Custom Coder Utilities

by Stephen Posey

Delphi straight out of the box is generally well thought out and convenient to use, and its code generation capabilities are actually quite remarkable. Nonetheless, there are a number of areas where "template style" code operations could be better facilitated. Fortunately, with a bit of ingenuity, we can use Delphi itself to improve matters.

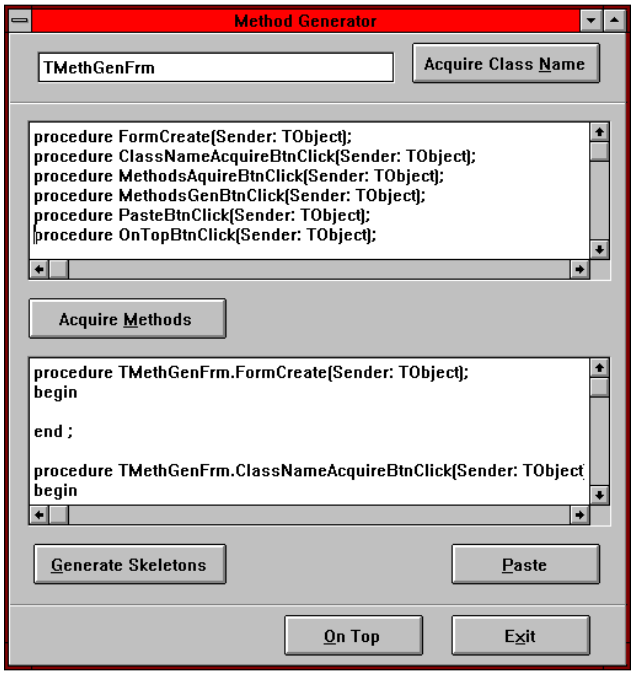
One tedious operation that I found myself repeatedly performing was generating the skeletons for the user defined methods in Forms and Components I was writing. I thought that it would be great if Delphi had a procedure to take the names of any new methods I'd added to a class definition, and then generate the skeleton heading and `begin...end;` pair in the `implementation` section, like it does for event response methods. To address this (at least to a minimal extent) I've created *MethGen*, the Method Generator, a pretty much quick and dirty workhorse to build skeleton methods. It does, however, demonstrate some very useful techniques that can be applied to creating custom coders to handle your own pet coding peeves. Figure 1 shows the utility in action.

A second common activity that begged for automation was creating `MessageBox()` statements, which required remembering (and spelling!) the appropriate `MB_???` constants to get the right style box. To this end I created *MBExpert* which puts up a dialog box to allow you to create `MessageBox()` statements. MBExpert uses some slightly more sophisticated string manipulations to get its work done, but is otherwise similar to MethGen – see Figure 2. Both can be added to Delphi's Tools menu for easy access and the full source is on the disk with this issue of course.

I've included a couple of utility units on the disk too, for your edification and amazement! EXPTTOOL.PAS is one, it contains a number of useful utility routines for accessing other programs, including definitions for a few "undocmented" Windows calls that are surely so common and necessary that they are very unlikely to become unsupported (I've also used them in Windows 95 and they seem to work fine). Both MethGen and MBExpert make use of the `FindChild` function to locate the Delphi code editor window. I discovered the `TEditWindow` and `TEditControl` class names by poking around with a Window Class Inspector that I'm creating based on several sources.

My `FindPartialWindowTitle` function, combined with one of the several Delphi implementations of the Visual Basic `SendKeys()` routine (I use an enhanced version of the one from Steve Teixeira and Xavier Pacheco's *Delphi Developer's Guide*; they have kindly agreed to allow us to include the DLL on the disk), give Delphi much of the same simple automation capabilities that VB programmers have been enjoying all along. I've also created `StayOnTop` and `NoStayOnTop` which are put to good use in these custom coders to keep them handy when you need them.

The second unit included is STRFUNC.PAS, which contains a number of useful string manipulation routines, a few of which are used by the coders. I know supplementary string function libraries are common, so if you have one you prefer, you can make the appropriate uses clause and function call changes.



➤ *Figure 1: MethGen in action*

Using MethGen

First, in the source editor, highlight the name of the class you're interested in and click MethGen's `Acquire Class Name` button. This activates the `FindChild` routine from the EXPTTOOL unit to find the current Delphi code edit window, then sends it a message to copy whatever is currently highlighted to the clipboard. MethGen then uses a clipboard object to place the text into the `Class Name` edit box (you could always type it in of course, but I'm lazy...).

Now highlight the method title(s) in your class definition, then click the `Acquire Methods` button, which (similarly to the `Acquire Class Name` button) copies the method titles into the `Methods TMemo`. You can perform this action several times to get methods that aren't declared consecutively. Alternatively, you could copy the entire class definition into the `TMemo` and then just delete the irrelevant parts. One thing I couldn't figure out how to do was to poll the clipboard to find out the size of the text it contains, so in the

MethodsAquireBtnClick method I had to pick some reasonably large number to set the transfer buffer to before retrieving the text. If anybody knows how to find out how much text the Clipboard is holding, I'd like to know about it!

Pressing the Generate Skeletons button takes each method title, inserts the Class Name and a period, appends a begin...end pair, then places the result into the Skeletons TMemo. Here you can make any further alterations you might want.

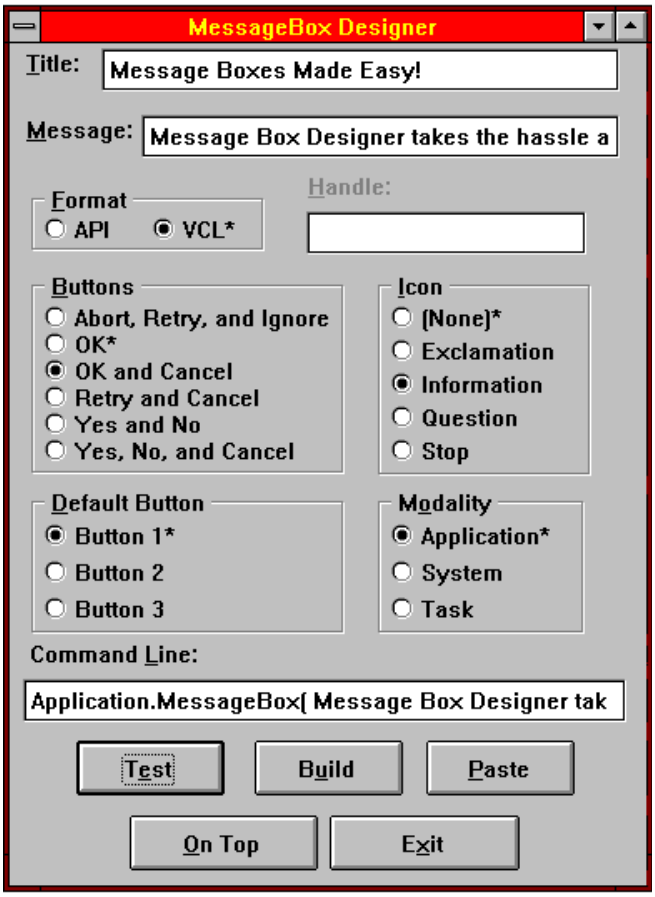
Now move the insertion point in the code editor into the unit's implementation section, then press MethGen's Paste button, which will dump the contents of the Skeletons TMemo at the insertion point.

As you can see from the code on the disk, MethGen is actually a pretty simple program. The interesting parts are probably the two "Aquire" methods and the "Paste" method which are shown in Listing 1 and use the FindChild function from the EXPTTOOL unit to get the HWnd of the Delphi code editor window in order to send it copy and paste commands via SendMessage.

Using MBExpert

Once MBExpert is running, its use should be pretty self-explanatory. Default items are marked with an asterisk. The Test button pops up an example message box to show what the final result will look like (though it doesn't actually do anything of course). The Build button generates the code into the Command Line field, which allows you to review if you wish before pasting it into Delphi's code window. The Format radio buttons determine whether the code generated makes use of the Application.MessageBox (which doesn't require the Handle parameter), or a plain API MessageBox call, which does. Picking the API options enables the Handle field; if nothing is entered, it defaults to HWindow.

One potential area of confusion here is the Title and Message fields. If you're going to use literal strings for these parameters you have to include the appropriate single quotes in the field, otherwise MBExpert won't know to add them and you'll get a syntax error when you compile.



➤ *Figure 2*

➤ *Listing 1*

```
procedure TMethGenFrm.ClassNameAcquireBtnClick(Sender: TObject);
begin
  SendMessage(FindChild('TEditWindow', 'TEditControl'), WM_COPY, 0, 0);
  ClassNameEdit.Text := Clipboard.AsText;
end;
procedure TMethGenFrm.MethodsAquireBtnClick(Sender: TObject);
const BufSize = 2048;
var   TheBuf : PChar;
      TheLen : word;
begin
  SendMessage(FindChild('TEditWindow', 'TEditControl'), WM_COPY, 0, 0);
  GetMem(TheBuf, BufSize);
  Clipboard.GetTextBuf(TheBuf, BufSize);
  MethodsMemo.SetTextBuf(TheBuf);
  FreeMem(TheBuf, BufSize);
end;
procedure TMethGenFrm.PasteBtnClick(Sender: TObject);
var  TheBuf : PChar;
     TheLen : word;
begin
  TheLen := SkeletonsMemo.GetTextLen;
  GetMem(TheBuf, TheLen);
  SkeletonsMemo.GetTextBuf(TheBuf, TheLen);
  Clipboard.SetTextBuf(TheBuf);
  FreeMem(TheBuf, TheLen);
  SendMessage(FindChild('TEditWindow', 'TEditControl'), WM_PASTE, 0, 0);
end;
```

What Now...?

I have some more grandiose plans for these techniques. I'd like to create a more automated method generator and am investigating using Delphi's VCS interface to help implement that.

I also created a very simple macro language for Delphi (and other text based applications) a while back using the ScriptMaker utility that comes with Norton Desktop, mostly because of its FindWindow and SendKeys capabilities. Since I now have those abilities in Delphi, I may redesign the language in Delphi and try to integrate it more fully with the IDE. I'll keep you posted.

Stephen Posey works at the University of New Orleans, USA, and can be contacted by email as SLP@uno.edu