Randolph Grapples With GUIBuilder

By William Baker

As a creator and key programmer of GUIBuilder, William has overseen its development from inception to release.

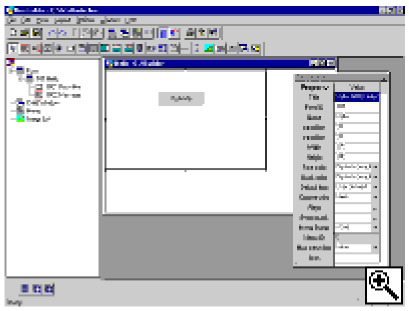
I was preparing to put Randolph to sleep with a three-hour dissertation on managing GUI interfaces with event loops when I was pleasantly surprised to find a new tool in my Early Access 4 download called GUIBuilder. This new tool seemed to take most of the event loop burden off my back and the tedium of hearing about event loops off Randolph's ears.

"Look at this new thing I found in my Early Access download," I said to Randolph.

"Early Access--there's nothing new in that," Randolph replied. "I got Early Access code a year ago at TechCon97 in Nashville. By now you have Late Access--about 11 p.m., I'd say."

"You have Early Access 1--it's definitely too late for that. You should be on Early Access 4 because it's the latest of the early releases," I explained. "Anyway, I was just now putting together a Hello World program. You can see I'm starting with ResBuilder™ to define a simple form with two controls on it: a button and a text control. The form is named Hello, the button is named **Push Me**, and the text control is named **Message**."

Randolph studied the ResBuilder screen for a while and said, "It doesn't look to me like you're writing a Hello World program. It looks to me like you're writing a Push Me program."

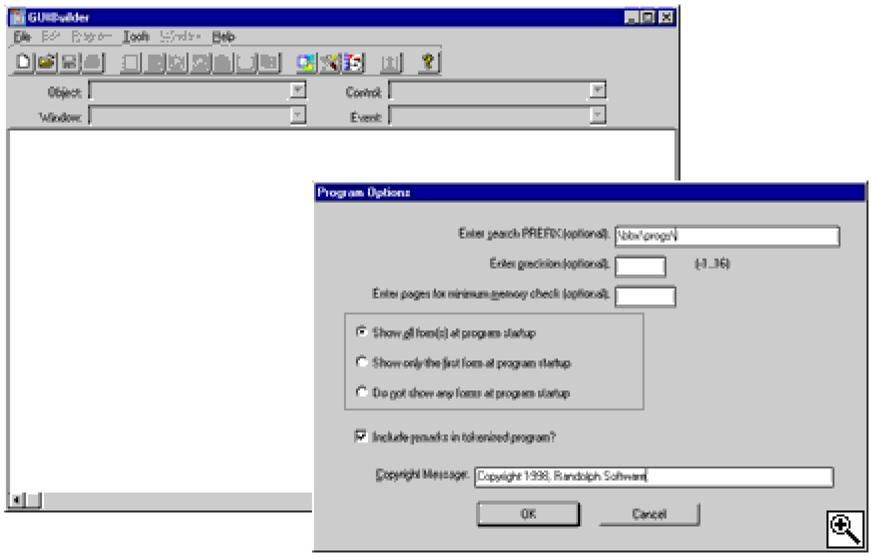

Designing an interface in ResBuilder. The forms are saved in a resource file

"Remember, this is just the interface I'm defining here," I remarked. "The 'hello' part will come later. I have to save this interface in a resource file before we start writing code. I'll call my file **`hello.brc`**.

"Now I'll bring up GUIBuilder, which is what they call this new tool, and click on **New** from the **File** menu. It prompts me for the name of my new GUIBuilder file, which will be the foundation for our final running program. When we're ready, GUIBuilder will use all the information we put into this file to generate and tokenize a working GUI program--complete with event loop. I'll call it **`hello.bbx`**. It asks me if I want to run ResBuilder to create a resource. I've already done that, so I'll click **No**, and now I'm prompted to name the resource file that my program is going to use. I'll enter the name of the resource I just created."

Then a dialog titled **Program Options** came up.

"These options affect how the generated program will work," I explained. "For example, if we want our program to have a PREFIX statement, we would enter paths in the first field. And I'll put in the copyright notice."



Enter search PREFIX (optional): \bbx\progs\

Enter precision (optional): (-1..16)

Enter pages for minimum memory check (optional):

(● Show all form(s) at program startup)
( ) Show only the first form at program startup
( ) Do not show any forms at program startup

[✓] Include remarks in tokenized program?

Copyright Message: Copyright 1998, Randolph Software

Upper left: GUIBuilder main screen before a .gbf file is loaded.
*Lower right: The **Program Options** screen lets you set certain operations of the generated program.*

We then were returned to the GUIBuilder main screen, and I selected the Hello form from the **Select Form** option on the **Program** menu. The form I had just created in ResBuilder appeared, with the **Push Me** button grayed out.

"Now we'll select a control that we want to work with. The easiest place to start is the **close** button, so I'll double-click it. Now we're in the text edit control where we write the code to be executed when the user clicks on the **close** button," I stated. "What do you think the user will want to do when clicking the **close** button?"

"Go home, I would think," Randolph said.

"Well, I meant what should the *program* do? " I asked. "A GUI program's equivalent of going home is to terminate its event loop and do end-of-job tasks. I can end the event loop by setting the loop control variable **`gb__eoj=1`**. (I later discovered GUIBuilder-generated programs will exit the event loop when all forms are closed, without the programmer explicitly setting **`gb__eoj`**.)

Then I brought up my form again and double-clicked on the button. A list box came up with three entries. "This list shows the three events that can be fired by a button," I explained. "The first possible event is that the button is pushed. The other possibilities are that the button gains focus or loses focus. We don't want our program to do anything when focus is gained or lost, but we do want to print our message on the screen when the button is pushed, so I'll select the first event."

The GUIBuilder main screen came back with the text editing area blank. Randolph was alarmed that my **`gb__eoj=1`** statement had disappeared. I assured him there was nothing to worry about, GUIBuilder was keeping track of my various blocks of code.

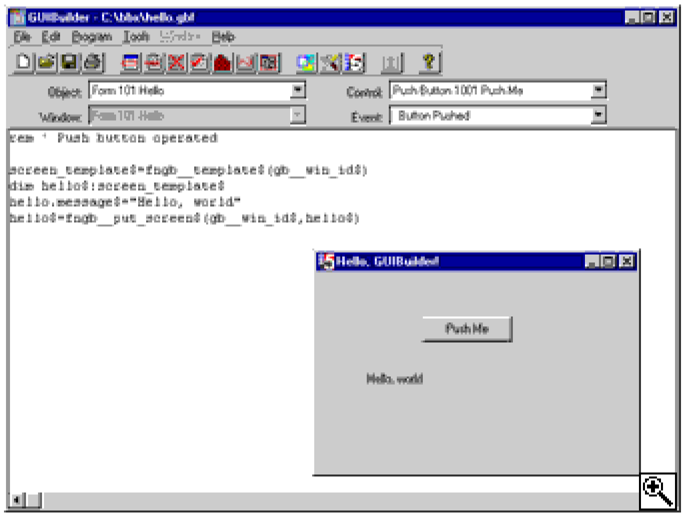
"Blocks of code?" Randolph asked nervously. "I write programs whole, not in blocks."

"Yes, and I suppose you're perfectly happy to type in a whole program at the READY prompt, but this is a more modern way that most people find more efficient. And I suppose if you really want to program the old way, you could write subroutines in GUIBuilder, generate a small program, and then finish writing at the READY prompt," I said. "Anyway, let's get on with the code that will be executed when the button is pushed." Then I typed in four lines that didn't make any sense to Randolph:

```
screen_template$=fngb__template$(gb__win_id$)
dim hello$:screen_template$
hello.message$="Hello, world"
hello$=fngb__put_screen$(gb__win_id$,hello$)
```

I explained that I was using a couple of functions provided with GUIBuilder to display a message on the screen. I could use Visual PRO/5® mnemonics to do this, but the functions encouraged more readable code, once you learned them. The first line invokes the **`fngb__template$()`** function, which returns a string template that describes all the controls on the Hello form. The second line dimensions the **`hello$`** variable with this template. I chose the **`hello$`** variable name to correspond with the form name I assigned in ResBuilder, but there is no requirement that the names correspond. The third line sets the template variable **`hello.message$`** to the value that I want to display, which is "Hello, world." The **`message$`** variable name corresponds to the name of the static text control. Finally, I used another function to transfer the values in the string template to the screen.

Now we were ready to generate and run our program. I selected **Run Program** and entered the name I wanted for my tokenized program file. My five lines of code were merged into several generated lines of code, including event loop logic, and saved in an ASCII source file. Then that source file was compiled into a Visual PRO/5 program. GUIBuilder ran my Hello World program, I pushed the **Push Me** button, and the screen looked like this:



Object: Form 101 Hello
Window: Form 101 Hello
Control: Push Button 1001 Push Me
Event: Button Pushed

```
rem ' Push button operated

screen_template$=fngb__template$(gb__win_id$)
dim hello$:screen_template$
hello.message$="Hello, world"
hello$=fngb__put_screen$(gb__win_id$,hello$)
```

The Hello World program running from the GUIBuilder interface.

Randolph said, "I have to admit getting a whole program out of five lines of code is sort of impressive, but I never did understand string templates, so four of the five lines don't make sense to me. I don't see how it's going to help me."

I replied, "I suppose you could learn how to write event loops from scratch, and you could learn to manage your own context IDs and refer to control IDs by number, and it would work pretty well for a program this simple. But when you get to more complicated interfaces with more fields on them, I think you'll see the advantage of using better tools."

"*When* I get to more complicated interfaces!" Randolph exclaimed. "What makes you so sure that will happen?"

"Just a hunch that your customers are going to demand features that will require the latest Visual PRO/5, so you'll reluctantly start trying to implement them with the least amount of work," I said. "Remember the third law of software development: It's always later than you think."