Chapter 4
An Overview of the IDE

In This Chapter

This chapter presents an overview of some of the features of CA-Visual Objects. Its purpose is to help you gain both an understanding of what features are available to you, as well as a familiarity with the basics of working in CA-Visual Objects, so that you can go on to complete the sample application introduced in the next chapter.

Note: This chapter only touches upon some of the tools provided by CA-Visual Objects. For complete details, please refer to the *IDE User Guide*.

Repository-Based Development

Before you start to use CA-Visual Objects, you need to understand the implications of moving from *file-based* development systems to a *repository-based* development system.

No Need to Work with Files

First of all, you do not have to deal with files when working with CA-Visual Objects. Instead of an application that is comprised of one or more files (.PRG, .CH, etc.), an application now consists of one or more *modules*. In addition, all the items that were in your files—such as functions and procedures—are now referred to as *entities*.

In CA-Visual Objects, all of these things—applications, modules, and entities—are stored in a *repository*. While they are all still manageable, editable pieces of the application, they are no longer file-based—the repository holds them all. (For example, if you import source code from another application into the repository, there is seldom a need to work with external files of any kind once the files are imported.)

Note: Modules in the repository can be linked to external files if you prefer to maintain a file-based application. Also, CA-Visual Objects provides File Import and Export commands that you can use for maintaining backup files.

An Internal, Automated MAKE Facility

Secondly, the repository manages all of the pieces of an application for you. It automatically maintains the relationships between the various entities of an application. Each time you build an application, the repository “knows” what to compile based on changes that you have made, and builds the application in the most efficient way. Such automation eliminates the need for make files and compiler and linker script files.

$$\begin{tikzcd}\text{Applications} \arrow[d] \\ \text{Modules} \arrow[d] \\ \text{Entities}\end{tikzcd}$$

The repository is based on a hierarchical, object-oriented view of an application. Applications (like “Order Entry”) and libraries (like the GUI Classes library) consist of modules (such as “Customer Review”) which in turn consist of entities (such as “CLASS Customer” and “METHOD Customer:PlaceOrder”).

The highest level in the hierarchy is the *application.* In CA-Visual Objects, an “application” can be defined as one of three types: application, DLL, or library. Specifically, an *application* is exportable as an executable file and a *DLL* as a dynamic shareable, whereas a *library* is used only at compile time and is included in an .EXE or .DLL file.

Modules, which form the second level in the hierarchy, are in many ways comparable to traditional source files (for example, .PRG files). They contain a group of logically related parts of the application, and may be used to limit the visibility of variables, functions, classes, etc. defined in the module.

Similarly, just as a typical .PRG file contains function and procedure declarations, modules in CA-Visual Objects contain *entities*. Entities form the third level in the hierarchy. An entity is any part of your application that has a name and can be edited. Some of the available entity types are:

- windows
- menus
- reports
- procedures
- functions
- resources
- classes
- methods
- structures
- globals
- constants

The IDE Tools

CA-Visual Objects features an integrated development environment (IDE) that provides you with a flexible, intuitive, and powerful environment for creating applications.

Within this single desktop, you can access almost any of the IDE features from any window at any time. You can also open and simultaneously work with multiple windows and editors.

The IDE provides a rich set of *tools* that can be used to create sophisticated, GUI applications. Like a hammer or ruler, a tool allows you to create things. For example, there are *browsers*, which let you organize and view your data, and *editors*, which allow you to create windows, menus, source code, databases, reports, and icons.

Visual Editors

Many of the editors in CA-Visual Objects are *visual* and, in almost all cases, the flexibility and ease-of-use provided by the visual editors can help you work more efficiently. Their point-and-click, drag-and-drop design approach and WYSIWYG environment allow you to develop an application *visually*, thereby improving the quality of the application and reducing the total development time.

Instead of working directly in programs with the CA-Visual Objects language, you can lay out the visual aspects of the application and much of its functionality, providing ongoing evaluation of the application as it is created, as well as meaningful feedback about the design.

For example, to add controls (like check boxes or list boxes) to a window using the Window Editor, you simply click an icon in a tool palette and click in the window to place it. You can then manipulate and define the control as desired (for example, resize, change colors and fonts, or add code to handle events).

Likewise, when designing a menu in the Menu Editor, it is displayed in a partially operational “preview” menu bar, so you can view what your menus look like as you create them. This preview area is continually updated as you work, providing immediate visual feedback.

Creating an application in a visual fashion improves the quality of the application and reduces the total development time, as it leads to a better definition of what is needed and thereby provides for an application that best meets the user’s needs.

Generating Code

When you are finished designing in any of these editors and have saved your work, CA-Visual Objects generates powerful and straightforward object-oriented code based on the underlying class libraries.

For example, creating a window in the Window Editor will generate a subclass of the Window class. The generated code is not only efficient and powerful, it is clean and maintainable and forms a solid foundation for the future evolution of the application.

A Complete Development Environment

Of course, CA-Visual Objects provides a host of other complementary tools to complete the development environment, including a source code editor, a compiler, and a debugger.

The CA-Visual Objects IDE is designed to provide a productive framework for developing all kinds of applications—including mission-critical business systems—and is specifically designed to support the iterative development paradigm.

All development tools provided in CA-Visual Objects are closely integrated with the repository. In fact, all aspects of working with application components—looking at them, analyzing their relationships, and editing them—is done from the repository. This ensures efficient development and protects the integrity of your applications.

The Browsers

Browsers provide a convenient and organized way to view the data that is currently stored in your repository. In CA-Visual Objects, you can browse:

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

Most browsers in CA-Visual Objects can be customized to display a particular subset of data. For example, the Class Browser displays classes in a collapsible/expandable tree structure that lets you determine what information to display.

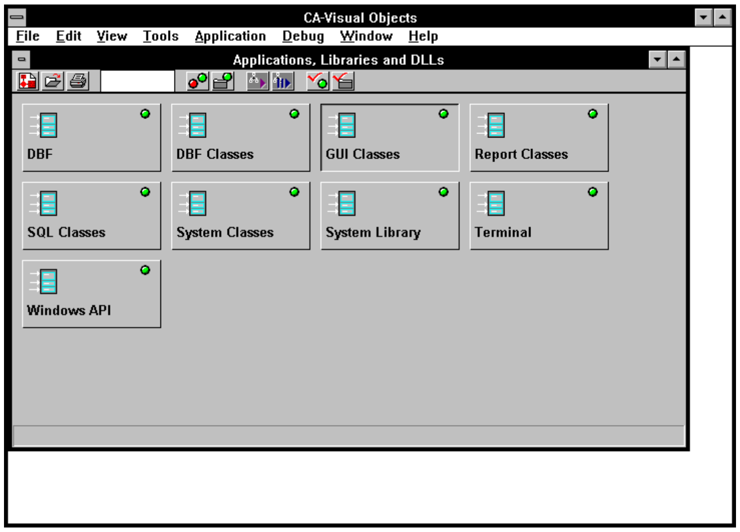
In addition, the close integration of the browsers with the repository provides easy access to the various editors. For example, double-clicking on a source code entity (such as a GLOBAL or a METHOD) in an Entity Browser loads the code for that entity in the Source Code Editor, while double-clicking on a window entity loads the window definition in the Window Editor.

Browsers, therefore, serve a variety of purposes. The views they provide give you an overall picture of the data that is stored in your repository. Browsers also allow you to manipulate that data—for example, you can rename an application or move a module to another application. Finally, they provide access to the various CA-Visual Objects editors.

Application Browser

As you have already learned, in the CA-Visual Objects hierarchy, applications are comprised of modules which contain entities. The primary browsers, therefore, follow this top-down hierarchy: Application Browser, Module Browser, and Entity Browser.

The *Application Browser* appears when you first start CA-Visual Objects:



This window allows you to view what is currently stored and maintained in the repository. Each button in the Application Browser represents an application, library, or DLL.

Initially, the Application Browser displays all of the libraries supplied by CA-Visual Objects. As you start to add to the repository, creating your own applications, libraries, and DLLs, the Application Browser will display them all.

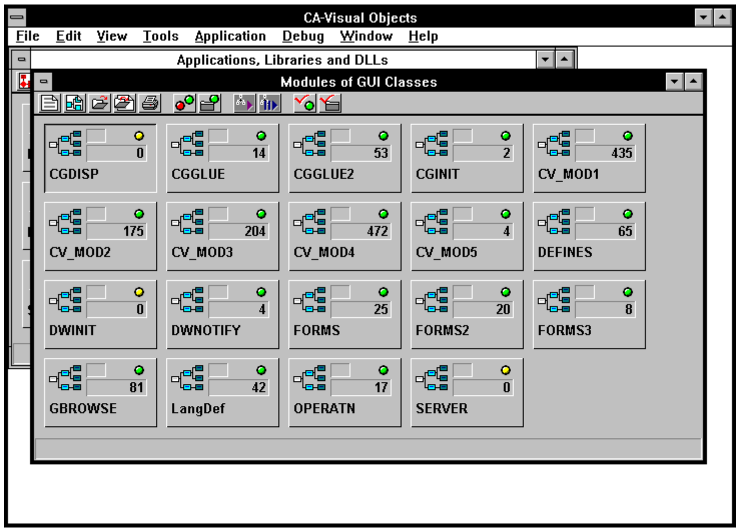
If desired, you can customize this browser as you work. For example, using the View Show menu command, you can choose to suppress the display of applications, libraries, and/or DLLs. Likewise, you can use the View Sort menu command to sort the buttons in the Application Browser by name or by type and use the Name Filter area on the toolbar to limit which names are displayed.

Double-clicking on one of these buttons "opens" it, displaying the modules associated with that application, library, or DLL in a Module Browser.

Note: You will soon see that you can have many different Module, Entity, Class, and Error Browsers open at the same time. However, there is only one Application Browser.

Module Browser

After double-clicking on a button in the Application Browser, CA-Visual Objects displays a *Module Browser*. For example, double-clicking on the GUI Classes button displays the following:



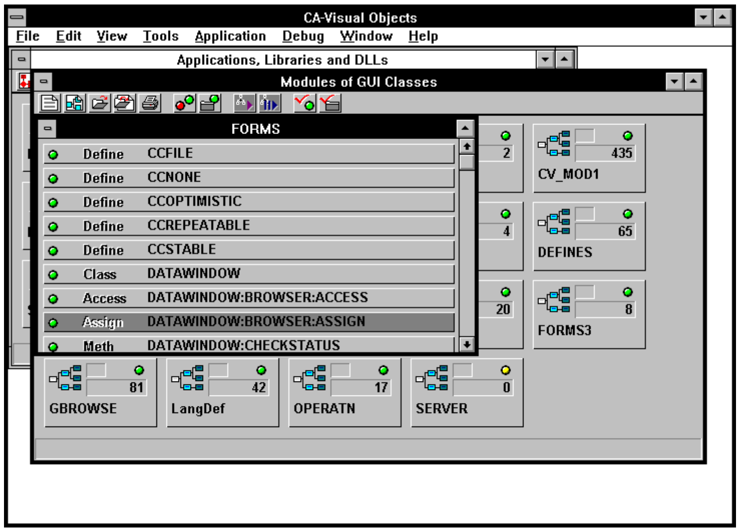
Like the Application Browser, a Module Browser displays each module as a button. You can display a Module Browser for every application, library, and DLL in your repository.

Buttons in Module Browsers are arranged in alphabetical order. Double-clicking on one of these buttons opens it, displaying its entities in an Entity Browser.

Entity Browser

Double-clicking on a module brings up a new window that displays all the entities defined in that module. This window is called an *Entity Browser*.

For example, double-clicking on the Forms module in the GUI Classes library displays the following:



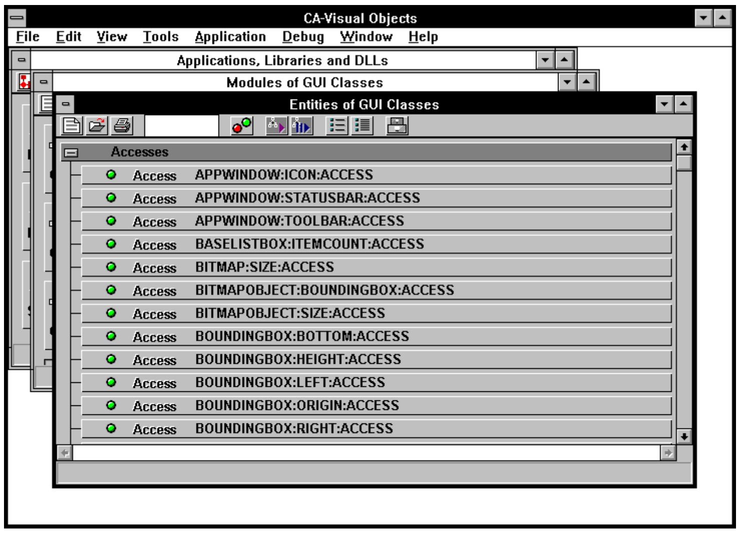
Define	CCFILE
Define	CCNONE
Define	CCOPTIMISTIC
Define	CCREPEATABLE
Define	CCSTABLE
Class	DATAWINDOW
Access	DATAWINDOW:BROWSER:ACCESS
Assign	DATAWINDOW:BROWSER:ASSIGN
Meth	DATAWINDOW:CHECKSTATUS

Entity Browsers display each entity as a sculpted 3-D bar, grouped by entity type in alphabetical order.

Like its predecessors, double-clicking on an entity causes an action. However, rather than loading another browser, double-clicking on something at the entity level starts an *editor*. For example, double-clicking on a menu entity invokes the Menu Editor, while double-clicking on a function entity activates the Source Code Editor.

Note that CA-Visual Objects allows you to browse entities on two levels: you can either view all the entities within a *module* (as described above) or all of the entities within an *application*.

To view all entities within an application, choose the Tools Entity Browser command. For example, choosing this menu command for the GUI Classes library would display the following:

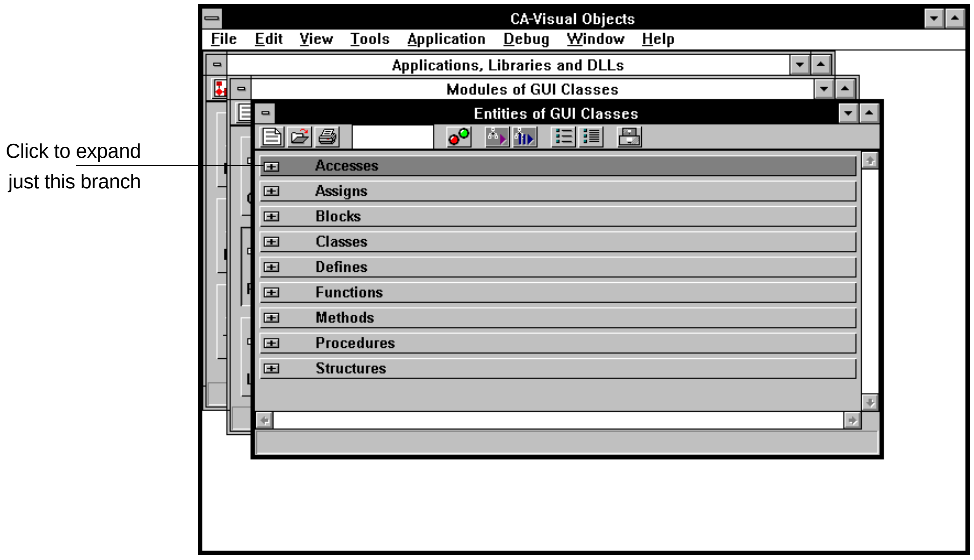

CA-Visual Objects
File Edit View Tools Application Debug Window Help
Applications, Libraries and DLLs
Modules of GUI Classes
Entities of GUI Classes

Accesses

Access	APPWINDOW:ICON:ACCESS
Access	APPWINDOW:STATUSBAR:ACCESS
Access	APPWINDOW:TOOLBAR:ACCESS
Access	BASELISTBOX:ITEMCOUNT:ACCESS
Access	BITMAP:SIZE:ACCESS
Access	BITMAPOBJECT:BOUNDINGBOX:ACCESS
Access	BITMAPOBJECT:SIZE:ACCESS
Access	BOUNDINGBOX:BOTTOM:ACCESS
Access	BOUNDINGBOX:HEIGHT:ACCESS
Access	BOUNDINGBOX:LEFT:ACCESS
Access	BOUNDINGBOX:ORIGIN:ACCESS
Access	BOUNDINGBOX:RIGHT:ACCESS

Note that the items in this Entity Browser are displayed in a collapsible/expandable tree structure (grouped by entity type) that allows selective viewing of entities.

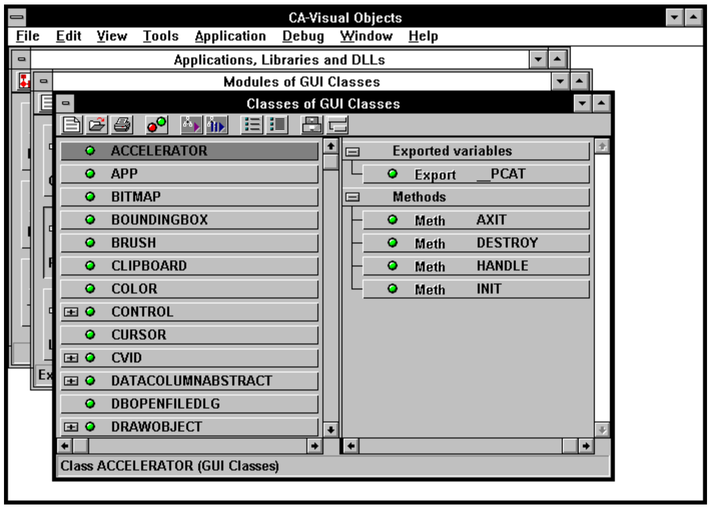
For example, collapse all branches in the entire tree by choosing the View Collapse All menu command; you could then expand only one branch to view its entities in more detail by clicking on the + icon to the left of the branch:



Class Browser

The *Class Browser* allows you to view all classes associated with an application. Like the application-wide Entity Browser, it is accessed via the Tools menu.

For example, choosing the Tools Class Browser command when the GUI Classes library is the current selection displays the following:


CA-Visual Objects
File Edit View Tools Application Debug Window Help
Applications, Libraries and DLLs
Modules of GUI Classes
Classes of GUI Classes

```
ACCELERATOR              Exported variables
APP                        Export   __PCAT
BITMAP                   Methods
BOUNDINGBOX                Meth     AXIT
BRUSH                      Meth     DESTROY
CLIPBOARD                  Meth     HANDLE
COLOR                      Meth     INIT
CONTROL
CURSOR
CVID
DATACOLUMNABSTRACT
DBOPENFILEDLG
DRAWOBJECT
Class ACCELERATOR (GUI Classes)
```

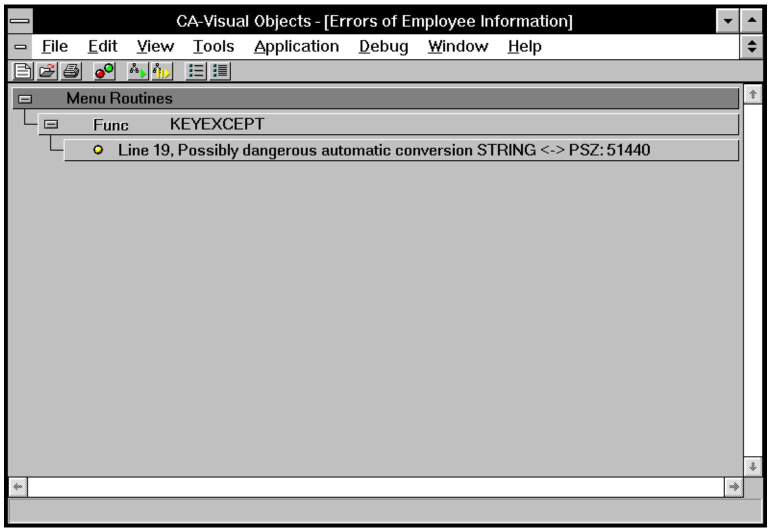
The Class Browser is also similar to the application-wide Entity Browser in that it displays the available classes in a collapsible/expandable tree structure and can be used to access the Source Code Editor by double-clicking on an item. The data displayed in the Class Browser can also be further customized (for example, by choosing the View Include Inherited menu command, you can view all of the instance variables and methods inherited by a class, not just those it owns).

Error Browser

During the development cycle, compiling (or *building*) an application often results in errors. To help you locate and correct errors and warnings quickly and efficiently, CA-Visual Objects provides an *Error Browser*.

As you may have noticed by now, applications, modules, and entities displayed in the various browsers you have seen so far include a small, LED-style icon. These icons indicate compilation status. For example, green means compiled successfully, while red denotes that the item needs to be compiled.

Therefore, when you build an application and a module remains red, the icon lets you know that the module contains one or more entities that have errors. To quickly view and go to these errors, choose the Tools Error Browser command. Choosing this command lists all the entities in the application that have errors or warnings:



Menu Routines
Func KEYEXCEPT
Line 19, Possibly dangerous automatic conversion STRING <-> PSZ: 51440

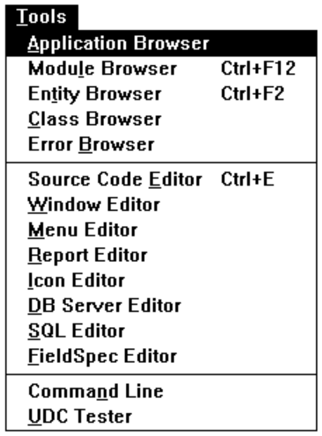
Similar to the Class Browser, the Error Browser displays the entities in a collapsible/expandable tree structure. If you then double-click on an error, you are brought directly to the line in the source code that contains the error.

The Editors

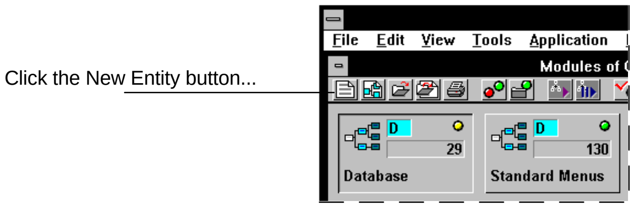
CA-Visual Objects provides the following types of editors:

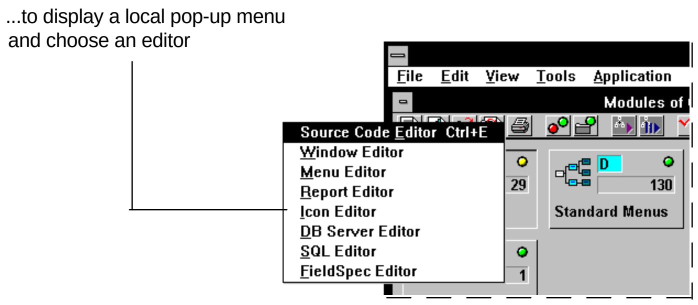
Editor	Creates
Source Code Editor	Source code entities, like functions, procedures, globals, etc.
Data Server Editors	Data server entities, as subclasses of the DBServer, SQLSelect, and FieldSpec classes.
Window Editor	Window entities (like data windows and dialog boxes), as subclasses of the various Window classes.
Menu Editor	Menu entities (like menus and accelerators), as subclasses of the Menu and Accelerator classes.
Report Editor (CA-RET)	Report entities, as subclasses of the ReportQueue class.
Icon Editor	Icon and cursor entities, in the form of .ICO and .CUR files.

These editors can be used to create all the components of a sophisticated GUI application. All editors can be started by choosing a command from the Tools menu:



Alternatively, you can also start an editor by clicking the New Entity toolbar button in any Module Browser and choosing an editor from the displayed list:


Click the New Entity button...
File Edit View Tools Application
Modules of
D
29
Database
D
130
Standard Menus

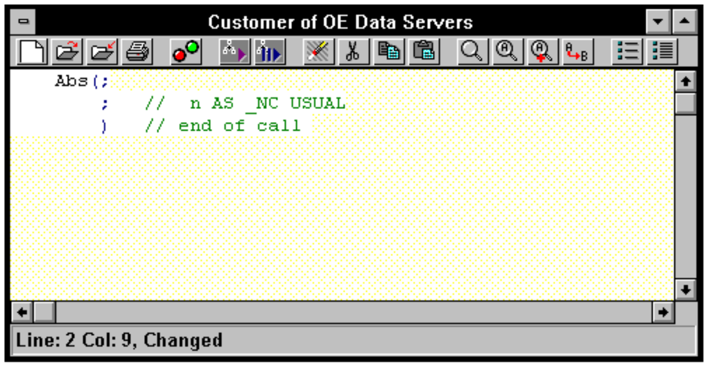


And of course, as described earlier in this chapter, you can access an editor from any Entity Browser, Class Browser, or Error Browser.

Source Code Editor

The Source Code Editor provides a powerful environment for writing and editing code. For example, you can cut, copy, paste, delete, search for, and replace text, as well as undo and redo editing actions, using standard Windows techniques.

You can also fill in an incomplete function or method call with its prototype using the Edit Insert Prototype command—a useful feature if you have simply forgotten the correct syntax. For example, here's the CA-Visual Objects-inserted prototype for the Abs() function:



```
Abs(;
      ;   //  n AS _NC USUAL
      )   // end of call
```

Line: 2 Col: 9, Changed

The Source Code Editor also provides visual feedback by continually parsing each keystroke as you enter source code (or import or paste text) to color-code text based on its structure. Keywords, literals, and comments, for example, are all displayed in different colors, while each entity is separated from the next by a horizontal marker.

The collapse/expand icons, available for every entity loaded, allow you to collapse entities that you are not currently editing to provide a cleaner view of the source code and to expand them again when you need to work with them.

Data Server Editors

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 index files in the Xbase model and WHERE and ORDER BY clauses in the SQL model.

The DB Server and SQL Editors

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.

With both the DB Server and SQL Editors, you can import an existing database structure and generate a default set of field specifications (explained below in The FieldSpec Editor section) that you can optionally modify. The DB Server Editor also lets you generate a database file (and index files) from the data server definition using the File Export command.

Note: No capability for creating SQL tables is provided in CA-Visual Objects.

Using data servers offers you some significant benefits. For example, 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 create using the Window Editor. Thus, you need only define the attributes for a data server once, and they will be automatically inherited and used by any data window that is linked to that data server.

Similarly, changes to a data server (such as the validation rules or picture formats for one or more fields) need only be made in one place, the data server itself. Resources that use the data server will automatically inherit those changes.

Using a data server also provides an integrated view of all the pieces of information related to it. Without this comprehensive entity, you would have to create and maintain the various pieces (tables, index files, relations, and field specifications) independently. Additionally, creating data servers for your database tables allows them to be easily viewed and manipulated within the IDE (for example, using the Class and Entity Browsers).

The FieldSpec Editor

In many cases, the different data servers your application uses contain similar, if not identical, fields (for example, all zip code fields are typically the same, regardless of where they are used). You can either define the properties of these common fields (such as validation and formatting rules) each time you create a new data server, or you can create a single field specification and reuse it in each data server that needs it.

A *field specification* created in the FieldSpec Editor is essentially a set of properties that are related to a field but are *independent* of any particular data server. Thus, multiple data servers can access the same property values for common fields (for example, if you create a Salary field specification, you can simply reuse its properties when creating an EmpSalary field in a data server for an Employee database). Additionally, if you change a field specification, the change will automatically propagate to all appropriate places.

Window Editor

The Window Editor is used for the interactive design of the various windows of your application.

Window Types

You can create several types of windows in the Window Editor, based on subclasses of the standard GUI Classes Window class. For example, you can create MDI shell windows, data entry screens (called *data windows*), and dialog boxes.

Tool Palette

To design these windows, the Window Editor features a floating tool palette. To place a control on a window (such as a push button, list box, or scroll bar), just click a button in the tool palette and click in the window.

You can then go on to define *properties* for your windows and the various controls you place on them (for example, you may want to specify the text that should appear in the status bar when a window or control is selected, or an ID for use in a context-sensitive help system).

Controls and Actions

One important property of certain controls is an event name. This is because in Windows applications, certain types of controls initiate actions, or *events* (for example, when the user clicks the OK button in a dialog box, the program processes the information entered in the dialog box and closes it).

The Window Editor makes it easy for you to associate actions with these types of controls by allowing you to specify an *event name* as a property. You have the option of using any method, window, or report that is visible to your application as an event name and can even specify source code for a customized event name method from within the Window Editor.

There are many different types of controls that you can define in the Window Editor, but before going on to describe them, a few words about data windows are in order.

Data-Aware Windows

The integration of the various tools in the CA-Visual Objects IDE provides some powerful benefits, one of which is the ability to create data windows. Data windows are *data-aware* because they “know” about the data server(s) upon which they are intended to operate.

A data window knows about a data server by a link that you establish between it and one or more data servers. Once a data window and a data server are linked, you can actually link individual controls in the window (such as edit controls and check boxes) with fields in the data server.

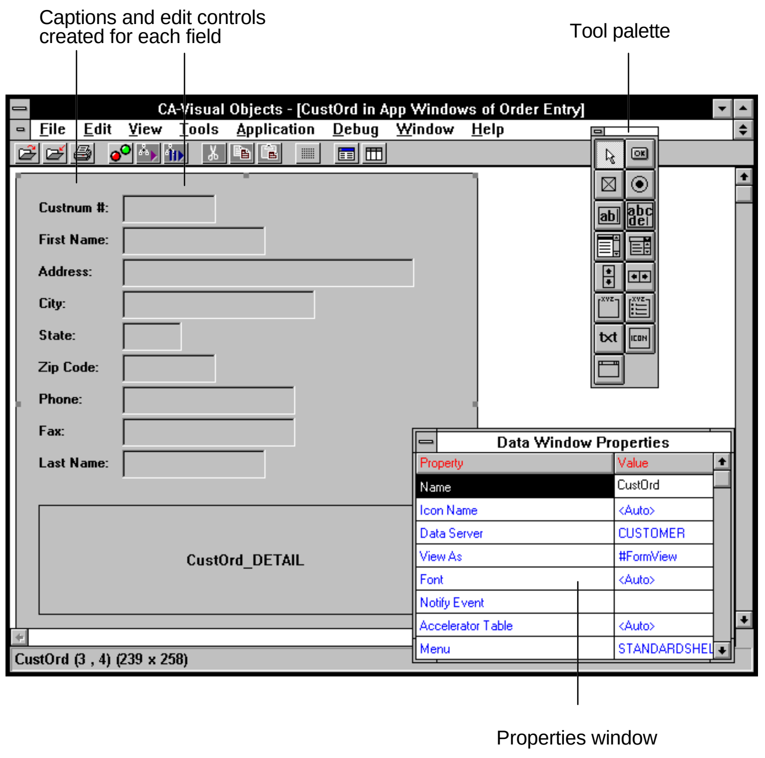
When you link a window control to a field, you are actually linking it with the field specification associated with that field—the control, therefore, automatically inherits and uses all of the field specification’s properties (for example, its validation and formatting rules).

By their very nature, data windows are capable of interacting intelligently with data servers. For example, data windows can easily display the contents of a data server and have preprogrammed methods for moving among the records and manipulating the data in a data server (i.e., Go to Top and Delete Record).

Not only are data windows powerful additions to your applications, but they are also easy to create. Using the Window Editor’s Auto Layout feature, you can quickly link a data window with one or two data servers, creating either a single-server or master-detail data window, respectively.

When you use Auto Layout, CA-Visual Objects automatically creates a fixed text caption and edit control for every available field in the associated data server(s). (See Types of Controls later in this section for details about these controls.)

For instance, here is an example of a data-aware window created using the Auto Layout feature:


Captions and edit controls created for each field
Tool palette
Properties window
CA-Visual Objects - [CustOrd in App Windows of Order Entry]
File Edit View Tools Application Debug Window Help
CustOrd_DETAIL
CustOrd (3 , 4) (239 x 258)
Data Window Properties

Custnum #:	
First Name:	
Address:	
City:	
State:	
Zip Code:	
Phone:	
Fax:	
Last Name:	

Property	Value
Name	CustOrd
Icon Name	<Auto>
Data Server	CUSTOMER
View As	#FormView
Font	<Auto>
Notify Event	
Accelerator Table	<Auto>
Menu	STANDARDSHEL

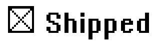
Types of Controls

The tool palette in the Window Editor contains a host of buttons representing different controls. (If you prefer, the Window Editor also features an Edit Select from Palette menu command, which allows you to place controls by choosing commands from a menu.)

The following is an overview of some of the various types of controls you can create (they are listed in alphabetical order).

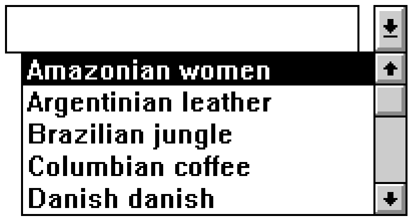
Check boxes indicate a set of options that are either on or off. If more than one check box is present on a window, the user can select as many as are applicable. The state of a check box is indicated in the box to its left: if there is an X in the box, it is selected; otherwise, it is not.

You might use a check box on a data window to indicate a logical field. Checking the box would indicate a value of TRUE, while unchecking it would indicate a value of FALSE. For example:



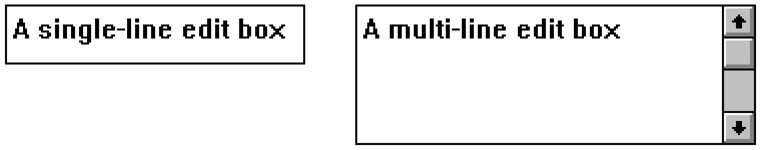
Combo boxes are list boxes with a single-line edit control attached at the top. The user can either type a value directly into the edit control, or click on the down-arrow button to the right to open a list box from which to make a selection. The selection is used to fill in the edit control, which can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

In a data window, you can use a combo box instead of a list box when the field value has more possibilities than you care to list. By placing the most commonly used values in the associated list box, you give the user a quick way to make a selection, without removing the flexibility of entering values that are not listed, as shown here:



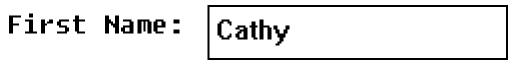
Edit controls present a blank area on a window into which the user can enter data from the keyboard. They come in two varieties, as shown below: single-line for entering one line of text, and multi-line for entering several. The user can edit the text in an edit control with the normal mouse and menu commands.

Edit controls are probably the most commonly used controls on data windows and are often used to represent fields into which the user may type almost any value.

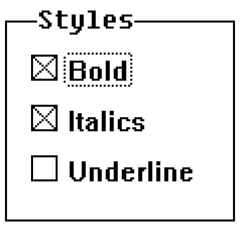


Fixed icons are graphic pictures that can be placed anywhere in a window. They are created with any graphics application, including the CA-Visual Objects Icon Editor. An example of a fixed icon is the question mark common to many warning dialog boxes:

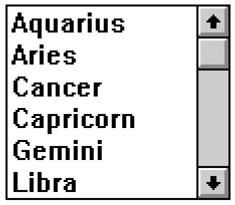
Fixed text displays a caption or label anywhere within a window. A common use of this type of control is to create a caption for a single-line edit control, a feature that is utilized by the Window Editor's Auto Layout feature. For example:



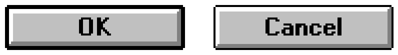
Group boxes visually indicate a set of related controls. They provide a caption to describe the controls, but serve no other purpose. They are most often used to display a group of related check boxes. On a dialog box, for example, you might give the user the option of choosing several styles for displaying text:



List boxes display a list of choices to the user and allow the user to scroll through them and select one. In a dialog box, you might use a list box to allow the user to select a file name. On a data window, you might use a list box to display all possible values for a particular field. For example:



Push buttons react when the user chooses them by generating an event (see Controls and Actions earlier in this section). Some examples of push buttons are the standard OK and Cancel push buttons, shown below, used to close a dialog box or a Commit push button on a data window that commits the edits you have made.

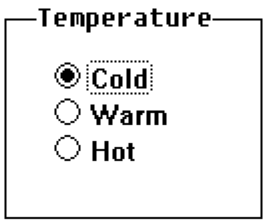


Radio buttons behave like check boxes unless contained in a radio button group box (described below), but their appearance is different. A selected radio button contains a black dot, as shown in the next illustration.

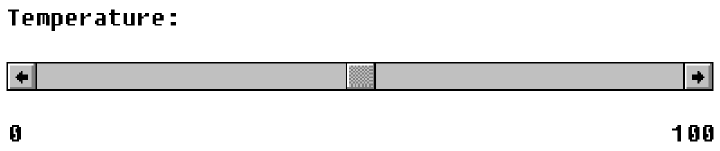
Radio button group boxes visually indicate a group of radio buttons. Like a regular group box, they provide a descriptive caption for the controls they contain, but they have another special purpose—only one of the radio buttons within a radio button group box can be selected at any time. When the user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

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

Radio button group boxes let you use radio buttons to present a set of choices to the user. For example, you might use a radio button group box on a data window to fill in a field that can only take on a limited number of values, such as a Temperature field that must be either “Cold,” “Warm,” or “Hot”:



Scroll bars display a gauge that the user can adjust using a scroll box or scroll arrows. They come in two varieties: horizontal and vertical. You could use a scroll bar on a data window to graphically represent a numeric field. For example:



Note: The scroll bars discussed here, although visually and functionally identical, do not apply to the windows themselves, but rather to the data that the window displays. Window scroll bars (and scroll bars in list boxes and combo boxes) are handled dynamically in CA-Visual Objects applications, depending on their current size and the amount of data that needs to be displayed.

Sub-data windows are simply data windows that you place on other data windows as controls (they are also referred to as subforms). Typically you would use a sub-data window to show a master-detail relationship between two related data servers. (We will create such a data window later as part of the tutorial in the next chapter, “Learning the Basics.”)

Menu Editor

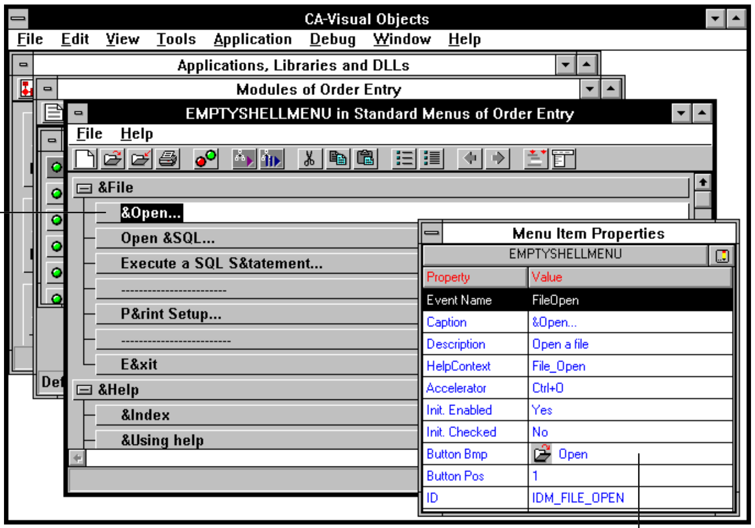
The Menu Editor, shown below, provides a powerful yet easy way to create menus and toolbars for your applications.

Auto Layout

First of all, like the Window Editor, the Menu Editor features an Auto Layout feature. In the Menu Editor, however, Auto Layout is used to add one or more predefined, standard menus to an application. For example, at the touch of a button, you can add File, Edit, View, Window, and Help menus to your application. In addition, each of these predefined menus (like File) contains a set of default menu items (for example, New, Open, and Save), for which default properties are already supplied, including event names and toolbar buttons.

For example, the following shows the properties initially set for the predefined File Open menu command:

File Open command


CA-Visual Objects
File Edit View Tools Application Debug Window Help
Applications, Libraries and DLLs
Modules of Order Entry
EMPTYSHELLMENU in Standard Menus of Order Entry
File Help
&File
&Open...
Open &SQL...
Execute a SQL S&tatement...
P&rint Setup...
E&xit
&Help
&Index
&Using help
Def
Menu Item Properties
EMPTYSHELLMENU

Property	Value
Event Name	FileOpen
Caption	&Open...
Description	Open a file
HelpContext	File_Open
Accelerator	Ctrl+O
Init. Enabled	Yes
Init. Checked	No
Button Bmp	Open
Button Pos	1
ID	IDM_FILE_OPEN

Properties initially defined for File Open

Auto Layout provides a quick way to get started with your menu structures—you can use the resulting menus as is, or you can customize them as desired to fit your application. Of course, you can easily create your own custom menu structures in the Menu Editor.

Creating Toolbars

For each menu structure you create, you can enable or disable a corresponding toolbar. If enabled, you can choose which items in the menu structure should have corresponding buttons displayed on the toolbar, as well as which graphic should be used to represent each item. In addition, if desired, you can choose the File Preview Toolbar menu command while in the Menu Editor to preview a menu structure's toolbar.

Menu Items and Events

Like some window controls, an important property of items on a menu is an event name. This is because menu items, like certain window controls, initiate actions or *events*. The Menu Editor makes it easy for you to associate actions with menu items using the *event name* property, exactly as previously described for the Window Editor.

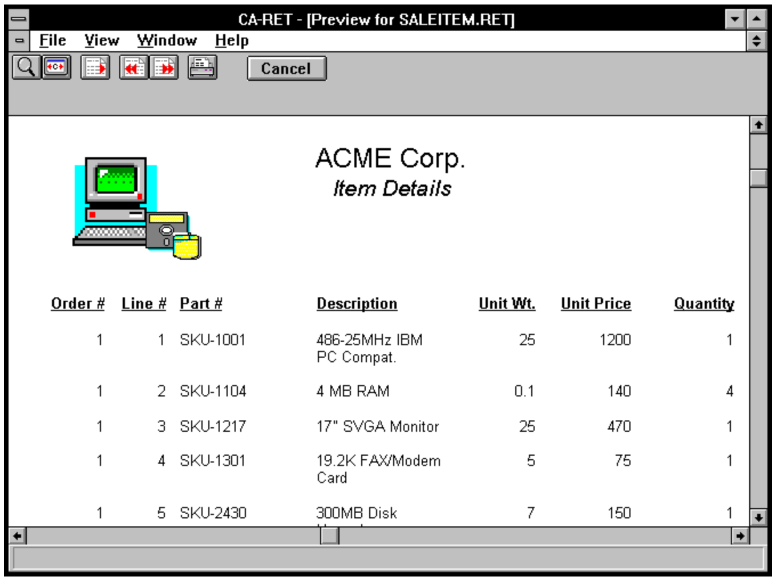
CA-RET

CA-RET is a state-of-the-art report publishing tool that has been integrated with CA-Visual Objects to allow you to create sophisticated reports.

CA-RET has its own environment that you can use to create, edit, and print reports. To create a report, simply choose the Tools Report Editor command to launch CA-RET from directly within the CA-Visual Objects IDE.

Then, choose a *data server* (DB or SQL) and a *report style* (CA-RET provides the following report styles: tabular, form, mailing label, form letter, and freestyle), and enter a name for the report definition.

Based on those choices, CA-RET lays out an initial report, which can then be customized as desired. For example:



ACME Corp.
Item Details

Order #	Line #	Part #	Description	Unit Wt.	Unit Price	Quantity
1	1	SKU-1001	486-25MHz IBM PC Compat.	25	1200	1
1	2	SKU-1104	4 MB RAM	0.1	140	4
1	3	SKU-1217	17" SVGA Monitor	25	470	1
1	4	SKU-1301	19.2K FAX/Modem Card	5	75	1
1	5	SKU-2430	300MB Disk	7	150	1

CA-RET's initial report can be customized by adding report details like literal, database, and computed fields; CA-RET spreadsheet-like functions; and text. You can also add graphics using CA-RET's drawing features. Typeface, color, and size can be selected for your report text, and you can add bolding, italics, and other print features.

Once a report has been designed and saved, in addition to saving the report on disk as an .RET file, CA-Visual Objects automatically generates object-oriented code that you can use to access the report (for example, print it or allow the user to preview it on screen) from within your application.

The integration of CA-RET makes it easy to incorporate professional-quality reports into your CA-Visual Objects applications. And, since CA-Visual Objects also includes the royalty-free CA-RET Runtime, you can deliver this powerful report technology free with any of your applications.

Note: Refer to the *IDE User Guide* and *Programmer's Guide, Volume II* for more in-depth information about using CA-RET with CA-Visual Objects.

Icon Editor

The Icon Editor is launched using the Tools Icon Editor command. Using the Icon Editor, you can create custom icons and cursors for your applications using a drag-and-drop interface that allows you to work with several images at the same time. Icons and cursors are saved in standard Windows .ICO and .CUR files (respectively) that can be defined to your application as resources.

The Debugger

The CA-Visual Objects Debugger provides advanced tools for tracking and correcting errors that occur at runtime. For example, you can:

- Control the execution of your application while viewing the source code in the Source Code window
- Execute any part of your application using one of several execution modes, including a mode in which you step through the code one line at a time
- Conditionally stop program execution using breakpoints or by pressing Ctrl+Alt+SysRq at any time
- Monitor watch expressions in a separate window
- Evaluate expressions on-the-fly
- View and modify variables of all storage classes
- View database, index, and other work area information in a separate window
- View and modify system settings

In addition, CA-Visual Objects allows you to set debugging options at *any level*—for a single entity or module, or for an entire application—and to override the current default setting at the next lower level.

This means, for example, that if you have a successful, stable application and decide to add new features to it, you can save valuable time by testing and debugging only the new module or entity. It also means that, with new applications, you can debug the 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.

What’s Next

This chapter has given you an overview of the browsers and visual editors that make up the CA-Visual Objects IDE. These tools provide an immediate means for you to examine and control your applications and will become even more useful as your application increases in sophistication. Building upon the tour provided in this chapter, the next chapter teaches you how to use CA-Visual Objects with a “hands-on” tutorial.