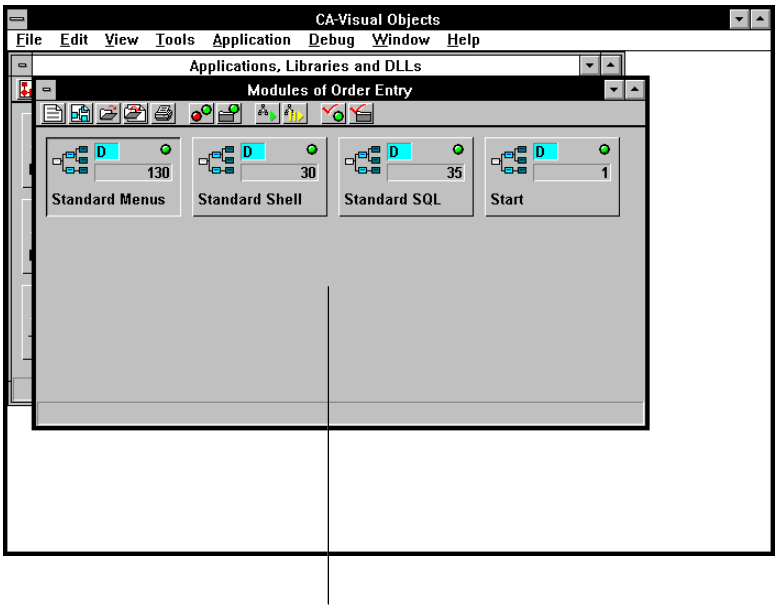
A Tour of the CA-Visual Objects Desktop

Your desktop should now look as follows:



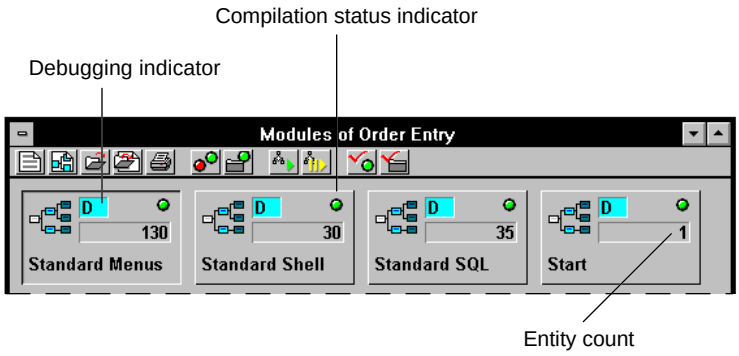
New Module Browser with four modules

By simply completing three dialog boxes, your new application already contains four modules: Standard Menus, Standard Shell, Standard SQL, and Start. Let's take a closer look at this Module Browser.

Note: The Standard SQL module is generated because we included the SQL Classes library when we originally created this application. If you do not require access to SQL data sources in your own applications and do not include the SQL Classes library, this portion of the Standard Program will not be generated.

Using the Module Browser

Every Module Browser includes some built-in details that point out useful information about the modules in an application:



Debugging Indicator

For example, there is a *debugging indicator* that lets you know whether the debug flag for a particular module is on (D+), off (D-), or controlled by the application to which it belongs (D).

Compilation Status Indicator

The Module Browser also features an LED-style *compilation status indicator*. Compilation status is indicated by color: *green* means compiled successfully, *red* denotes that the module needs to be compiled (either because it has compilation errors or because one of its entities has been modified since the last build), and *yellow* indicates that it has never been compiled.

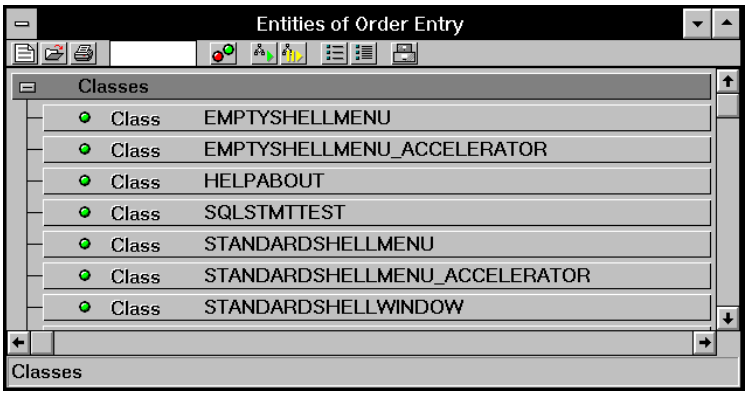
Note: If desired (for example, if you are working with an LCD monitor), you can change the indicators so that they use symbols rather than colors to denote compilation status (a check mark for green, an X for red, and a question mark for yellow). Choose the System Options command on the File Setup menu, and turn off the Color LEDs check box.

Entity Count

Finally, there is an *entity count* for each module, indicating the number of entities in a particular module. An *entity* is the smallest named unit of the application, such as a function or method definition. For example, the Start module has one entity, the App:Start() method.

Using the Application-Wide Entity Browser

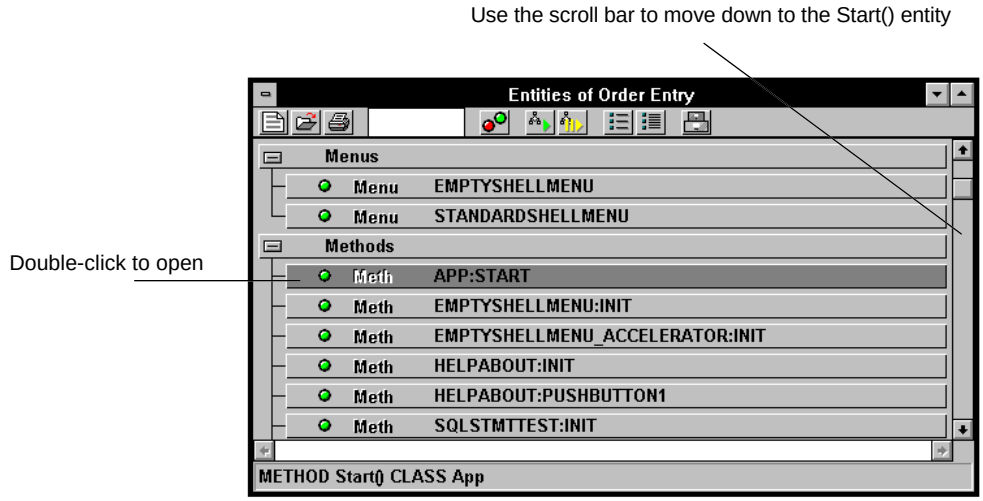
Let's now take a look at the various entities created by CA-Visual Objects by choosing the Tools Entity Browser command. Doing so displays an application-wide Entity Browser, listing every entity in the new application:



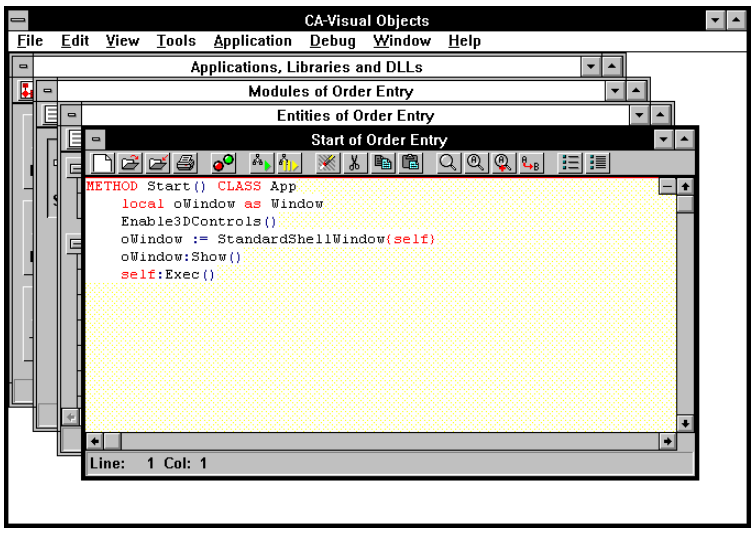
If you scroll through this Entity Browser, you can see that the Standard Program contains a host of entities, including classes, defines, menus, methods, and resources. Displaying these entities in the application-wide Entity Browser provides a useful and customizable means of viewing them (for example, you can collapse the tree or set a name filter).

Accessing the Editors

The Entity Browser also offers you easy access to these entities. Double-clicking on an entity in the Entity Browser displays that entity in its associated editor. For example, scroll down to the Methods branch, find the App:Start() method, and double-click on it:



CA-Visual Objects displays its code in the Source Code Editor:



A Closer Look at the Application

The App:Start() method is a good place to begin exploring the structure of the Standard Program, and we will take a closer look at it in this section along with the rest of the generated source code. But first, it will help you to understand some of the basic concepts underlying the structure of an MDI application.

MDI Application Structure

The Windows

MDI applications are structured around the presentation of multiple windows. They typically use a *shell window* as the main, or "owner," window. The documents that are opened in the shell window are referred to as *child windows*. Child windows are owned by the shell.

In the Standard Program, the shell window is created using the StandardShellWindow class. This class inherits from the ShellWindow class in order to add more functionality to the basic shell window, while preserving what is already there (for more information on inheritance and subclassing, refer to "Object Oriented Programming Concepts" in this guide). Thus, characteristic of most MDI applications, the Standard Program defines a shell window in which you can open any number of child windows.

When we speak of child windows, we are referring to windows that are owned by the shell window. These windows are typically derived from one of two classes: ChildAppWindow or DataWindow.

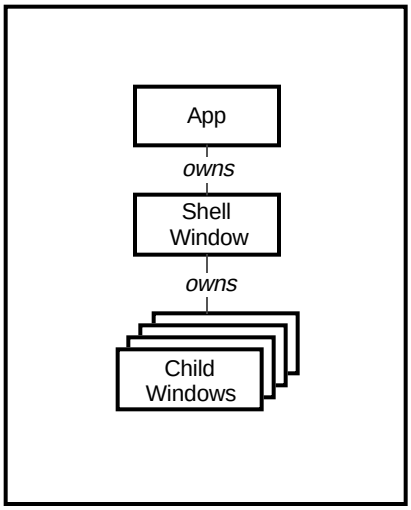
The DataWindow class actually inherits from the ChildAppWindow class and adds functionality to support linking the window directly to a database through a data server. Thus a data window is a kind of child window. We, therefore, use the terms "child window" and "data window" interchangeably in this discussion.

Note: An SDI application, by contrast, is structured around displaying a single document at a time. Typically, it uses a top window (based on the TopAppWindow class) as the owner window, and a ChildAppWindow or DataWindow as the child. The difference is that with a top window as owner, only a single child window can be open at a time.

The App Object

Just as each child window has the shell window as its owner, everything in a CA-Visual Objects GUI application has an owner. This is a very important concept that controls much of the action in the application.

At the highest level, the topmost window in an MDI application—the shell—is owned by the *App*, an invisible object that controls the basic event processing of the system.



The App object represents the overall application—it starts, runs, stops, and handles all events (such as mouse clicks and keystrokes) in the application.