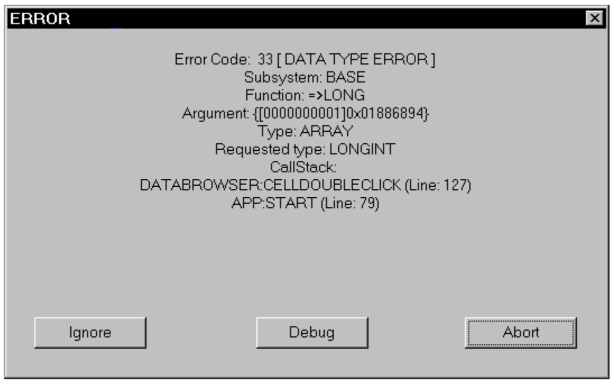
After analyzing the information in the Error dialog box, you typically have three options: Ignore, Debug, or Abort.

Ignore	The Ignore option lets you move on to the next step in the program. This typically is not recommended unless you know that ignoring the error will not adversely affect the rest of the application.
Debug	If the debug flag is on, the Debug option opens the Debug Source Code window exactly at the offending line of code. If not, the Debug Source Code window is empty; however, you can use either the various Debug menu commands or the View toolbar button to view variables, breakpoints, etc.
Abort	Use the Abort option to break out of the current error handling sequence in the application. Note that this option may or may not terminate the application, depending on how it is structured. Refer to the BEGIN SEQUENCE statement in the online help system for more information.

To see an example of the Error dialog box:

1. Run the Sales Totals application again.
2. When the application starts, double-click on a sales representative.

   The Error dialog box appears:

   Error Code: 33 [DATA TYPE ERROR]
   Subsystem: BASE
   Function: =>LONG
   Argument: {[0000000001]0x01886894}
   Type: ARRAY
   Requested type: LONGINT
   CallStack:
   DATABROWSER:CELLDOUBLECLICK (Line: 127)
   APP:START (Line: 79)

   The Error dialog box tells you that:

   - The error is a data type error
   - The method is CellDoubleClick() of the DataBrowser class
   - The legal argument type is an array, but the code shows it as numeric

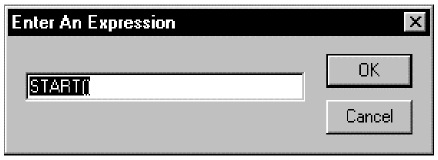
3. After recording the error information, choose Abort and close the application.

Correcting Errors Using the Debugger

To get to the bottom of this error, you will run the application again—this time using the Debugger, so that you can see what is going on in the code while the application is in progress.

1. Click the Debug toolbar button or choose the Debug Run command.

   The Enter an Expression dialog box is displayed:

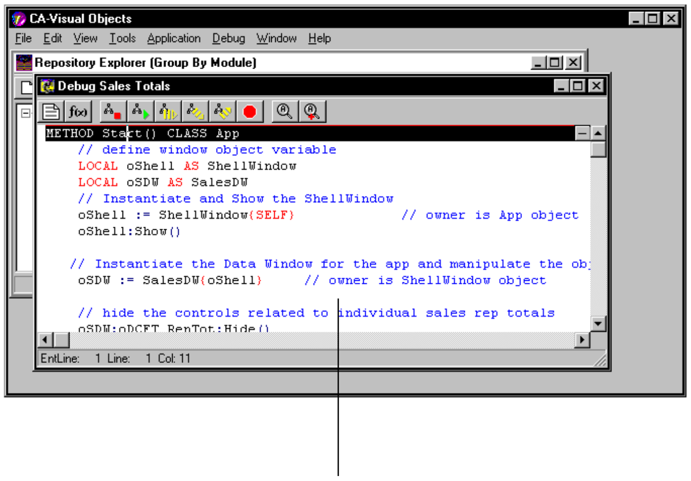
   In this dialog box, you can enter any expression that can be evaluated in the context of the current application. For example, if you have an isolated function that you want to debug, you can type its name with the necessary arguments in parentheses.

Notice that, by default, it is assumed that you want to debug the entire application; therefore, it contains the Start() entity.

Note: If the new Debug AutoStart option was selected in the System Options dialog box when you initially set up your system, then this dialog box does not appear and you are brought straight into the Debugger.

2. Choose OK.

   This launches the Debugger with the source code for the Start() method loaded in the Debug Source Code window:


CA-Visual Objects
File Edit View Tools Application Debug Window Help
Repository Explorer (Group By Module)
Debug Sales Totals
EntLine: 1 Line: 1 Col: 11

```
METHOD Start() CLASS App
    // define window object variable
    LOCAL oShell AS ShellWindow
    LOCAL oSDW AS SalesDW
    // Instantiate and Show the ShellWindow
    oShell := ShellWindow{SELF}             // owner is App object
    oShell:Show()

   // Instantiate the Data Window for the app and manipulate the obj
    oSDW := SalesDW{oShell}     // owner is ShellWindow object

    // hide the controls related to individual sales rep totals
    oSDW:oDCFT_RepTot:Hide()
```

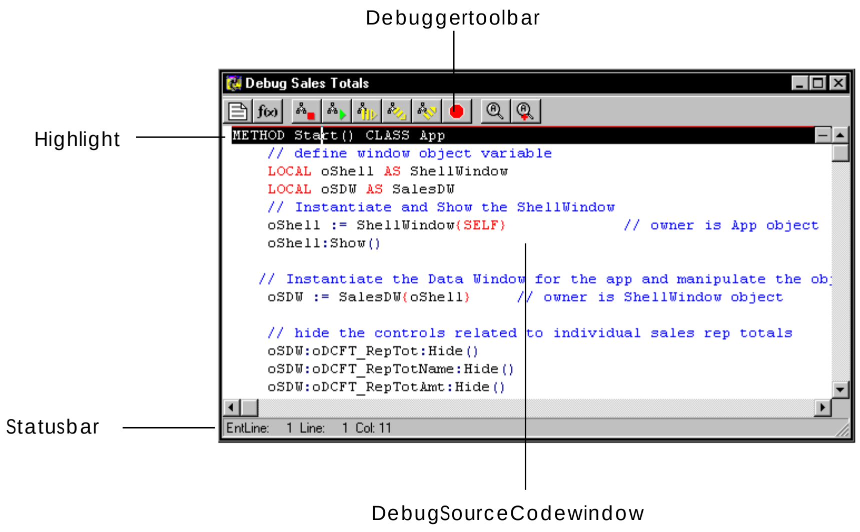
DebugSourceCodewindow

Note: The Debugger cannot load source code for any entity that does not have its debug flag turned on. Therefore, if the Start() entity has its debug flag turned off, the Debugger will not pause as indicated here. Instead, it will run the application until an entity is encountered for which the debug flag has been turned on.

The Debugger Workspace

Before moving on to debug the error in the Sales Totals application, here is a brief overview of the Debugger to familiarize you with its features.

The Debugger has its own workspace, including a menu bar and a window for displaying source code:



```
METHOD Start() CLASS App
    // define window object variable
    LOCAL oShell AS ShellWindow
    LOCAL oSDW AS SalesDW
    // Instantiate and Show the ShellWindow
    oShell := ShellWindow(SELF)                // owner is App object
    oShell:Show()

   // Instantiate the Data Window for the app and manipulate the obj
    oSDW := SalesDW(oShell)     // owner is ShellWindow object

    // hide the controls related to individual sales rep totals
    oSDW:oDCFT_RepTot:Hide()
    oSDW:oDCFT_RepTotName:Hide()
    oSDW:oDCFT_RepTotAmt:Hide()
```

```
EntLine:   1  Line:   1  Col: 11
```

When the Debugger is active, the commands on the Debug menu also become available.

Debug Source Code Window

The Debug Source Code window is the main Debugger window. It 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.

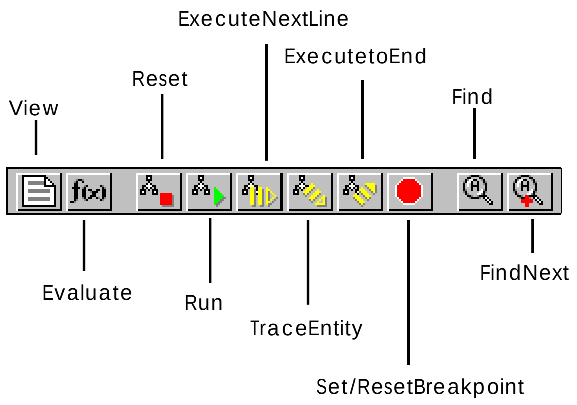
Tip: If there are other windows open when using the Debugger (for example, other Repository Explorers and editors), you may want to minimize or rearrange those windows to reduce the clutter in the CA-Visual Objects desktop. Then, you can use the Window Tile menu command to conveniently arrange the remaining open Debugger windows.

Highlight

The *highlight* in the Debug Source Code window indicates the current point of execution in the source code. You can scroll through this window using the scroll bars and position the cursor using the mouse, but the highlight can only be moved via a toolbar selection or menu command.

Toolbar

In addition to the Debug menu commands available in the CA-Visual Objects 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 Next, are discussed in this chapter. Find and Find Next are performed using standard Windows techniques, and you can refer to your online help system for further information.

Tip: For the name and a quick description of these toolbar buttons, look at the tooltips as the mouse pointer passes over the buttons.

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:

Debug Menu Command	Toolbar Button
Run	Run
Step	Execute Next Line
Step In	Trace Entity
Step Out	Execute to End
Step To Cursor	N/A
Reset Process	Reset
Breakpoint	Set/Reset Breakpoint

Each execution command is described in more detail below.

Note: Use the Reset toolbar button to stop execution and restart the application from the beginning.

Run

To execute an application in *run mode*, use the Debug Run menu command or the Run toolbar button. The application returns control to the Debug Source Code window when:

- It terminates normally
- A runtime error occurs
- A breakpoint is encountered
- 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.

Tip: Remember, you can control which entities and modules the Debugger will step into using individual debug settings at these levels. See Setting Debugging Options earlier in this chapter for more information.

Step Out

The Step Out command (or Execute to End toolbar button) executes in run mode until the end of the current entity is encountered. It behaves as if a breakpoint were set at the last statement in the entity.

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 Setting Breakpoints later in this chapter for more information on breakpoints.

Analyzing the Problem

Now that you have an overview of the Debugger workspace and execution commands, you are ready to use it to resolve the problem in the Sales Totals application.

Setting 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 online help.)

Breakpoints can also be set in the Source Code Editor by placing the cursor on a line of code and clicking on the Set/Reset Breakpoint toolbar button of the Source Code Editor. The line will be highlighted in red to denote that the breakpoint has been set.

Note: See the online help for detailed information about the System Options dialog box's new Activate on Break option. This option, if selected, automatically activates the IDE and brings it to the foreground when a debugged application reaches a breakpoint.

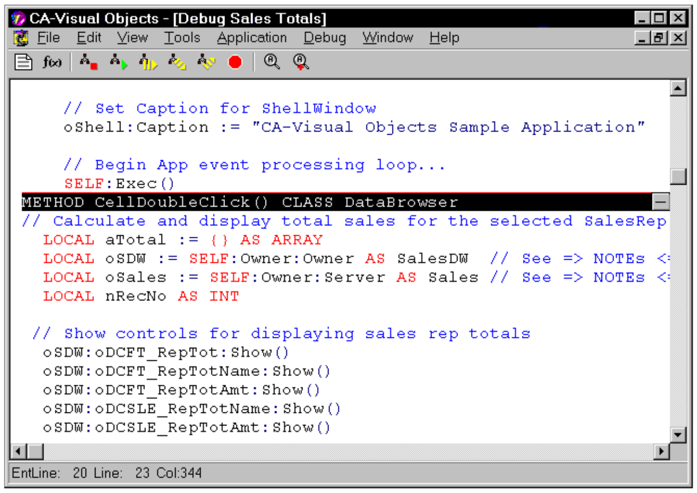
Setting a Breakpoint

For example, when you last ran the application, the program resulted in a runtime error in the CellDoubleClick() method. By loading this entity into the Debug Source Code window, you can cause the application to pause its execution and return control to the Debugger using a breakpoint.

To do this:

1. With the Debugger still running, switch to the Repository Explorer. (Tile these windows if you like.)
2. Select the EVHandle module in Sales Total to display its entity list, and then double-click on its CellDoubleClick() method.

   The source code for the method is loaded in the Debug Source Code window as shown below:


CA-Visual Objects - [Debug Sales Totals]
File Edit View Tools Application Debug Window Help

```
    // Set Caption for ShellWindow
    oShell:Caption := "CA-Visual Objects Sample Application"

    // Begin App event processing loop...
    SELF:Exec()
METHOD CellDoubleClick() CLASS DataBrowser
// Calculate and display total sales for the selected SalesRep
  LOCAL aTotal := {} AS ARRAY
  LOCAL oSDW := SELF:Owner:Owner AS SalesDW  // See => NOTEs <
  LOCAL oSales := SELF:Owner:Server AS Sales // See => NOTEs <
  LOCAL nRecNo AS INT

 // Show controls for displaying sales rep totals
  oSDW:oDCFT_RepTot:Show()
  oSDW:oDCFT_RepTotName:Show()
  oSDW:oDCFT_RepTotAmt:Show()
  oSDW:oDCSLE_RepTotName:Show()
  oSDW:oDCSLE_RepTotAmt:Show()
```

EntLine: 20 Line: 23 Col:344

   Note that this code is appended to that of the Start() module.

3. Move the cursor to the line of code reading:

   `oSDW:oDCFT_RepTot:Show()`

4. Click the Set / Reset Breakpoint toolbar button or use the Debug Breakpoint command.

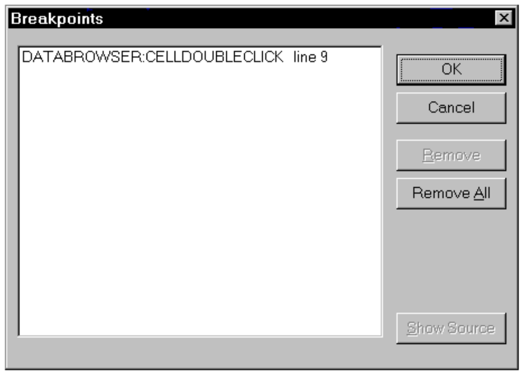
   The system highlights the specified line in red, denoting that a breakpoint is set at that location.

You can set other breakpoints using this same procedure—just click on the desired line then click Set / Reset Breakpoint. The only thing to be aware of is that you cannot load entities that do not have their debug flag set (either directly or through inheritance from the module or application).

Viewing/Clearing Breakpoints

Note also that the Set / Reset Breakpoint toolbar button 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 Breakpoints dialog box:



DATABROWSER:CELLDOUBLECLICK line 9

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.)
- Scroll to the line containing the currently selected breakpoint in this dialog box by choosing the Show Source push button

  Doing so places the execution highlight at that line in the Debug Source Code window.

To close this window, click Cancel.

Running with a Breakpoint

Once the breakpoint is set:

1. Run the application using the Run toolbar button.
2. When the application window appears, click the Restore button of its shell window and move and/or size the window so that you can view the application and Debugger windows at the same time.
3. Double-click on the first sales representative.

   This will bring you to the breakpoint you defined in the CellDoubleClick() method.
4. Click the Execute Next Line button or choose the Debug Step menu command to step through the application. Continue to step through the application until an error occurs. This will help you isolate the line of code causing the error.
5. Once the error is isolated, close the Debug Source Code window to shut down both the Debugger and the Sales Totals application.

   The Debugger is closed and you are returned to the EVHandle entity list view.

Correcting the Error

Once you have located the lines where errors occur, you are ready to fix the problems.

The lines of code identified earlier indicate that you need to define the local variable *nTotal* as an array. Double-click on the CellDoubleClick() method to load it into the Source Code Editor, and make the following corrections:

1. Change the line

   ```
   LOCAL nTotal AS INT
   ```

   so that it reads:

   ```
   LOCAL aTotal := {} AS ARRAY
   ```
2. Change the line

   ```
   nTotal := oSales:Sum(#Amount,{||...
   ```

   so that it reads:

   ```
   aTotal := oSales:Sum(#Amount,{||...
   ```
3. Change the line

   ```
   oSDW:SLE_RepTotAmt := nTotal
   ```

   so that it reads:

   ```
   oSDW:SLE_RepTotAmt := aTotal[1]
   ```
4. Close the Source Code Editor window, saving your work.

   The error is corrected.

5. Rebuild the application.

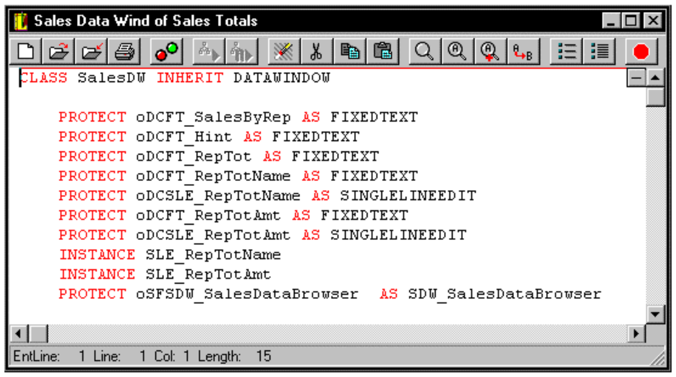
More Debugging

If you run the application, you will not get another error message; however, you will find that nothing displays in the “Totals for Sales Rep” single-line edit controls when you double-click on a sales representative. This section of the application is controlled by single-line edit controls.

To get to the root of the problem, you need to find out what variable names are used for the single-line edit controls:

1. If you are still running the Sales Totals application, close it now.
2. Click on the Sales Data Wind module, and then click on the Group By Type toolbar button to display a listing of its entities.
3. Click on the Access entity in the Repository Explorer’s tree view pane.

   You will see the entity names, SalesDW:SLE_REPTOTAMT and SalesDW:SLE_REPTOTNAME, listed. Each is defined as an access.
4. Similarly, click on the Assign entity in the Repository Explorer’s tree view pane. You will now see the entity names, SalesDW:SLE_REPTOTAMT and SalesDW:SLE_REPTOTNAME, listed.
5. If you click on the Class entity in the Repository Explorer’s tree view pane and then double-click on the SALESDW class, you will also see SLE_REPTOTAMT and SLE_REPTOTNAME displayed as instance variables in the Souce Code Editor:



```
CLASS SalesDW INHERIT DATAWINDOW

    PROTECT oDCFT_SalesByRep AS FIXEDTEXT
    PROTECT oDCFT_Hint AS FIXEDTEXT
    PROTECT oDCFT_RepTot AS FIXEDTEXT
    PROTECT oDCFT_RepTotName AS FIXEDTEXT
    PROTECT oDCSLE_RepTotName AS SINGLELINEEDIT
    PROTECT oDCFT_RepTotAmt AS FIXEDTEXT
    PROTECT oDCSLE_RepTotAmt AS SINGLELINEEDIT
    INSTANCE SLE_RepTotName
    INSTANCE SLE_RepTotAmt
    PROTECT oSFSDW_SalesDataBrowser  AS SDW_SalesDataBrowser
```

   These are the variable names for the single-line edit controls in this application. Note the names of these variables, as you will use them later in the debugging process.
6. Close the Source Code Editor.

7. Click on the Group By Module icon to display the Sales Totals application's modules.

8. Click the Debug toolbar button to run the application using the Debugger. (If prompted to enter an expression, enter Start() and then choose OK.)

   The Debugger opens.

Running with a Preset Breakpoint

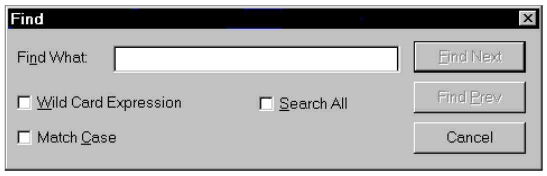
In the Sales Totals application, the CellDoubleClick() method is responsible for displaying information in the single-line edit controls. To view this method using the Debugger and examine the code that calculates the single-line edit control values:

1. From within the Repository Explorer, click on the EVHandle module to display its entity listing.
2. Double-click on the CellDoubleClick() method.

   The source code for this method is added to the Debug Source Code window.

3. From within the Debugger, click the Find toolbar button.

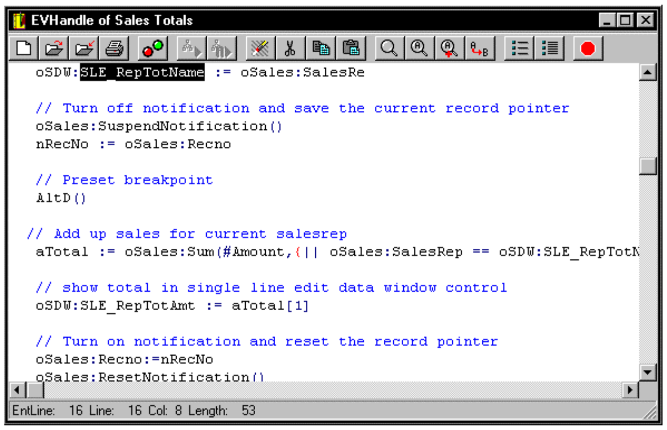
   The Find dialog box appears:



   Note: The Find dialog box now displays its search history via a drop-down list box. This new feature makes it easier to repeat a recent search for specified text. For more information, see the online help.

4. Type **SLE_REPTOTNAME** in the Find What edit control.
5. Click Find Next twice, and then choose Cancel.

The cursor is now located on the line of code that caused the compiler warning earlier:



```
   oSDW:SLE_RepTotName := oSales:SalesRe

   // Turn off notification and save the current record pointer
   oSales:SuspendNotification()
   nRecNo := oSales:Recno

   // Preset breakpoint
   AltD()

  // Add up sales for current salesrep
   aTotal := oSales:Sum(#Amount,{|| oSales:SalesRep == oSDW:SLE_RepTotN

   // show total in single line edit data window control
   oSDW:SLE_RepTotAmt := aTotal[1]

   // Turn on notification and reset the record pointer
   oSales:Recno:=nRecNo
   oSales:ResetNotification()
```

EntLine: 16 Line: 16 Col: 8 Length: 53

This line of code involves one of the single-line edit controls, so it is probably the source of the problem.

6. Scroll down until you locate the AltD() statement.

   AltD() is a function that invokes the Debugger automatically, just as if you had set a breakpoint. We intentionally put this preset breakpoint in the sample application to invoke the Debugger.

   Note: Before executing the application, the breakpoint that was set earlier in this chapter must be removed. Place the cursor on the line with the breakpoint and click on the Set/Reset Breakpoint toolbar button. Alternatively, select the Debug View Breakpoints menu command, and then click on the Remove All button in the View Breakpoints dialog box.

7. To see how it works, click the Run toolbar button to launch the application.

8. When the application window appears, click the Restore button of its shell window and move and/or size the window so that you can view the application and Debugger windows at the same time.

9. Double-click on a sales representative, and you will see the highlight in the Debug Source Code window move to the AltD() statement.

Viewing Work Areas

Highlight the Sales Total application in the Repository Explorer's tree view pane, and then choose the Error Browser command from the Tools menu to refresh your memory about the warning message received earlier. It tells you that the variable named SALESRE does not exist or is not accessible. The line of code that produced this message looks like this:

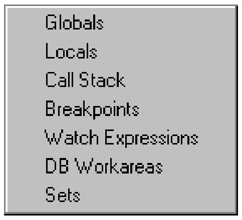
```
oSDW:SLE_RepTotName := oSales:SalesRe
```

Thus, SalesRe is supposed to be a field defined for the data server, oSales. To get more information about the fields defined for this data server, you can open the *Database Work Area window*. 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. Close the Error Browser, if it is open, and click on the Debugger's View toolbar button.

   A local pop-up menu appears:

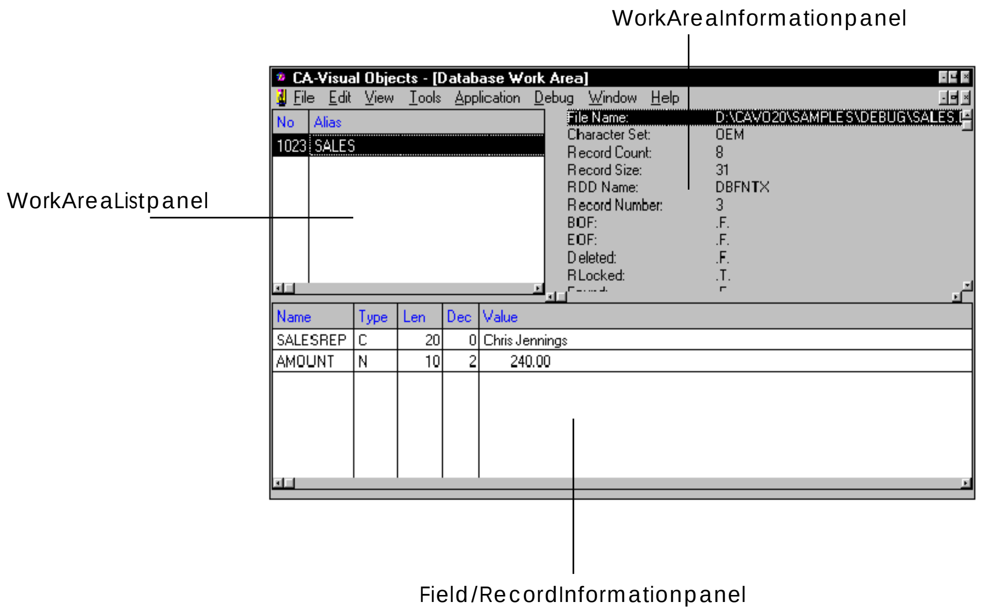
   

2. Choose the DB Workareas command.

   Alternatively, choose the View DB Work Areas command from the Debug menu.

   The Database Work Area window appears.

3. Click on the SALES alias to display the server information, as follows:


WorkAreaInformationpanel
WorkAreaListpanel
Field/RecordInformationpanel
File Edit View Tools Application Debug Window Help

CA-Visual Objects - [Database Work Area]

No	Alias
1023	SALES

File Name:	D:\CAVO20\SAMPLES\DEBUG\SALES.
Character Set:	OEM
Record Count:	8
Record Size:	31
RDD Name:	DBFNTX
Record Number:	3
BOF:	.F.
EOF:	.F.
Deleted:	.F.
RLocked:	.T.

Name	Type	Len	Dec	Value
SALESREP	C	20	0	Chris Jennings
AMOUNT	N	10	2	240.00

Note that at this point, SALES.DBF is in use by your sample application. This database file is being accessed by the oSales data server.

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

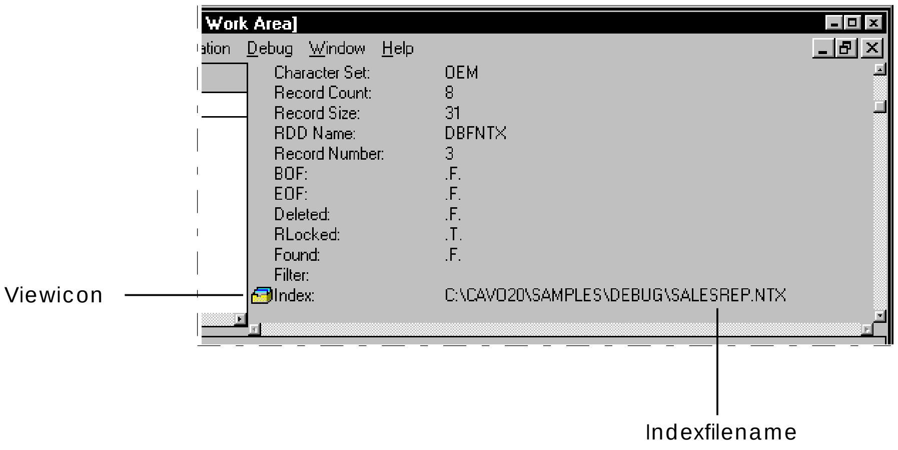
The Work Area List panel displays a list of the work area number and alias for each database file currently in use by the application. This list contains one highlighted entry, which represents the database file about which this window is displaying information.

If there are multiple work areas in use, 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

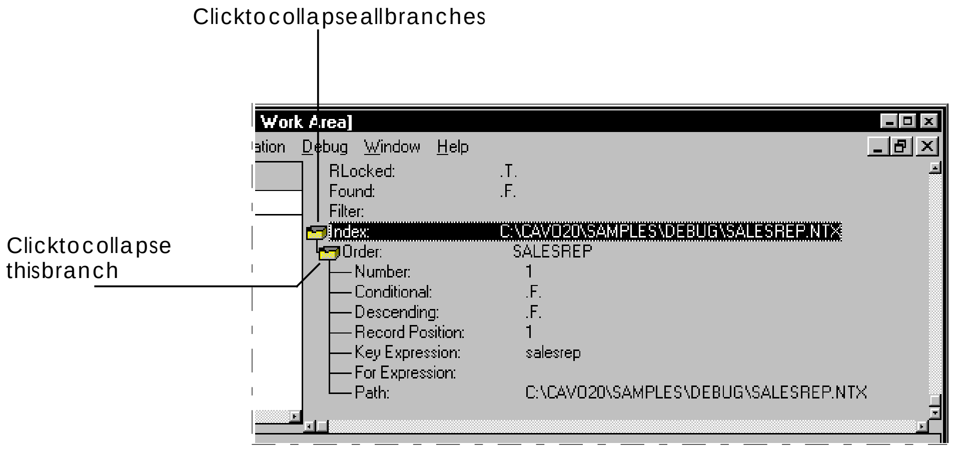
The Work Area Information panel displays a host of data about the selected work area, including beginning and end of file status, current record number, and filter. Of particular interest in this panel is the Index setting (you may need to scroll the window down to see it).

When you first open this window, the Index setting displays a single index file name to the right and a *View icon* to the left, as shown below. (If there are multiple index files, each one will have its own icon.)



```
Character Set:    OEM
Record Count:     8
Record Size:      31
RDD Name:         DBFNTX
Record Number:    3
BOF:              .F.
EOF:              .F.
Deleted:          .F.
RLocked:          .T.
Found:            .F.
Filter:
Index:            C:\CAVO20\SAMPLES\DEBUG\SALESREP.NTX
```

If you click on this View icon, more detailed information about the index file (and the orders in the file) is displayed in a collapsible / expandable tree-like structure:



```
RLocked:          .T.
Found:            .F.
Filter:
Index:            C:\CAVO20\SAMPLES\DEBUG\SALESREP.NTX
  Order:            SALESREP
    Number:           1
    Conditional:      .F.
    Descending:       .F.
    Record Position:  1
    Key Expression:   salesrep
    For Expression:
    Path:             C:\CAVO20\SAMPLES\DEBUG\SALESREP.NTX
```

You can manipulate this tree to customize the view.

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 “SalesRe” in the SALES.DBF file—rather, it is called “SalesRep.”

Implementing a Temporary Fix

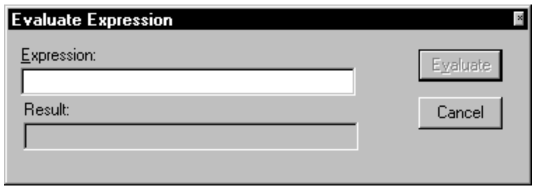
You can evaluate any expression (except one involving dimensioned arrays) from within the Debugger using either the Expression command on the Debug menu or the Evaluate toolbar button. This is a convenient way to change the current value of a variable using the assignment operator or simply to view the contents of a variable.

You will use this feature now to implement a temporary correction for the error involving the SalesRep field. This will enable your program to display the totals for the current sales representative. Later, you will need to make a permanent correction using the Source Code Editor.

To implement the temporary fix:

1. Close the Database Work Area window.
2. Click the Evaluate toolbar button or choose the Debug Expression command:

   The Evaluate Expression dialog box appears:

3. Enter the following expression in the Expression edit control:

   ```
   oSDW:SLE_RepTotName := oSales:SalesRep
   ```

   Note: In CA-Visual Objects 2.7, this dialog box displays its expression evaluation history via the Expression drop-down list box. This new feature makes it easier to repeat a recent expression evaluation. Additionally, the Result edit control has multiple lines and is scrollable. For more information, see the online help.

4. Click Evaluate.

   The name of the current sales representative displays in the Result edit control. This action evaluates the line of code that you entered, which is a correction for the line we suspect is causing the application to fail.

5. Close the Evaluate Expression dialog box by choosing Cancel.
6. Click the Run toolbar button to continue running the application after the preset breakpoint.

   The name and total should now display properly.

7. Return to the Debug Source Code window and close it to shut down the Debugger and the sample application. Then implement the permanent fix, described below.

Correcting the Final Error

This is the final error in the sample application, and you can now correct it.

1. If it is not already open, access the entity list for the EVHandle module.
2. Double-click on the CellDoubleClick() method to open the Source Code Editor.
3. Change the line that reads

   ```
   oSDW:SLE_RepTotName := oSales:SalesRe
   ```

   to:

   ```
   oSDW:SLE_RepTotName := oSales:SalesRep
   ```

4. Close the Source Code Editor, saving your changes.
5. Rebuild the application.

   This time, the status bar will not show any warnings.
6. Run the application.

   Since all errors have been corrected, the application should now work as intended.

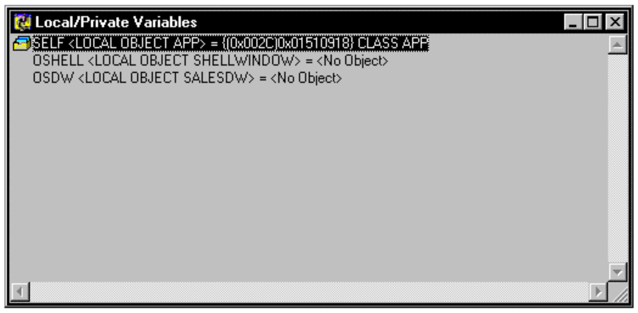
Even though there are no more errors, the remaining features of the Debugger will be demonstrated using this application, so close it now and run it again using the Debugger as you have already done several times (that is, click the Debug toolbar button and choose OK if prompted to evaluate the Start() expression).

Viewing Local and Private Variables

To view local, static local, and private variables within an entity:

1. Select the Debugger's View toolbar button.
2. Select the Locals command from the local pop-up menu.

   The Local / Private Variables window appears:



```
SELF <LOCAL OBJECT APP> = {(0x002C)0x01510918} CLASS APP
OSHELL <LOCAL OBJECT SHELLWINDOW> = <No Object>
OSDW <LOCAL OBJECT SALESDW> = <No Object>
```

If there are multiple variables displayed in this window, you can highlight a new one by clicking on it.

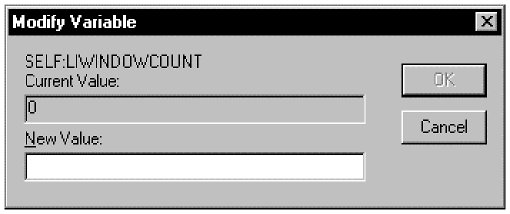
For multi-component variables, such as arrays, objects, and structures, the View icon to the left of the variable expands / collapses the variable to show / hide its components. For example, click on the icon to the left of SELF to view its components. You can then select the LIWINDOWCOUNT protect variable by clicking on it.
Normally, this window displays the variables that are visible to the currently executing entity. If the Call Stack window has focus, however, this window will show the variables for the currently highlighted entity in the call stack. See Viewing the Call Stack for more information.

Modifying Local and Private Variables

By right-clicking on the current variable, you can access a local pop-up menu with three commands for modifying variables:

Modify Variable

This command allows you to change the value of the highlighted variable using the Modify Variable dialog box:



SELF:LIWINDOWCOUNT
Current Value:
0
New Value:

In it, you can enter a simple value or an expression for the new value. (See Viewing Sets later in this chapter for an example.) Note that this option is not available for multi-component variables, such as arrays and objects.

Watch Variable

This command sets the currently highlighted variable as a watch expression. (See Using Watch Expressions later in this chapter.)

Set Range

This command allows you to specify a range of elements to be displayed for an array variable using the Change Array Index Range dialog box.

Tip: This local pop-up menu is also available for variables displayed in the Global/Public Variable and Call Stack windows (discussed in Viewing Global and Public Variables and Viewing the Call Stack, respectively, later in this chapter.)

Viewing Global and Public Variables

There is a separate window available for viewing globals, static globals, and public variables. Open the Global/Public Variables window by clicking the Debugger's View toolbar button and then selecting the Globals command from the local pop-up menu.

The behavior of this window is identical to the View Local/Public Variables window. For example, if you click on a variable, you can see its components.

See Viewing Local and Private Variables earlier in this chapter for more information.

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.

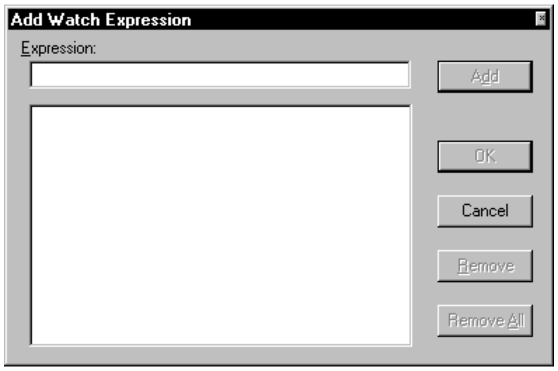
Setting and Clearing Watch Expressions

Setting

To set a watch expression that you can subsequently view in the Watch Expression window:

1. Choose the Debug Watch Expression command.

   The Add Watch Expression dialog box appears:



2. In the Expression edit control, enter the expression to be monitored (for example, **aTotal[1]**).
3. Choose Add to add it to a list of watch expressions.
4. Repeat steps 2 and 3 to create as many watch expressions as you like.
5. Choose OK to close the dialog box and save the list of watch expressions.

   The Watch Expression window appears, displaying all defined watch expressions.

Clearing

To clear one or more watch expressions:

1. Choose the Debug Watch Expression command.
2. Highlight the expression you want to remove, and click the Remove button.
3. To clear all watch expressions, click the Remove All button.

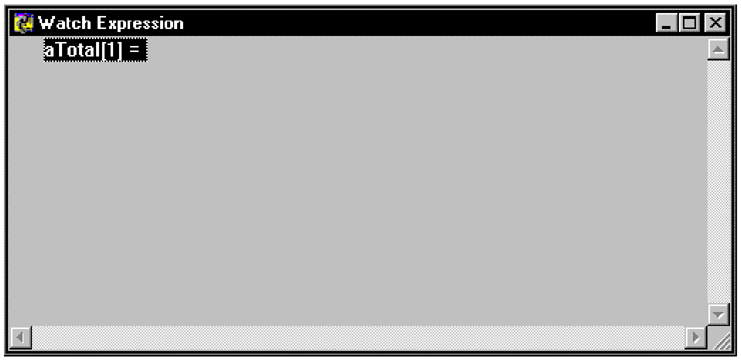
4. Choose OK to close the dialog box and save the changes you have made to the list of watch expressions.

Tip: You can also create watch expressions from variables in the Local/Private Variables, the Global/Public Variables, and the Call Stack windows by right-clicking on the specified variable and then selecting the Watch Variable command from the local pop-up menu.

Viewing Watch Expressions

To view the current value of all watch expressions, use the Debugger's View toolbar button and select the Watch Expressions command from the local pop-up menu.

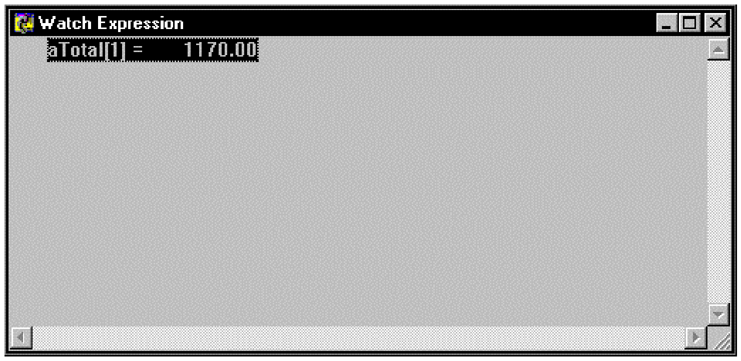
The Watch Expression window appears:



aTotal[1] =

Note: If the expression cannot be evaluated, an expression error message is displayed instead of the value of the expression.
You can watch the contents of this window change as the expressions are updated by the application. To do this, make sure that both the Debug Source Code and the Watch Expression windows are visible (you may want to use the Window Tile command to do this), and single step through the code.

For example, in the Sales Totals application, aTotal[1] changes as soon as the line of code initializing the array is executed within the CellDoubleClick method of the DataBrowser class:



```
aTotal[1] =      1170.00
```

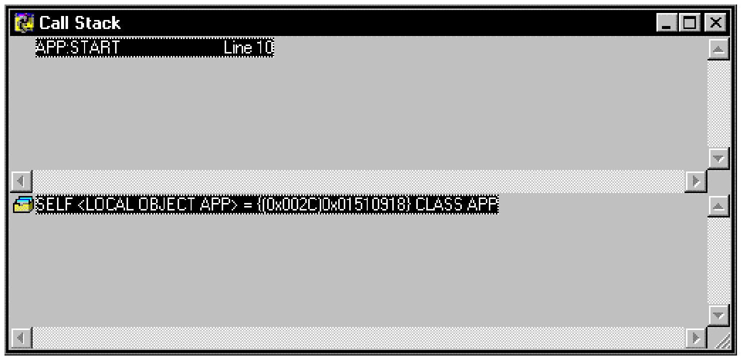
Viewing the Call Stack

A *call stack* is a list of all activations that are currently pending on the stack. When a routine, such as a function or method, 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 Total application:

1. Click 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 entity first:



```
APP:START                Line 10
SELF <LOCAL OBJECT APP> = {(0x002C)0x01510918} CLASS APP
```

Note that this window is divided into two panels, each of which has independent scroll bars. In the top panel of the Call Stack window, you can move the highlight to any entity listed by clicking on it. The bottom panel shows the name, data type, and current value for each of the parameters of the highlighted entity.

If the View Locals / Privates window is also open, it reflects the variables visible to the highlighted entity. Double-clicking on an entity will bring it into the Debug Source Code window.

Tip: By right-clicking on one of the parameters in the bottom panel, you can access a local pop-up menu with three commands for modifying variables. This menu is the same as the local pop-up menu available for the Local / Private Variables window. See Viewing Local and Private Variables earlier in this chapter for information about the available menu commands.

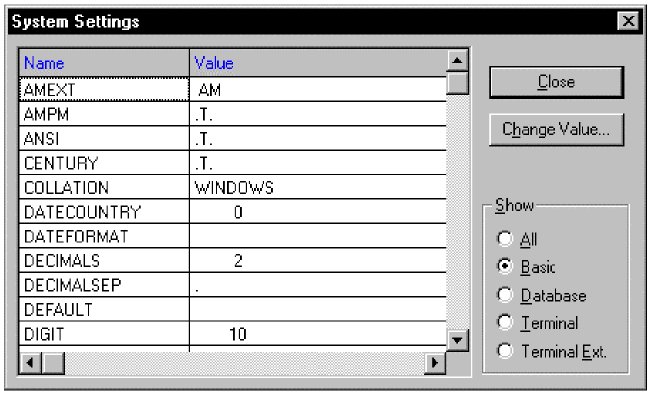
Viewing Sets

You can also view and modify the system settings for your application using the System Settings dialog box. This dialog box displays the current system settings 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 System Settings dialog box appears:



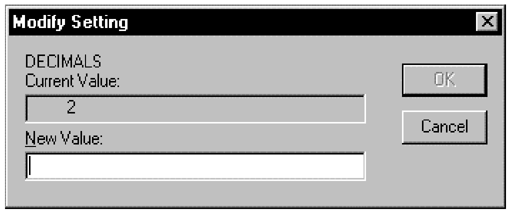
Name	Value
AMEXT	AM
AMPM	.T.
ANSI	.T.
CENTURY	.T.
COLLATION	WINDOWS
DATECOUNTRY	0
DATEFORMAT	
DECIMALS	2
DECIMALSEP	.
DEFAULT	
DIGIT	10

Note: This dialog box has been updated slightly in CA-Visual Objects 2.7. Specifically, the Terminal and Terminal Ext. options are no longer available.

3. Highlight the name of a setting (for example, DECIMALS).

4. Click the Change Value push button.

   The Modify Setting dialog box appears:

5. In the New Value edit control, enter a value (for example, **5**).
6. Choose OK.

   You are returned to the System 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 application's source code (for example, include SetDecimal(5) in your App:Start() method)

Note: The options in the Show group box in the System Settings dialog box allow you to define the level of settings to be accessed.

Other Debugging Techniques

CA-Visual Objects offers other debugging techniques, including "Just-In-Time" debugging and a terminal window.

Just-In-Time Debugging

With Just-In-Time debugging, the Debugger will be invoked if a runtime error has occurred while running the application from within the IDE. The application does not necessarily need to have the debug flag turned on to utilize the Just-In-Time debugging technique. If debug is turned on, the line of code that caused the error will be displayed. If debug is not turned on, the Debugger will be invoked but the offending line of code will not be displayed. However, the call stack and variable information will be available.

Terminal Window

CA-Visual Objects provides a terminal window that can be used for logging or verification of data. By including the Terminal Lite library in an application's properties, a terminal window will be displayed at runtime. The following are some of the functions and commands supported in the terminal window:

?	SetAlternate()
??	SetPrinter()
INKEY()	SetConsole()
WAIT	SetColor()

Chapter 12

Importing and Exporting Applications

This chapter describes how to import and export applications, modules, and several types of source files relating to an application and its components. You can import and export the following types of files:

- Application export files
- Module files
- Text-based source files

Tip: CA-Visual Objects also allows you to import and export database and index files. For information about working with these files, see **Chapter 7: Defining Data Servers** and Field Specifications earlier in this guide.

Note: CA-Visual Objects 2.7 has a new reference feature, the Recent Imports List, which allows you to view all applications (.AEF files) that have been imported recently. To access the Recent Imports List, choose the Recent Imports command from the File menu.

Exporting Applications and Modules

You can export any application or module so that it is comprised in its entirety in a single .AEF or .MEF file, respectively. This feature might be used to copy an application or module to disk so that it can be copied onto another PC. You can also export all the applications you have created in all your projects with a single command. This allows you to easily create backups of your repository.

The information denoting an application's type—application, library, or DLL—and any related information is preserved when the application is exported as an .AEF file. If there are any associated external source code or .DLL files, the code in those external source files, as well as information regarding the locations of the .DLL files on disk, is also included in the export file.

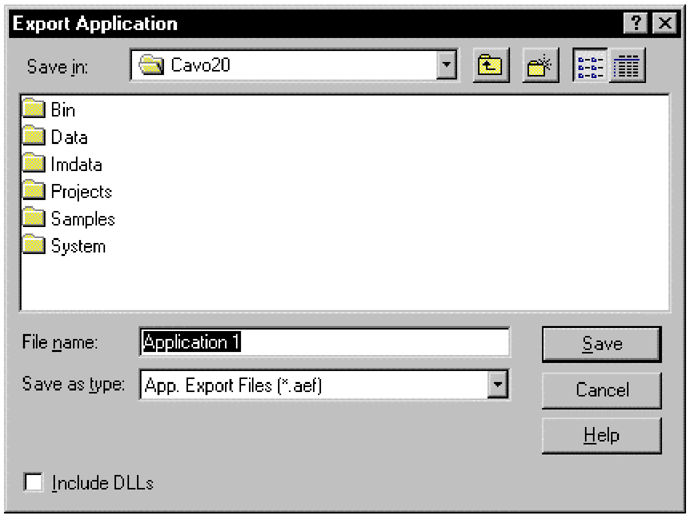
Exporting Applications

Single Application

To export a single CA-Visual Objects application as an .AEF file:

1. Choose the Export command from the File menu in the Repository Explorer.

   The Export Application dialog box appears:



Save in: Cavo20
File name: Application 1
Save as type: App. Export Files (*.aef)
☐ Include DLLs

2. Select a directory folder from the Save In drop-down list box.

   Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory.

   Use the New Folder toolbar button to create a new directory folder, if necessary.

3. If you do not want to use the default file name (for example, Order Entry), enter the name of a new or existing file in the File Name edit control for the exported application.

   Note: You will be prompted to confirm before overwriting an existing file.

4. Optionally, include all .DLL files used by the application by selecting the Include DLLs check box.

   Note: If this option is selected, any _DLL declarations referring to Windows system DLLs (for example, USER32.DLL) will be included, too, which is unnecessary and can increase substantially the size of your .AEF file.

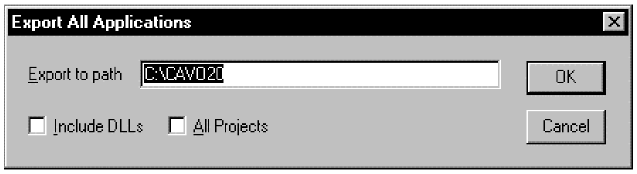
5. Choose Save.

All Applications

To export all user-created applications as .AEF files:

1. Choose the Export All command from the File menu from the Repository Explorer.

   The Export All Applications dialog box appears:

2. Enter the drive and path for the exported files in the Export to Path edit control.

   Note: By default, the exported files have the same names as those used during the last export, but the path of the last export is ignored. Instead, the .AEF files are placed in the directory that is specified here in the Export All Applications dialog box.

3. Optionally, include all .DLL files used by the applications by selecting the Include DLLs check box.

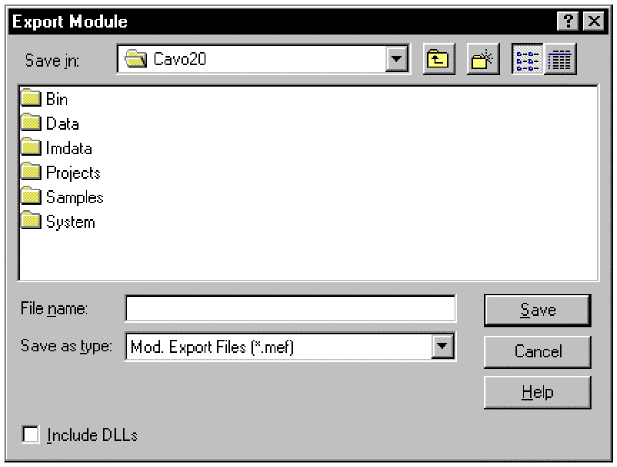
4. Optionally, include all applications associated with all projects by selecting the All Projects check box.

5. Click OK.

Exporting a Module

Similarly, to export a module as an .MEF file, highlight it and choose the Export command from the File menu.

The Export Module dialog box appears, allowing you to either specify a new name in the File Name edit control for the exported module or choose an existing .MEF file:



You can also include or exclude DLLs using the Include DLLs check box.

Note: If you choose an existing .MEF file, you will be prompted to confirm before overwriting the file.

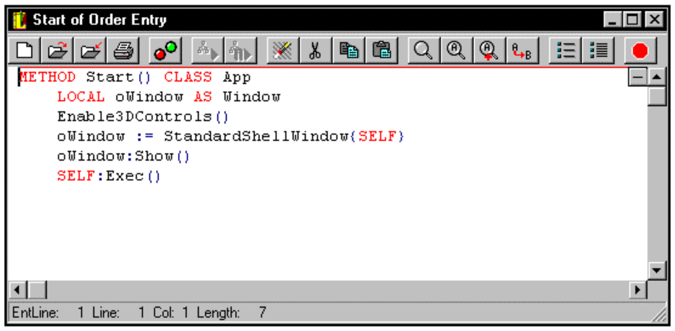
After selecting your export options, choose Save to export the specified module.

Exporting Source Files

Source code can be exported *only* from within the Source Code Editor. To export source code as a text-based source file:

1. Open a Source Code Editor window so that it contains the code to be exported.

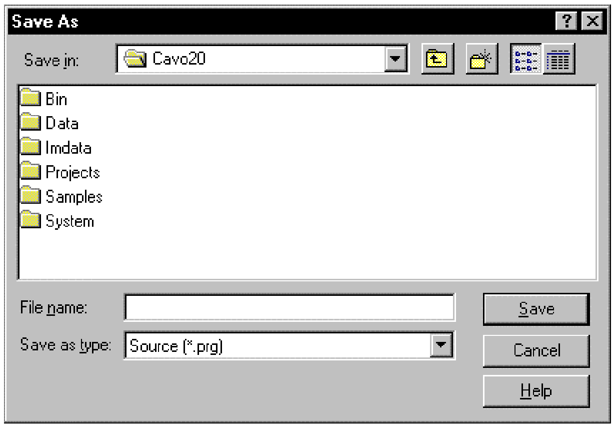
   For example, double-click on App:Start in the Repository Explorer's list view pane when the Order Entry application is selected. The following Source Code Window appears:



```
METHOD Start() CLASS App
    LOCAL oWindow AS Window
    Enable3DControls()
    oWindow := StandardShellWindow(SELF)
    oWindow:Show()
    SELF:Exec()
```

2. Choose the Export command from the File menu.

   A standard Save As dialog box appears:



3. Select the desired drive and directory.

Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory.

Use the New Folder toolbar button to create a new directory folder, if necessary.

4. Enter the name of a new or existing .PRG file in the File Name edit control for the exported source file.

   Note: You will be prompted to confirm before overwriting an existing file.

5. Choose Save.

Importing Applications and Modules

All exported applications are stored as .AEF files, including libraries and DLLs; exported modules are stored as .MEF files. You can import either type of file whenever you are in the Repository Explorer using the File Import menu commands.

The information denoting an application's type—application, library, or DLL—and any related information is preserved when importing an .AEF file.

Important! *When you export an application or module that has an external source code library or .DLL files associated with it, the code in those external files is included in the export file, as well as information regarding the locations of the files on disk. If you import the file on any machine that does not have the same directory structure, the external files are recreated on the disk from which you import (which may be a floppy disk). You need to ensure that enough disk space exists for these files; otherwise import errors can occur.*

Import Options

Both applications and modules can be imported with the same options described below.

Application or Module Name

The name of a CA-Visual Objects application or module is the text that represents the application or module in the Repository Explorer.

Note: Application and module names can be up to 30 characters in length.

By default, the name of an exported application or module is stored in its export file. When you import an .AEF or .MEF file, the name of the application or module is read from the file and placed in the Repository Explorer's tree structure.

If you wish, when importing the .AEF or .MEF file, you can optionally specify either a new name or the name of an existing application or module.

Specifying a *new name* allows you to change the name of the application or module to something other than the name stored in the imported file. This is useful either for simply renaming the application or module, or to avoid deleting, or *overwriting*, an existing application or module that has the same name as the one in the imported file.

Conversely, giving an export file the same name as an *existing* application or module would allow you to quickly replace that existing application or module with the newly imported version. For example, if someone gives you an updated version of a module that you already have on your system, you can quickly replace the out-of-date version using this technique.

Note: When importing multiple applications, Application Name is ignored.

Open as Read-Only

If selected, the imported application / module cannot be changed. Note that any further export operation will preserve this flag.

Overwrite Protection

If you want to safeguard against deleting existing applications or modules when importing, CA-Visual Objects provides a Delete Old Application option that can be turned off to prevent overwriting any existing, same-named application or module. If you choose not to delete the old application, you will be prompted to make a decision if there is a conflict during the import process.

Building the New Application or Module

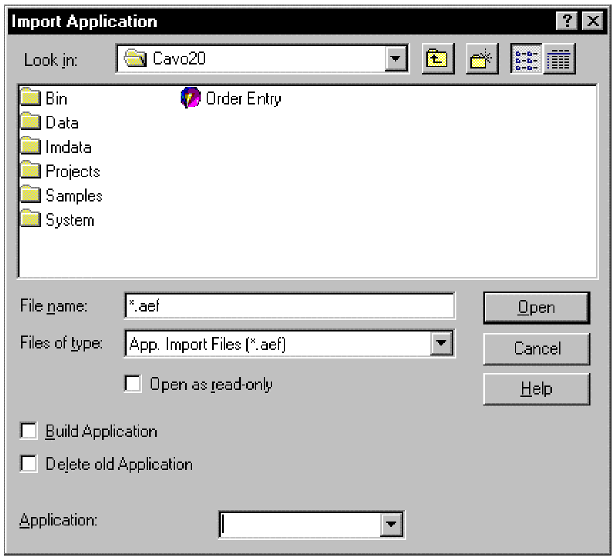
When importing an application or module, you can specify whether to immediately build the imported file as soon as it is imported. If not, the imported application or module is not compiled until you explicitly build it.

Importing an Application

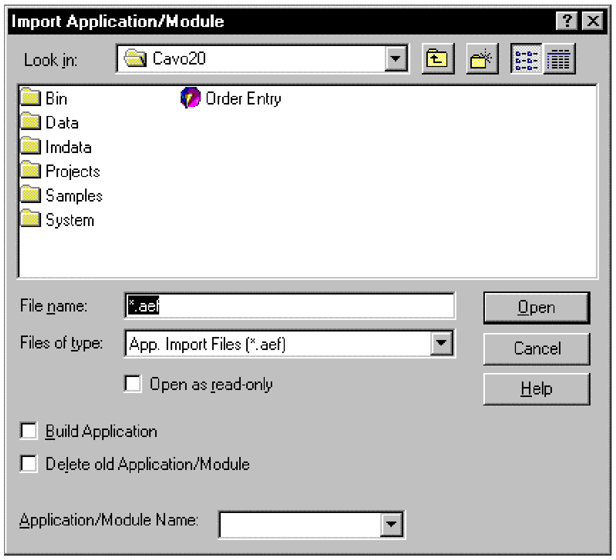
To import one or more CA-Visual Objects applications:

1. Choose the Import command from the File menu.

   At the project level, the Import Application dialog box appears:



   At the application level, the Import Application / Module dialog box appears:

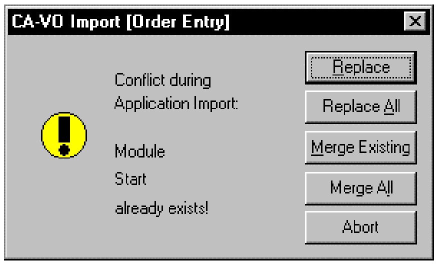


2. Select a directory folder from the Look In combo box.

   Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory.

Use the New Folder toolbar button to create a new directory folder, if necessary.

3. Choose one or more applications (.AEF files) to be imported. Hold down Shift to select multiple contiguous file names; hold down Ctrl to select multiple non contiguous file names.
4. If you are importing a single application, select the name of an existing application from the Application Name combo box, or optionally enter a new name of up to 30 characters.
5. Specify whether to immediately build the imported application(s) using the Build Application check box.
6. Specify whether to delete existing application(s) of the same name using the Delete Old Application check box. If you do not check this box, you will be prompted by a dialog box—similar to the one shown below—regarding conflicts during the import process:


CA-VO Import [Order Entry]

Conflict during Application Import:

Module
Start
already exists!

(See the online help for detailed information about this dialog box's options.)

7. Choose Open.

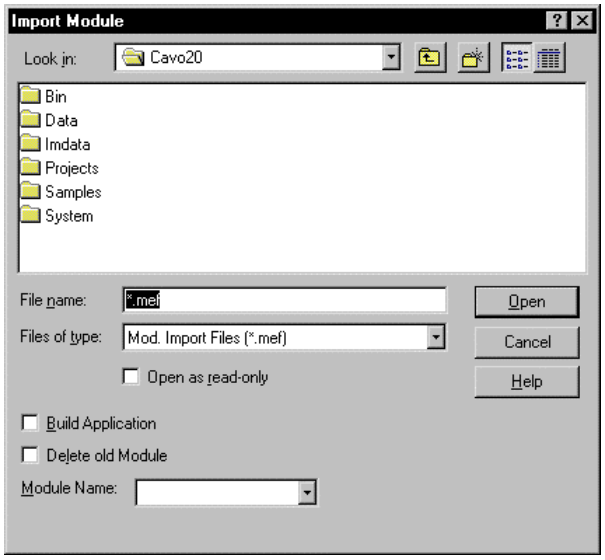
CA-Visual Objects imports the selected application(s).

Importing a Module

To import one or more CA-Visual Objects modules:

1. Choose the Import command from the File menu.

   The Import Module dialog box appears:

   Look in: Cavo20

   File name: *.mef

   Files of type: Mod. Import Files (*.mef)

   Note: At the application level, the Import Application / Module dialog box appears instead.

2. Select a directory folder from the Look In combo box.

   Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory.

   Use the New Folder toolbar button to create a new directory folder, if necessary.

3. Choose one or more modules (.MEF files) to be imported. Hold down Shift to select multiple contiguous file names; hold down Ctrl to select multiple non contiguous file names.

4. If you are importing a single module, select the name of an existing module from the Module Name combo box, or optionally enter a new name of up to 30 characters.

5. Specify whether to immediately build the imported module using the Build Application check box.

6. Specify whether to delete an existing module of the same name using the Delete Old Module check box. If you do not check this box, you will be prompted regarding conflicts during the import process.

7. Choose Open.

   CA-Visual Objects imports the selected module(s).

Importing Source Files

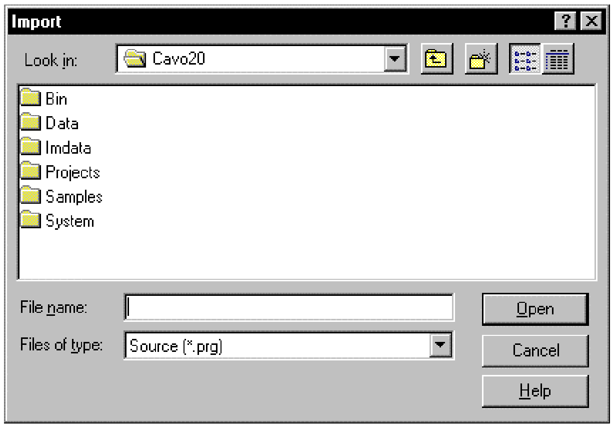
Text-based source files can be imported *only* from within the Source Code Editor.

Note: If you want the source code to be imported as its own module, create a new module before importing the source file.

To import source code from a text-based source file:

1. Choose the File Import menu command while in a Source Code Editor window.

   A standard Import dialog box appears:

2. Select a directory folder from the Look In combo box.

   Use the Up One Level and Details toolbar buttons, respectively, in your search for the desired drive and directory.

   Use the New Folder toolbar button to create a new directory folder, if necessary.

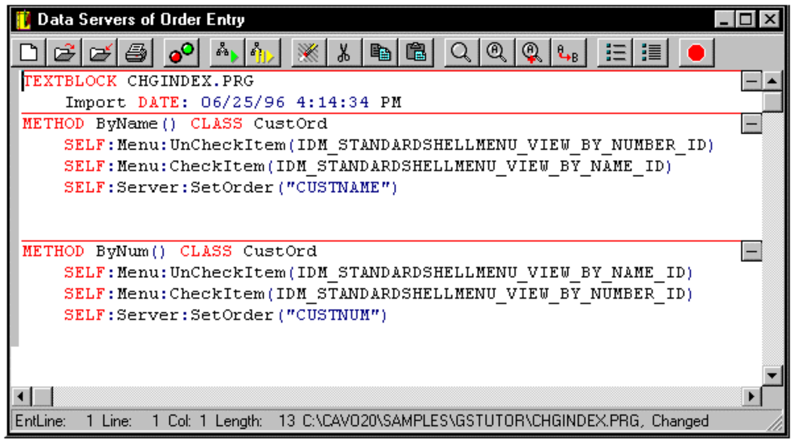
3. Select the desired file.

   Note that the default extension for imported source files is .PRG, but you can import *any* text-based source code file. For example, you can click the down arrow button in the List Files of Type box and change the selection to *.CH to list header files.

4. Choose Open.

   CA-Visual Objects imports the selected file in its entirety.

For example, if you choose CHGINDEX.PRG from the GSTUTOR subdirectory, the entire sample program will appear in the Source Code Editor window:



```
TEXTBLOCK CHGINDEX.PRG
    Import DATE: 06/25/96 4:14:34 PM
METHOD ByName() CLASS CustOrd
    SELF:Menu:UnCheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NUMBER_ID)
    SELF:Menu:CheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NAME_ID)
    SELF:Server:SetOrder("CUSTNAME")

METHOD ByNum() CLASS CustOrd
    SELF:Menu:UnCheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NAME_ID)
    SELF:Menu:CheckItem(IDM_STANDARDSHELLMENU_VIEW_BY_NUMBER_ID)
    SELF:Server:SetOrder("CUSTNUM")
```

EntLine: 1 Line: 1 Col: 1 Length: 13 C:\CAVO20\SAMPLES\GSTUTOR\CHGINDEX.PRG, Changed

Note: If you are importing the source code into its own newly created module as suggested earlier, the current Source Code Editor window will be empty beforehand. If, however, you are using an existing module, the current Source Code Editor appends the imported text to the existing text.

Also, during the import, CA-Visual Objects analyzes the file, visually separating each individual entity with a marker and color-coding the text based on the current Source Code Editor options.

Once the import is complete, click the Save toolbar button to save the imported code before you begin editing. Once you have completed editing the code, close the Source Code Editor and save any additional changes to the source code entity. When you attempt to save, you will be prompted to resolve any conflicts with existing source code that may arise.

Exchanging Projects

In CA-Visual Objects, you can create and use multiple repositories which are represented by projects. As mentioned earlier in this guide, all of the projects that are available to you are managed through a *project catalog*.

Adding Projects to a Catalog

A project can belong to only one catalog at a time. When you create a new project, it is added automatically to your catalog. You can also add a new project as long as it does not belong to another user's catalog. To do so, use the File menu commands, New Project and Add Project, to create or add projects to the Repository Explorer, respectively.

Removing a Project from a Catalog

If you wish, however, to exchange your work with other members of a development team, you can remove the project from your catalog. Note that this action does not *delete* the project; the Delete from Catalog command, accessible from a local pop-up menu, removes the project from the Repository Explorer, but keeps the directory and contents of the repository intact. This action simply allows another developer to add the existing project to his or her own catalog.

Important! *Projects cannot be moved between different versions of CA-Visual Objects.*

Deleting a Project

Of course, you can delete a project in its entirety, using the Delete command from the same local pop-up menu. This will delete the repository directory and remove the project from the Repository Explorer.

For more detailed information about the New Project, Add Project, Delete from Catalog, and Delete menu commands, see the **Managing Projects** section in "Using the Repository Explorer."

Appendix

A

File Types

The following table contains a list of the various file types you will work with in the IDE and CA-Visual Objects applications:

File Type	Description
.AEF	Exported application
.APP	Internal repository file
.ASP	Active Server Page file
.BMP	Bitmap and ribbon files
.CDX	FoxPro index file
.CH	Xbase include file
.CUR	Cursor file
.DBF	Xbase database file
.DBG	Temporary executable file for debugging
.DBT	Xbase memo file
.DFL	Xbase OLE file
.DLL	Dynamic link library file
.EXE	Executable program
.HTM	HTML source file
.ICO	Icon file
.IND	Internal repository file
.INF	Information file

File Type	Description
.ISC	Installation script file
.MAP	Linker map file
.MDF	Master document file
.MDX	dBASE IV index file
.MEF	Exported module
.NTX	Index file
.OCX	ActiveX control
.PRG	CA-Clipper source code
.REG	Registry import file
.RET	CA-Report Writer reports
.SCC	Project subdirectory for source code control
.TPL	Template file for source generation
.UDC	User-defined command file
.VO	Internal repository file
.VOM	Internal map file

Note: CA-Visual Objects also supports various SQL database file types, whose file extensions depend on the corresponding ODBC driver.

Appendix

B

CA-Visual Objects Registry Entries

This appendix is for reference purposes only, providing an overview of important entries that CA-Visual Objects places in the System Registry. These entries control environmental settings and compiler settings, as well as the directories where important data files are stored.

Note: Any settings not documented here are not relevant for regular operation, and consequently must not be touched! Moreover, changing registry settings directly using the RegEdit program is not recommended, as deleting or modifying registry entries might cause some programs to stop working.

The Multi-Tiered Registry

The registry is organized as a hierarchical tree structure in the following ways:

- On the top level, there are six system-defined root keys. The important key to look at for the active CA-Visual Objects 2.7 settings is HKEY_CURRENT_USER.
- Each key can have any number of subkeys. The CA-Visual Objects 2.7 entries are found under HKEY_CURRENT_USER\Software\ComputerAssociates\ CA-Visual Objects 2.7.

  The entries are grouped into different collections. The actual entries are values of subkeys in HKEY_CURRENT_USER\Software\ComputerAssociates\ CA-Visual Objects 2.7. Generally, a key can have any number of values as well as any number of subkeys.
- Each value consists of a name and an actual data value (e.g., ExecutablePath="C:\CAVO27\BIN").

Note: For more detailed information about the registry in CA-Visual Objects 2.7, see the “Operating Environment” chapter in the *Programmer’s Guide.*

Adam Options Key

Setting	Function
StandardUDC	Shows the name of the default .UDC file which, by default, is STD.UDC.

Compiler Key

Setting	Function	Where Set
AcceptUDE	Determines whether the program accepts undeclared variables for Xbase compatibility.	The Undeclared Variables check box in the Default Compiler Options tab of the System Settings dialog box (File, Setup). The default is 0 (unchecked).
CheckClassPtr	Determines whether the program generates checks to ensure that objects assigned to object variables are of the right class.	The Class Checking check box in the Default Compiler Options tab of the System Settings dialog box . The default is 1 (checked).
CompatibleDiv	Determines whether dividing two integers produces a floating point or an integer result.	The Integer Divisions check box in the Default Compiler Options tab of the System Settings dialog box. The default is 0 (not checked).
EnableProc	Determines whether CA-Visual Objects will support the ProcName() and ProcLine() functions.	The PROCNAME / PROCLINE check box in the Default Compiler Options tab of the System Settings dialog box. The default is 1 (checked).
Generation	Determines whether to optimize for speed or size.	The speed and size radio buttons in the Default Compiler Options tab of the System Settings dialog box. Use 2 for speed and 3 for size in the registry key. The default is 2.
MethodOperator	Determines whether CA-Visual Objects allows common operators such as +, –, and * for method invocation.	The Operator Methods check box in the Default Compiler Options tab of the System Settings dialog box. The default is 0 (unchecked).

Setting	Function	Where Set
OffsetCheck	Determines whether CA-Visual Objects generates runtime checks for numeric overflow conditions.	The Overflow check box in the Default Compiler Options tab of the System Settings dialog box. The default is 1 (checked).
OldAssignAllowed	For Xbase compatibility, determines whether the program allows you to have the equal sign as an assignment operator. When not used as the assignment operator, any use of "=" is considered the equality comparison operator.	The Old Style Assignments check box in the Default Compiler Options tab of the System Settings dialog box. The default is 0 (unchecked).
Optimization Level	Determines the level of optimization (high, medium, low, or none).	The radio buttons in the Default Compiler Options tab of the System Settings dialog box. The default is 0 (none). Other options are 1, 2, and 3 for low, medium, and high, respectively.
RangeCheck	Determines whether CA-Visual Objects generates checks for attempts to access dimensioned array elements outside the current size of the array.	The Range Checking check box in the Default Compiler Options tab of the System Settings dialog box. The default is 0 (unchecked).
Type Inference	Determines whether CA-Visual Objects attempts to infer the data type of undeclared variables by looking at their usage.	The Type Inference check box in the Default Compiler Options tab of the System Settings dialog box. The default is 0 (unchecked).
WarningLevel	Determines the level of warnings the compiler generates. The options are 4, 3, 2, and 1, ranging from every type of warning message to no warning messages at all.	The radio buttons in the Default Compiler Options tab of the System Settings dialog box. The default is 2 (low).

DBServerEditor Key

Setting	Function
ParentClass	Specifies the parent class of the DBServer Editor, which you can modify. The default is "DBSERVER".

Directories Key

Setting	Function	Where Set
AEFPath	Sets the default export and import path for applications only. This entry does not appear in the registry key until set in the IDE.	In the Export or Import dialog box (File, Export or Import when in the Repository Explorer).
ApplicationsPath	Sets the default project directory.	Implicitly changed through switching projects in the Repository Explorer.
ExecutablePath	Sets the default directory where generated .EXE and .DLL files are stored.	In the System Settings dialog box.
MEFPath	Sets the default export and import path for modules only.	In the Export or Import dialog box (File Export or Import from within a module). Note that changing the setting in this dialog box will not update the registry key; however, the registry key will update this dialog box.
PRGPath	Sets the default directory where the Source Code Editor looks for files and the compiler looks for icon (.ICO) files declared using the RESOURCE ICON statement.	In the System Settings dialog box. Note that changing the setting in this dialog box will not update the registry key; however, the registry key will update this dialog box.
PrjCatPath	Sets the directory that holds the project catalog file (PRJCAT.VO). The project catalog contains a list of the currently available projects and their locations.	Only to be set by the installer.
Project	Name of the currently selected project.	Implicitly changed through switching projects in the Repository Explorer. The default is "Default Project".
SysPath	Sets the path to the system repository	Only to be set by the installer.

EnvironmentOpt Key

Setting	Function	Where Set
AskClose	Determines whether CA-Visual Objects will ask you to confirm that you want to quit when you choose File Exit or double-click the System Menu.	The Confirm on Exit check box in the System Settings dialog box (File, Setup). The default is 0 (unchecked).
ColorLEDs	Determines how the program indicates compilation errors and warnings in the Error Browser.	The Color LEDs check box in the System Settings dialog box. The default is non-0 (checked).
CreateDefaultModule	Determines whether the system creates a new empty module every time you create a new application.	The Create Default Module check box in the System Settings dialog box. The default is non-0 (checked).
EditorTabStops	Determines the tab stop setting for the Source Code Editor.	This entry cannot be set from within the IDE. The default tab stop setting is 4, but you can change this setting to range anywhere from 1 to 40.
EditShowBlocks	Determines whether CA-Visual Objects displays entity separators (a red line separating entities) in the Source Code Editor.	The Entity Markers option on the View menu. The default is non-0 (checked).
NewModuleDebug	Determines whether the system automatically turns debugging on for new modules.	The New Module Debug check box in the System Settings dialog box. The default is 0 (unchecked).
Show	Determines what application types (Executable, DLL, or Library) are displayed in the Repository Explorer.	The Options dialog box (View, Options). The default is 1;1;1, meaning all items in this dialog box are checked. You must choose File, Save Desktop to save this entry for all future sessions.
ShowPrototypes	Determines whether the program displays prototypes for entities (like classes, methods, and functions) on the status bar.	The Show Prototypes check box in the System Settings dialog box. The default is non-0 (checked).

Tools Key

Setting	Function
CAVODED.DLL	Name of the DB Server Editor DLL
CAVOFED.DLL	Name of the FieldSpec Editor DLL
CAVOIED.DLL	Name of the Image Editor DLL
CAVOMED.DLL	Name of the Menu Editor DLL
CAVOSED.DLL	Name of the SQL Editor DLL
CAVOWED.DLL	Name of the Window Editor DLL

Window Key

Setting	Function
OLEObjInPlaceEnabled	Determines if the Window Editor will allow in-place editing of OLE objects. The default is 1 (allows in-place editing).

Dynamic Memory

The Dynamic Memory System is not designed to replace database processing via external disk space; however, it should keep the programmer from allocating and freeing memory for local, global, and instance variables. The default maximum size available for dynamic memory is 16MB. This can be verified by checking the return value of the DynInfoFree() function.

If some applications need more dynamic memory, this size can be changed via the System Registry:

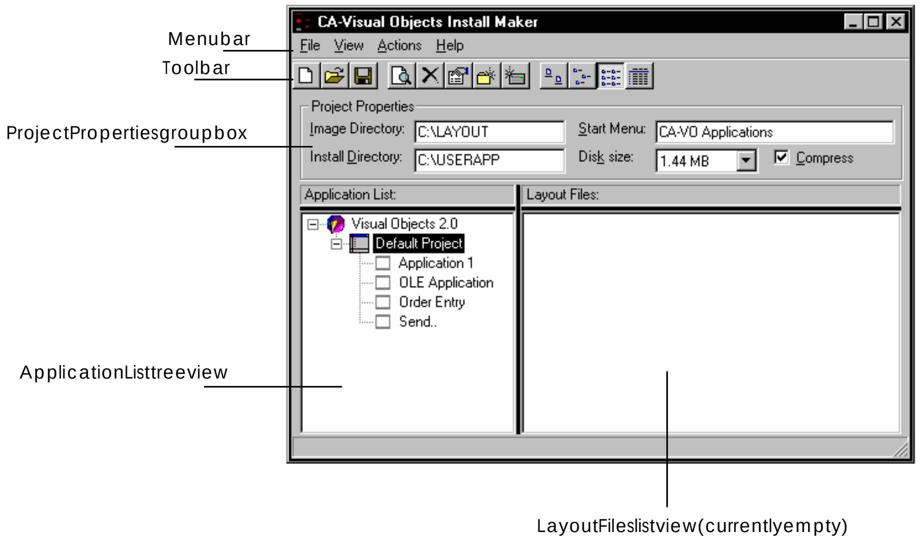
Key	HKEY_CURRENT_USER\Software\ComputerAssociates\CA-Visual Objects Applications\Runtime
Value	MaxDynSpace (DWORD), where MaxDynSpace represents the new value in bytes either as a decimal (16777216 for 16MB) or hex (1000000 for 16MB). The default setting for MaxDynSpace is 16777216. To increase the dynamic memory to 32MB, you would set MaxDynSpace to 33554432.

Appendix

C

Using the Install Maker

The Install Maker, shown below, allows you to make an *install set* for a selected application in your CA-Visual Objects repository by specifying what files (DLLs, RDDs, and so on) in the Layout Files list view are to be included on the disk(s) or CD-ROM. It also allows you to specify several options, such as floppy disk size, directory for disk images, and Start menu name.


Menubar
Toolbar
ProjectPropertiesgroupbox
ApplicationListtreeview
LayoutFileslistview(currentlyempty)
CA-Visual Objects Install Maker
File View Actions Help
Application List:
Layout Files:

Project Properties
Image Directory: C:\LAYOUT
Start Menu: CA-VO Applications
Install Directory: C:\USERAPP
Disk size: 1.44 MB
☑ Compress

Visual Objects 2.0
- Default Project
  - ☐ Application 1
  - ☐ OLE Application
  - ☐ Order Entry
  - ☐ Send..

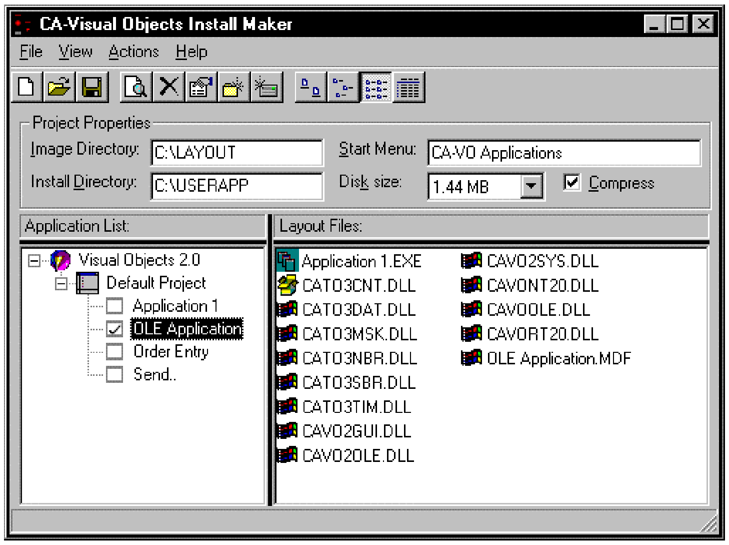
Important! *Since both need exclusive access to the repository, you cannot run the Install Maker and CA-Visual Objects at the same time.*

Program Components

Image Directory	When you create installation disks or CD-ROMs for your application, the Install Maker creates images of these disks. The disk images are stored in subdirectories under the directory name specified in the Image Directory edit control. The disk image subdirectories are named according to disk number (e.g., DISK1, DISK2). Once they are created, you must copy each disk image to a separate floppy disk. The install disks should then be tested before being distributed.
Install Directory	This edit control determines where the files will be installed.
Start Menu	When you run the install disks created by the Install Maker, a folder will be created and added to the Windows Start menu. The name of this folder is specified in the Start Menu edit control. This name will also appear at the top left-hand corner of the screen when the install program is run.
Disk Size	The Install Maker program must know the size of the installation disks. Use the Disk Size combo box to specify the disk size. Valid choices are: No Limit, 2.88 MB, 1.44 MB, 1.22 MB, 720 KB, and 360 KB. The default is 1.44 MB. See Floppy Disk Utilization later in this appendix for more detailed information about selecting floppy disk size for your install set.
Compress Option	By default, the project files are compressed by CA-Visual Objects file compression facility. **Note:** If you deselect this option, you will receive a warning message if a file is too large for the specified disk size.

Application List

The Application List tree view displays all the applications in the current CA-Visual Objects repository. When you click on an application, its corresponding files display in the Layout Files list view. For example:



Project Properties
Image Directory: C:\LAYOUT
Start Menu: CA-VO Applications
Install Directory: C:\USERAPP
Disk size: 1.44 MB
[✓] Compress

Application List:
Visual Objects 2.0
Default Project
Application 1
[✓] OLE Application
Order Entry
Send..

Layout Files:
Application 1.EXE
CATO3CNT.DLL
CATO3DAT.DLL
CATO3MSK.DLL
CATO3NBR.DLL
CATO3SBR.DLL
CATO3TIM.DLL
CAVO2GUI.DLL
CAVO2OLE.DLL
CAVO2SYS.DLL
CAVONT20.DLL
CAVOOLE.DLL
CAVORT20.DLL
OLE Application.MDF

Note that you can select any number of applications for the specified installation disk(s).

Tip: Just as you can customize the Repository Explorer's list view pane, you can do the same for the Install Maker's Layout Files list view by selecting the Large Icons, Small Icons, List or Details command from the View menu. You can also arrange the icons by name, type, size, or date using the corresponding Arrange Icons menu commands.

Layout Files

The Layout Files list view displays the files for the selected application that will be placed on the install disks created by the Install Maker. Files in this list can be added or deleted—simply click the Add File or Delete File button, respectively. (You can also choose the Delete command from a local pop-up menu when you click on a file.)

Important! *The Install Maker tries to find the best match for files associated with the selected application. It is up to you to test the install disks and make sure that all files are included. You have the option of deleting or adding files to the Layout Files list view. Refer to the "Operating Environment" chapter in the Programmer's Guide for more information about preparing your application for delivery.*

Add File Button

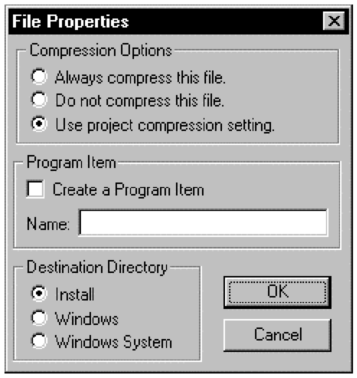
This toolbar button opens a standard Open dialog box from which you can add files to the Layout Files list view. This is useful if the Install Maker did not add all of the files needed for the install. It is also necessary if you have ancillary files (such as database, index, report, help, and read me files) that you need to add to the install.

Delete File Button

This toolbar button will delete the selected file from the Layout Files list view. This is useful if the Install Maker has added unnecessary files to the Layout Files list view.

Properties Button

The Properties toolbar button will open the File Properties dialog box, shown below, for the selected file allowing you to set the properties for each file in the Layout Files list view (such as whether the file should be installed in the default Install Directory or your Windows directory). You can also decide if a file will be a program item and what name that program item will have. By default all .EXE files are set as program items, but you may also want to include .HLP files, for example.


File Properties

Compression Options
- Always compress this file.
- Do not compress this file.
- [✓ Use project compression setting.]

Program Item
- Create a Program Item

Name:

Destination Directory
- [✓ Install]
- Windows
- Windows System

Tip: You can also choose the Properties command from a local pop-up menu when you right-click on a file.

Compression Options

The Compression option on the main Install Maker window, if checked, provides for global compression of *all* files listed in the Layout Files list view. (Checked is the default setting.)

The File Properties dialog box, however, allows you to override the default setting and to select compression options for individual files. Valid options are:

- Always Compress the File

  Select this option if there is a particular file that you *always* want to compress, like a README.WRI file.

- Do Not Compress This File

  Select this option if there is a particular file that you *never* want to compress, like a large .DBF file.

- Use Project Compression Setting

  This option—the default setting—confirms the project compression setting as specified in the main Install Maker window.

Program Item

To specify that a file be installed as a program item in the folder, select the Create a Program Item option. Then enter a description in the Name edit control that will appear as the item in the folder.

Note: All executable (.EXE) files are, by default, set to be program items in a folder with the file name of each .EXE used as the description. If you want to override the file name with more descriptive text, you can do so. Also, if you have created a multi-application installation set where there is more than one .EXE file and you do not want certain .EXE files to be installed as items, you can override the default setting.

Destination Directory

Select the destination directory on the target machine for the specified file. Valid choices are:

- Install

  If selected, the destination directory is the Install directory that is specified in the main Install Maker window.

  Note: Most of the files in the Layout Files list view will be installed in the Install directory, as these will be CA-Visual Objects-specific DLL and RDD type files.

- Windows

  If selected, the destination directory is the Windows directory.

- Windows System

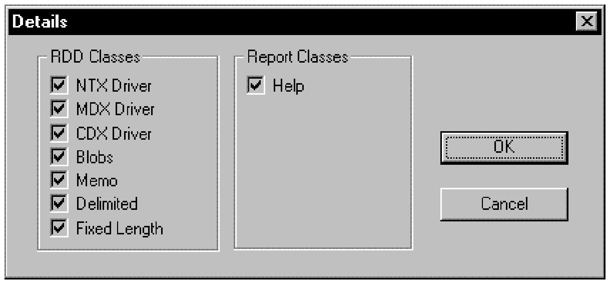
  If selected, the destination directory is the Windows System directory.

CA-Visual Objects automatically sets this property for you for the files that CA-Visual Objects recognizes. If you have added any files to the Layout Files list view that CA-Visual Objects does not recognize, they will default to the Install directory. If these files need to be installed in either Windows or Windows\System, you will need to change the property manually

Note: You should save the current Install Maker session to a project file, because after testing you may find you want to modify certain file properties. If this is the case, you can restart the Install Maker and open up your old project file and modify it. See Project Files later in this appendix for more detailed information.

Details Button

The Details toolbar button will open the Details dialog box, shown below, which displays RDD Classes and Report Class information for the specified file.



This dialog box allows you to fine-tune the install set for your application, excluding files or subsystems that you have determined to be unnecessary for the proper operation of your application.

RDD Classes/Report Classes

Each check box in the Details dialog box corresponds to a file or subsystem in CA-Visual Objects. These components may or may not be present in your install set depending on what libraries you selected when you created your application.

Tip: To see exactly which files correspond to each check box in this dialog box, look at INSTAPP.INF in your CAVO27\BIN directory.

Note that all boxes are initially checked by default, and that all files found in the Files Layout list view will be included in the install set.

To fine-tune your install set:

1. Deselect any files or subsystems that you want to exclude from the Files Layout list view.
2. Click OK.

   If the files or subsystems were found in your original Files Layout listing, those files will now be excluded when you return to the main Install Maker window.
3. Save this Install Maker session to a project file.

   After testing you may find you need certain components restored. If this is the case, you can restart the Install Maker and open up your old project file and modify it. See Project Files later in this appendix for more detailed information.

Important! *It is imperative that you install and test your application on a machine that does not have CA-Visual Objects installed (or any remnants of a prior CA-Visual Objects installation). You must ensure that all files necessary for your application are included on the installation disks. Refer to the INSTAPP.INF file in the CAVO27\BIN directory for a list of CA-Visual Objects files that are required for each library in your application.*

Make Disks Button

This toolbar button creates the disk images from the files in the Layout Files list view. It will also check if there are disk images in the current Image Directory. If there are already images, it will give you the option of deleting these directories or canceling the make disks process.

Floppy Disk Utilization

Toiward the end of INSTAPP.INF there is a new group of settings, DISK SIZE, that will determine the maximum utilization of your floppies at the end of the make disks process. The value assigned to each floppy density is approximately ninety percent (90%) of the size allowed for an empty, non-bootable floppy disk of the selected density. This number can be modified up or down. Once changed, it is effective for all Install Maker sessions.

The default disk size settings are shown below:

NoLimit = 2147483647

2.88MB = 2785126

1.44MB = 1384780

1.22MB = 1153254

720KB = 692390

360KB = 344371

You should never choose the maximum allowed size, as this will not allow for space for the FAT table entries, especially when there are many small files. There will probably also be a discrepancy between the sector size of your hard drive and the sector size of the floppy. Both of these factors may result in overflowing of the floppy when you actually try to copy the files to the floppy disk.

If you want to allow for expansion or inclusion of more files on the floppy outside the control of the Install Maker, you can make the size smaller, but this will affect all of the floppies. Each disk of your floppy install set will be limited to the size selected.

If you choose the No Limit option, the Install Maker will copy all files to a DISK1 subdirectory. This is useful for creating a CD-ROM installation disk.

Important! *You should always check that there is enough disk space on your destination hard drive before you complete the make disks process. The Install Maker will copy each file in the install set for a new IMAGE subdirectory that you create with the Actions Make Disks menu command or the Make Disks button.*

Project Files

Under the File menu, there are the New, Open, Save, and Save As options. Using these commands, or their corresponding toolbar buttons, you can save all the information for the current install set you defined. These options are useful in case you need to make changes to the install set, such as adding a new file.

Producing Install Disks

In summary, the basic steps required to produce install disks for your application are as follows:

1. After complete development of your application in the IDE, check the application's properties to make sure that the .EXE file name and the folder name are as you want them to be.
2. Generate an executable file using the Make EXE toolbar button.
3. Shut down CA-Visual Objects, then start the Install Maker from the Windows Start menu.
4. In the Image Directory edit control, enter the directory where you want to save the disk images.
5. In the Install Directory edit control, enter the name of the directory in which the application will be installed on the end user's computer.
6. From the Disk Size combo box, select the floppy disk size that your application will be distributed on.
7. Enter the Windows Start menu name for your application in the Start Menu edit control.
8. Optionally, deselect the Compress option.

   Note: If you do so, you should select one of the compression options available at the file level via the Properties dialog box. Otherwise, you may receive a warning message.
9. Click on an application in the Application List tree view.

   The Layout Files list view displays the default files needed by the application.
10. Using the Add File button, add any additional files needed by your application, such as .DBF and .HLP files.
11. Optionally remove any file from the Layout Files list view that you know is not needed by your application using the Delete File button.
12. Highlight each file in the Layout Files list view and click on the Properties button to make sure the file will be installed in the correct subdirectory and to optionally designate the file as a separate program group item.
13. Make sure you have enough disk space on the drive indicated in the Image Directory to path, and then press the Make Disks button to create the disk images.

14. Choose Exit to close the Install Maker.
15. After the disk images are created, copy each disk image to a separate, newly formatted floppy disk of the size selected.
16. Install your application on another computer that does not have CA-Visual Objects installed, and retest it as thoroughly as you have already done during dynamic execution from within the IDE.

Appendix

D

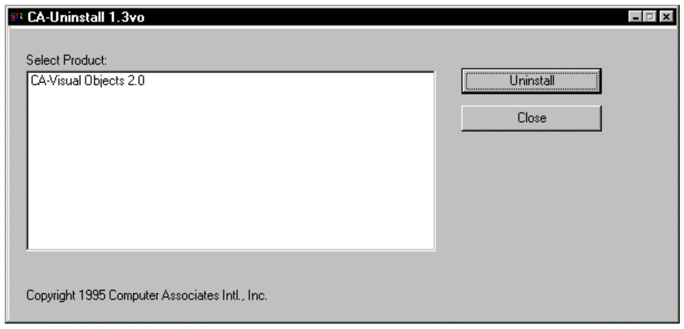
Using the CA-Uninstall Utility

The CA-Uninstall utility will remove CA-Visual Objects program files, registry entries, Start menu entries, and all .EXE files in the BIN directory.

Starting CA-Uninstall

To start the CA-Uninstall utility, select the Uninstall program item from the CA-Visual Objects 2.7 Start menu.

The CA-Uninstall dialog box appears:



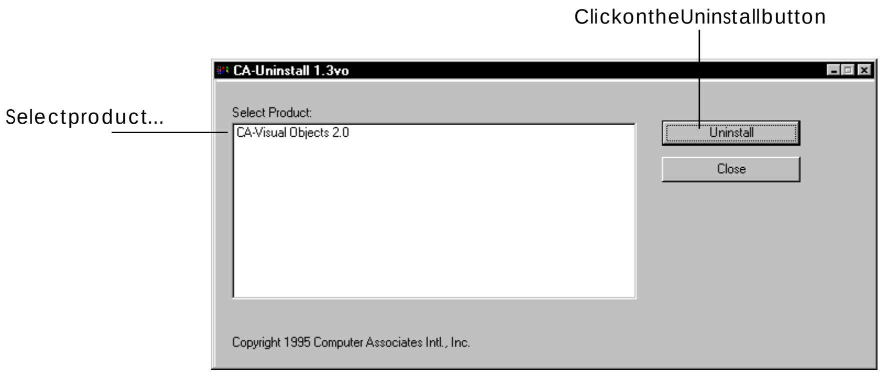
CA-Visual Objects 2.0

Copyright 1995 Computer Associates Intl., Inc.

Uninstalling CA-Visual Objects

To uninstall CA-Visual Objects:

1. Select CA-Visual Objects 2.7 in the Select Product list box:



2. Click on the Uninstall button.

   Once the uninstall is complete, the CA-Uninstall dialog box closes.

To verify that the registry entries have been removed, run the RegEdit program. Under the Software branch of the HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE key entries, the Computer Associates entry will be removed.

Note: If you have other Computer Associates software installed, the Computer Associates entries will remain but the CA-Visual Objects 2.7 entry will be removed.

Index