Getting PM Programs Up and Running
OS/2 Professional February 1994

Gpf 2.1

BY STEVE MASTRIANNI

A friend of mine who is a Presentation Manager Guru once told me that in more than six years of writing PM programs, he wrote only the first one from scratch. How could this be, you might ask? Simple: all PM programs look and operate essentially the same way, so new programs can be assembled from previous code with simple cuts and pastes.

This basis in a common architecture and functionality lends itself well to CASE tools that can draw on a database of standard functions. Gpf is such a tool.

Writing PM programs is a fairly time-consuming chore. As a software developer, time is very important to me; if I can find something that saves development time, I'll always buy it in favor of writing it myself. I was pleased to find that Gpf not only saved coding and debugging time, but actually made writing PM applications easy and enjoyable.

Gpf provides an excellent set of tools for generating and maintaining PM code. You first design your PM application in a prototype environment, and then run it with an emulator to test the operation. Once you're satisfied, you simply click on the code generator option and Gpf generates, compiles, and links the code automatically. Of course, the PM section is only a part of your program, so you still have to add your application-specific code.

Getting Acquainted

The Gpf tutorial walks you through the development of three PM applications at increasing levels of difficulty. Since tutorials are a handy way to learn a product from the vendor's perspective (not to mention a typical first step for many first time users in a foreign environment), I began working my way through the exercises.

The first tutorial emphasizes the basics of Gpf as you create a simple PM application with one main window and a logo window. I immediately ran into my first problem; the tutorial said to start the Gpf Editor, but the icon was nowhere to be found. I clicked on the Interface Builder icon in the Gpf folder, and low and behold, the Gpf Editor screen appeared. With this minor problem behind me, I continued working through the instructions, creating the first application.

The tutorial's layout could use some improvement. I made a few mistakes because I missed a few instructions along the way; some of

the instructions sit too close to the screen pictures and are easy to overlook. Once I corrected my mistakes, the application prototype worked perfectly. Eager to take a look at the compiled code, I broke away from the tutorial and invoked the Gpf code generator (the topic would normally be covered in the second tutorial). I checked the code by compiling and linking it, and it worked exactly like the prototype without requiring a single manual change to the generated source code.

The second tutorial introduces the details of creating and applying a presentation object, an icon object, a dialog window, an action pushbutton, a bitmap object, group boxes, and radio buttons. To help test these more complex programs, Gpf includes an animation feature (which I used on the prototypes until they worked as I expected).

Again, the application worked exactly as outlined. However, in addition to my missing a few instructions, I encountered a few more problems along the way. Several times, Gpf would not let me access the source file, reporting a File 1/O Error Gpf006. The only way I could correct this was to close Gpf and restart it. In one case, my source file was corrupted, and I had to recreate it.

Adding Code

In addition to the extensive tutorial (which spans more than 100 pages), the manual includes some 30 additional pages of programming considerations for using Gpf. This is required reading if (like most programmers) you intend to add your own code to the Gpf-generated code. If you modify the generated code, you cannot go back and use the code generator again. Should you try, Gpf will overwrite the existing source, thus removing all of your changes. While Gpf automates the generation of the Presentation Manager code, that may be only a small percentage of your application.

To overcome these limitations, Gpf lets you create User Function Objects that can be used to modify the Gpf-generated source code safely. You can edit your custom source using any standard editor, and paste into the Gpf source via the clipboard. The preferred method is to store your functions in a DLL. You then place your function definitions in a custom header file with the same name as your application’s definition (.ORC) file. At this point, Gpf will automatically recognize the functions.

By following this technique, you can modify and extend the functionality of your Gpf application without directly modifying the code generated by Gpf. Alternatively, you can include your source code with an #INCLUDE “myfunc.c” directive. I find this method less desirable while reading my code listings, however, as I don’t have all the source in one place.

The balance of the manual includes a 200-page Gpf reference describing the various pull-down menus and operations, the Gpf functions and

parameters, as well as instructions for writing your own controls for inclusion in your Gpf application. I had trouble reading some of the code examples due to the narrow pages; much of the code text wrapped to the next line. Fortunately, the examples also reside on the disk.

As you develop a library of resources, Gpf Tools (a set of object-oriented tools) helps you build new .ORC files by reusing objects from previous applications. You build the library of user interface objects by simply clicking on objects and dragging them to an .ORC file. Gpf Tools will also generate ASCII and INF files of the definition as well as help panel text.

The Code Worked

I wrote a few simple PM programs to test the usability of Gpf in a real environment. In every case, Gpf produced code that worked perfectly and exactly the same as in the test environment. The code was clean, well-documented, and made building the PM front ends a snap. Gpf is an excellent prototyper and a great way to get Presentation Manager code up and running. It was quite satisfying to get a Presentation Manager front end for a simple application up and running within a few hours instead of a few days or weeks.

Gpf has a few shortcomings, however. Since important variables like window handles are kept in global memory, Gpf has trouble handling multiple instances of the same window. Gpf also does not allow you to specify resource constants.

The lack of support for User Interface Class Libraries, or ICLUI and other cross-platform class libraries, may limit Gpf's usefulness in a true object-oriented development environment. Gpf plans to offer support for cross-platform class libraries in the future.

If you need to get your Presentation Manager program up and running in the shortest amount of time, Gpf is the tool to use. It will pay for itself in the first few weeks of use, and it will allow you more time to concentrate on your actual application.♦

Steve Mastrianni is president of Personal Systems, Inc., a Canton, Connecticut consulting firm specializing in device drivers and realtime applications for OS/2 and Windows NT. He can be reached at 71501.1652@compuserve.com.