August, 2002

Advisor Answers

Shutting down abandoned applications

VFP 7/6/5/3

Q: I'm using Automation to Word in an application. Occasionally, things go wrong and Word is left running behind the scenes. Is there any way I can find and kill all instances of Word that are not visible?

–John Hosier (Naperville, IL)

A: Not surprisingly, this is one of those problems that can't be solved with just native VFP code. But the Windows API gives us the tools we need.

As long as you have an object reference to an application you're automating, you can control it, including shutting it down whether it's visible or not. Once you make an automated application visible, the user can shut it down manually. The problem occurs when your application fails and leaves the automated application (Word, in your case) running without making it visible.

Here are the steps we need to clean up in this situation:

1. Find each instance of Word.
2. Check whether the instance is visible.
3. If it's not visible, stop it.

The most widely known way to get your hands on a running application is with the FindWindow API call. FindWindow takes the window title as a parameter and returns the window handle. Declaring and using it looks like this example, where we'll grab a handle to the Calculator applet.

```
DECLARE LONG FindWindow IN WIN32API ;
   STRING lpClassName, STRING lpWindowName
nHandle = FindWindow( NULL, "Calculator" )
```

To use FindWindow in this way, you need to know the application's title, that is, the contents of the title bar. However, Word (like the other Office applications) makes that difficult because its title bar changes based on what document you're editing.

As the declaration and call indicate, FindWindow offers another approach. Instead of passing the window's title, you can pass the application's class name. The class name identifies the application rather than the window. For some reason, the class name for Word is "OpusApp", so you can find a Word window like this:

```
nHandle = FindWindow( "OpusApp", NULL )
```

However, this approach finds a single Word window, not all Word windows, and there's no way to get from the window found to the next Word window.

Fortunately, there's another way to tackle the problem. Rather than looking for a particular window, we can go through the entire list of windows, and check each to see if it's one we want. The secret is that all the windows we're interested in are "children" of the Windows desktop. (There are lots of other windows around, but applications are always one level below the desktop.)

To cycle through all the windows that belong to the desktop, we first need a handle to the Windows desktop. The GetDesktopWindow function takes care of that.

```
DECLARE LONG GetDesktopWindow IN WIN32API
lnDesktopHWnd = GetDesktopWindow()
```

Once we have the desktop window, the GetWindow function can cycle through all its children. Here's the declaration:

```
DECLARE LONG GetWindow IN WIN32API LONG hWnd, LONG wCmd
```

The first parameter is a reference window. The second parameter, wCmd, tells which window handle to return in relation to the reference window. Pass the constant GW_CHILD to get the first child. Pass GW_NEXT to get the next child. (Of course, you need to declare these constants to make them available.) This loop cycles through all the children of the desktop window and displays their window handles.

```
#DEFINE GW_CHILD 5
#DEFINE GW_HWNDNEXT 2
lnHWnd = GetWindow( lnDesktopHWnd, GW_CHILD )

DO WHILE lnHWnd <> 0
   ? "Next child is ", lnHWnd
   lnHWnd = GetWindow( lnHWnd, GW_HWNDNEXT )
ENDDO
```

Now that we have a way to find every application-level window, we need to figure out which ones belong to the application we're interested in. Back to the class name. The GetClassName function takes a window handle and finds the class name for that window. It returns the length of the class name, while storing the name in its second parameter, which must be passed by reference. As with many other API functions, in order to get a string back, you have to create the string variable first, initializing it to an empty string of the maximum length and pass both the variable and the length.

```
DECLARE LONG GetClassName IN WIN32API ;
   LONG hWnd, STRING lpClassName, LONG nMaxCount

lcClass = SPACE(256)
lnLen = GetClassName( lnHWnd, @lcClass, LEN(lcClass) )
lcClass = LEFT( lcClass, lnLen )
```

The next step is determining whether the Word window is visible. Not surprisingly, that's the function of the IsWindowVisible API function. Using this one is simple. You pass a window handle and it returns a numeric value. A return value of 0 means the window is invisible; otherwise, it's visible.

```
DECLARE LONG IsWindowVisible IN WIN32API LONG hWnd
lnIsVisible = IsWindowVisible( lnHWnd )
```

Finally, we need a way to close a window, once we've determined that it's an invisible Word window. For that, we use the PostMessage API function. Again, you pass the window handle. The wMsg parameter contains the message to pass; the WM_CLOSE constant has the right value. For our purposes, you can simply ignore the last two parameters, which are used to pass additional information about the message.

```
DECLARE LONG PostMessage IN WIN32API ;
   LONG hWnd, LONG wMsg, LONG wParam, LONG lParam

#DEFINE WM_CLOSE 0x10
PostMessage( lnHWnd, WM_CLOSE, 0, 0)
```

We can put all this together to write a KillApp function that seeks out and destroys all abandoned Word instances. You'll find the working function on this month's Professional Resource CD. It accepts one parameter, the class name of the application to kill.

Clearly, knowing the class name for an application is the key to invoking this function. You can use the GetClassName function to find the class name for an application you're running. (Use FindWindow, if

necessary, to get the window handle.) Visual Studio comes with Spy++. This tool lists all existing windows in a treeview and lets you examine all available information about a window. In addition, the articles at the following URL's list the class names for a number of applications.

http://support.microsoft.com/default.aspx?scid=kb;EN-US;q288902

http://support.microsoft.com/default.aspx?scid=kb;EN-US;q88167

http://www.biocom.arizona.edu/tbook/faq/faq_200.htm

You may also want to add a second parameter to KillApp to indicate whether to shut down all instances of the specified application or only those that are invisible.

–Tamar