Issue Date: FoxTalk September 1997

Instrumenting FoxPro Applications

Rod Paddock

You've spent days, weeks, months, or years putting together the perfect application -- but what do your users think? Specifically, are they making full use of it, or are there pieces that are underused, or even unused? In this article, Rod discusses a technique to measure which pieces of your application are being used and to what extent.

It's always fun how these articles come together. I was sitting in the Toronto airport with Whil Hentzen talking about -- you guessed it -- FoxPro development. During this discussion, we came to the topic of navigation buttons and whether or not users actually use them. It was my contention that users don't use navigation buttons because they're not a very convenient method of finding any real information. Whil wasn't so sure I was right. So I asked, "Have you ever instrumented your applications to find out?" Whil said he had used a similar technique for testing, but not in a production application to track the actual usage of an application. I stated that it's important to instru-ment applications to see just what people are actually doing with them. Subsequently, Whil asked me to write the article you're holding in your hands now.

Software developers have been faced with the problems of feature bloat for years (a.k.a. Rampant Featuritus). In my first MIS job, I was tasked with the job of producing the daily, weekly, and monthly reports that were used to run our company. There were literally dozens of reports to be run on a periodic basis. After a time, I began to question the need for so many reports and whether people were actually reading them. So I decided to conduct a little experiment -- I stopped creating and sending the reports. Now guess what happened: Only a few people asked for the reports. Through this rather unscientific study, I was able to trim down the number of reports being created by more than half. Why did this work? Because people who needed the reports "screamed," but everyone else remained quiet. Why were so many reports abandoned? Mostly because business needs had changed. Information that was once useful was no longer so valuable. Eventually the job of running reports was placed directly into the hands of the users, thus eliminating my intervention in the report generation process. However, I still wanted to know what people were up to when it came to the reports. This required me to take a different approach.

Applications have a tendency to grow in both complexity and functionality over their lifetimes. Eventually your applications might become bloated with an excess of underused functionality. Your applications might have too many reports or functionality that simply isn't used. Over time, this bloated condition will cost you a great deal of time and money, because bloated functionality needs to be constantly upgraded as the application changes. So the question arises: "How do you find these bloated areas of your application and eradicate them?" You can do two things to accomplish this:

1. Sit with *all* the users of your software and see how they use it; or
2. Instrument your application to track how it's being used.

The first option isn't very realistic, so you need to take a closer look at the second. Instrumenting your application is actually a simple operation. You simply need to figure out which areas of your applications evoke actions and put code in those areas to track their use. The Windows 95 interface guidelines provide seven basic tenets of GUI design. The first says that your application should follow the object-action paradigm. What this means is that certain controls are used to invoke application functions. In VFP development, two come to mind: menus and command buttons. These two control types are the primary controls used by developers to activate application functionality. With this in mind, you can begin instrumenting your applications.

Before you can begin, you need to take a look at the types of information you should be tracking. You should gather the following:

1. What operation was run?
2. Who ran it?
3. When did the user run it (date and time)?
4. How long did the operation take? (This one is optional.)

Instrumenting menus

The first task on the list is to place code in your menus to track their usage. To track menu usage, you need to do one of two things: 1) place your instrumentation code directly into each menu pad; or 2) place your code into your central application object that launches different application components. In either case, you still need to add code that tracks usage from menus. The following code does this:

```
*-- This function instruments menus.
Function g_insmnu
Lparameters pcPrompt, pcPad
*-- Open instrumentation file
If Not Used("Instrument")
  Use Instrument In 0
```

```
Endif
*-- Get the text from the menu pad
Local lcPadPrompt
lcPadPrompt = prmpad("_msysmenu",pcPad)
Insert Into Instrument (calling_source, ;
  program_name, other_name, call_date, call_time) ;
  Values ;
  ("MENU", pcPrompt, lcPadPrompt, date(), time())
Return
```

To call this function, place the following function call into your menu bars or your global launcher application:

```
g_insmnu(prompt(), pad())
```

The primary job of this code is to write your usage information to a file whenever someone hits a menu bar. This code uses functions built into VFP that allow you to collect information about menu hits.

Instrumenting command buttons

Your second task in your instrumentation project is to track the use of command buttons. Instrumenting command buttons should be a rather straightforward task; you simply place code into your command button base class that logs your instrumentation information. The following code can be used to instrument command buttons:

```
*-- This function instruments a command button
Function g_inscmd
Lparameters poCommandButton, poForm
*-- Open instrumentation file
If Not Used("Instrument")
  Use Instrument In 0
Endif
Insert Into Instrument (calling_source, ;
  program_name, other_name, call_date, call_time) ;
  Values ;
  ("CMD", poCommandButton.caption, poform.Caption, ;
    date(), time())
Return
```

To call this function, place the following code into the click event of your buttons:

```
g_inscmd(This, Thisform)
```

So far, this article has covered creating code that can be called from menus and command buttons. You need to be aware that people can launch forms or application components from alternative sources. The two that come to mind are check boxes and option groups. If your application launches applications from these or other types of objects, you might need to alter the previous code for insertion into these styles of controls.

Instrumenting forms

There is another interesting object you might want to track. How about forms? Forms are an interesting type of object to instrument because they can be called in many different ways. Forms can be called from a menu, a command button, as part of a form set, or they can be opened multiple times. This will require you to think a little differently about how forms are instrumented. The following function can be used to instrument forms:

```
*-- This function instruments forms
Function g_insfrm
Lparameters poForm
*-- Open instrumentation file
If Not Used("Instrument")
  Use Instrument In 0
Endif
Insert Into Instrument (calling_source, ;
  program_name, other_name, call_date, call_time) ;
  Values ;
  ("FRM", poForm.caption, SYS(16,1), ;
    date(), time())
Return
```

The most important piece of this function is the SYS(16,1) call. This call shows where the instrumentation code was called. You could also modify this function to loop through the calling stack and store the results into a memo field. This would allow you to track the different areas of your application that might call a common form.

To call this function, you can place the following call into the Activate, Deactivate, and Init methods of your forms. This will

show how your form is being used in conjunction with other forms:

```
g_inscmd(Thisform)
```

Analyzing the results

After running the instrumented version of your software, you can begin analyzing the data that was tracked. All you need to do is create a few queries and reports that will answer the two critical questions: What features of your application are people actually using, and how often are they using them?

The first item you will want to query is how are people activating items from your menu. The following query generates a count of menu hits grouped by the menu pad and the prompt. Querying the data using this method takes care of the situations where you have menu prompts with the same names. The query lists the most active prompts at the top.

```
Select program_name as prompt, ;
       other_name as pad , ;
       sum(1) as menucount ;
  From Instrument ;
 Where calling_source = "MENU" ;
 Group By 2,1 ;
 Order By 3 Descending ;
  Into Cursor c_menucalls
```

The second query generates results from the command button hits. It groups by form name, then caption from the button. It also lists the most active buttons on top.

```
Select program_name as button, ;
       other_name as form , ;
       sum(1) as menucount ;
  From Instrument ;
 Where calling_source = "CMD" ;
 Group By 2,1 ;
 Order By 3 Descending ;
  Into Cursor c_buttoncalls
```

As you can see, with a few simple queries you can gather some very interesting information about your application and its usage. What I've found in my instrumentation analysis is that the 80/20 rule is very true -- people use 20% of your applications 80% of the time. Does this mean that you can go in and start ripping out code once you've performed an analysis? *No!* You need to make sure you run your analysis for a realistic timeframe. This ensures that functions that aren't called frequently (such as month-end closing procedures) will have ample time to be instrumented.

Once you've found a function that you want to turn off, you can simply replace the function calls with messages like: "Contact MIS to reactivate this function." This gives users a chance to put the function back at a later date is they need it. After a predetermined time, you can remove the function permanently if no one has requested it.

Summary

With a few simple functions and queries, you can gather some rather useful information about your applications. In order to prevent application feature bloat, you need to analyze your applications and how they're used with some objective statistics. Instrumenting your applications gives you the information you need to do so.