Yes, Virginia, There Are Objects In PRO/5®, Part Two

By Michael Martinez

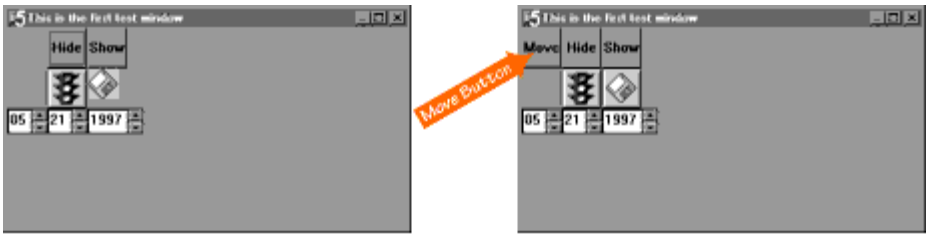
In our last issue we discussed the advantages of implementing objects through object managers, special programs which contain the methods (functions and procedures) that control object behaviors. By communicating with an object manager we gain indirect access to an object created and manipulated in virtual memory. These are virtual objects, virtual because they are created in memory and there is no real or direct access to them. An object is a self-contained program that deals with one or more specific types of data. It responds to messages it receives and can in turn send messages to other objects.

Object management is a technique devised originally for large systems which implemented objects alongside older applications. It became necessary to develop "wrappers" or interfaces to the objects for applications developed with older or foreign compilers so their functionality was not lost. Through object management, older applications can benefit directly from object technology.

The benefits derived from using object management extend to graphical objects as well as virtual objects stored in memory. You can easily define a class of controls - say, a date control -for which there is no native support in Visual PRO/5™. This class can implement methods such as "create," "destroy," "set," "move," "hide," "show," "get month," "get day," "get year," etc. The date controls are actually aggregates constructed from simpler control types such as a child window, edits, and scroll bars. Visual PRO/5 tells you which context (window) the events generated by the aggregate control come from, so you can easily determine whether an object manager for the aggregate control should respond to the events.

An example code block which creates a window with some controls (including an aggregate date control) through object managers is provided here. It uses the object managers FnWindow_Class, FnButton_Class, FnTool_Button_Class, and FnDatebox_Class.

The purpose of the date box is self-evident. The "HIDE" and "SHOW" buttons let the user hide the button labeled "MOVE" and the two tool-buttons. When the tool buttons are hidden, they reappear if the mouse is passed over their boundaries (see *Screen Capture 2*). The "MOVE" button moves itself from row 1 to row 2 and vice versa.



Screen Capture 1 Screen Capture 2

The classes' object managers are implemented as multiline functions, and the program shown when following this link creates the window. The functions are defined in their own include files which are too long for inclusion in this article. This sample demo, with all source code, will be made available for download from the BASIS FTP server-ftp.basis.com-in the pub/online-documentation-project/ directory as objclass.exe. The demo will be released when *Kilauea*ships. The include file "make_win.inc" contains the code detailed above.

Object-oriented programming has gotten a lot of bad press through the years. It's slow. It's cumbersome. It's hard to learn. These criticisms apply to various languages. Some pure OOP languages, such as Smalltalk and Eiffel, include many base object types, whereas some hybrid languages like C++ and Delphi require you to build your own object types from their limited base types. Every language has its strengths and weaknesses.

A strength of Business Basic has always been that if it didn't support a native data type you could define your own and develop code to support it. By creating object managers you can encapsulate your specialized type-supporting code (i.e., keep it from getting scattered throughout your application, where it's subject to tinkering) and at the same time provide flexible, time-saving routines for rapidly creating interesting user interfaces and application structures.

An object manager can make use of other object managers. Instead of adding new methods to an existing object manager, or rewriting existing methods, in order to extend the capabilities of your objects, you can create a new object manager which invokes the old one whenever a desired method already exists. There are two immediate advantages to using calls to an existing object manager when writing a new one:

1. Existing code, which has been tested, is not altered.
2. When debugging the new object manager, you know where to look for problems.

The potential benefits we can derive from using object-oriented design paradigms in Business Basic applications have only barely been touched upon. In the future, especially when the *Pinatubo* interpreter emerges, BASIS will help unravel some of the myths and mysteries surrounding the powerful concepts found in object-oriented languages and applications. But we don't have to wait for the future. It's already here in some ways, and things will just get better.