One Last Compile...

Delphi Goes To Hollywood

Occasionally in life you get what I call 'William Goldman reversal.' For those of you not familiar with the works of Mr Goldman, he's a Hollywood screenwriter and novelist who has written, among many others, *Butch Cassidy and the Sundance Kid*, *All the President's Men*, *The Princess Bride*, and a book about Hollywood called *Adventures in the Screen Trade*. (Then, just to prove that nobody's perfect, he wrote *The Ghost and the Darkness*.) It's in *Adventures in the Screen Trade* that he explains the idea of a 'reversal': a moment which conveys information to the audience that is the opposite of what they've come to expect, or that changes their understanding of the plot, making them suddenly sit upright, put down their popcorn, and start concentrating again.

I was somewhat surprised to have a William Goldman reversal reading Issue 35 of *The Delphi Magazine*. Step forward Hallvard Vassbotn, author of *The Rise and Fall of TObject*. Don't worry, I'm not going to make fun of you, it's just that Mr G himself would have been proud of the jolt you gave me in your second paragraph. And it's lucky for me that you did.

I've always considered myself a careful programmer. In the byways of Windows I liked to think my program was a model driver, always signalling before making a turn, using the litter-bins at laybys, considerate to others on the road. And then I read this:

> *"Everyone knows that you use the* `Create` *constructor to create an object instance and the* `Free` *method to free it. Most know that you have to override the* `Destroy` *destructor to add behaviour when the object instance is going away."*

The key sentence here is the second one, and the key phrase is 'Most know...' Most. But not me. Nor, sadly, did it appear to be common knowledge to the authors of a book I'd cribbed extensively from, who had cheerfully used a destructor `Free` in all of their examples, with nary a mention of `Destroy`. All my objects had the additional behaviour in a destructor `Free`. Hallvard seemed to think this was bad. But everything seemed to work okay in my new program where, for the first time, I'd made extensive use of objects. It compiled okay, ran without any errors, did what I expected it to do. In any case, did it really matter whether I used `Free` or `Destroy`? Just another uptight OO purist, I thought, fussy about the correct protocol. In the back of my mind there was some uneasiness about all this override, abstract and virtual business, but I ignored it.

Still, just to be on the safe side, I fired up Memond to check my memory use. This was a humbling experience. I discovered that, far from being a model driver, my program was more akin to an out of control juggernaut, hurtling at high speed down the back lanes of Windows, throwing hand grenades out of the window and leaving a trail of destruction behind it. When the program closed, doubtless accompanied by a breathless sigh of relief from the operating system, the amount of memory still allocated to it looked like the GNP of China.

It took me a while to figure out what was going wrong (and I'm still not 100% sure about it) but here's what I think was happening. The way I'd written them, each of my objects had two `Free` routines that they could use. They could call the standard `TObject.Free`, or they could use mine. When it came to closing down, I'd just cavalierly said `MyObject.Free`, and the program had gone off and used the standard version. So unless I explicitly used a typecast, ie `(MyObject as TmyObject).Free`, my bloated objects were being cast out into the Windows darkness with all of their data still attached.

So, I'm sorry I doubted you Hallvard, and I'm sorry to all those of you reading this clutching your heads and saying 'This guy should never be let near an object again.' And I'm sorry to those of you who've only read this column because they thought a William Goldman reversal was going to turn out to be something rude.