Moving From Legacy To GUI

By Jim Douglas

Your existing code base is your biggest asset, but it's also your biggest hurdle in moving to a GUI environment. The GUIBuilder™ visual programming environment offers some help by offering cut-and-paste code capability from an existing legacy program to a new GUI program. But if you have lots of data entry programs and would like to put together a batch update procedure, you'll need to step outside of the visual programming environment. If your existing code base and screen layouts are sufficiently well structured, you might be able to partially automate the conversion process. This is possible because the components that make up GUIBuilder are implemented as independently callable procedures. The following is a brief overview of the process:

1. Write a program to read your existing screen formats--from screen files or directly from the program source--and write ASCII resource files. You can read about this process in detail in "ResCompiler: Another Path to the User Interface Dictionary" in *The BASIS Advantage* at *www.basis.com/advantage/mag-v2n1/rescompiler.html*. One way to simplify this process is to create your ASCII resource files with the unit of measure defined as *characters* as opposed to *pixels* or *semichars*. Because character screens don't map directly to GUI screens, you'll also need to make some decisions about how you will convert various screen elements. The easiest way to start is to convert static text to Text controls, string fields to INPUTE controls, numeric fields to INPUTN controls, and boxes and lines to Groupbox and Line controls. For a more complete discussion of this process, see "A New Approach to Going GUI" at *www.basis.com/advantage/mag-v1n3/goinggui.html*.

2. Compile each ASCII resource file to a binary resource file using the ResCompiler program, **`rescomp.exe`**. For example, an ASCII resource file named **`sample.arc`** would compile to a binary resource file named **`sample.brc`**.

3. Generate the framework for a GUIBuilder control file (**`sample.gbf`**) by calling the GUIBuilder module **`gb_func::gb__make_gbf`**.

4. Write a program to analyze your legacy program and extract pieces that can be used in a GUI version of the program. To help with this process, you might find a use for the called program **`_label`**, which converts all line number references in a program to line label references, as well as the **`pro5lst`** utility,

which converts a program file to text. As you extract these pieces, write them to the GUIBuilder control file (`sample.gbf`) using the function `fngb__put_code$()` included in `gb_func.src`. Think in terms of the following functional areas:

- **Initialization.** Open data files, define record templates and other global variables. Note that GUIBuilder generates an ENTER line with a single variable, `gb__arg$`, which can have an associated template to allow for passing multiple values. Initialization code is written to the GUIBuilder control file with a header of `[Init]`.

- **End of Job.** Close data files, clean up global variables. End of Job code is written to the GUIBuilder control file with a header of `[EOJ]`.

- **Event Handlers.** This is the most difficult because this is where the structures of character-mode and GUI programs most differ. As a very rough approximation, consider using the Got Focus and Lost Focus events.

    - The Got Focus event is available for all data entry controls--INPUTE, INPUTN, TEXT, and TXEDIT. It's triggered as the user enters the control.

    - The Lost Focus event is also available for all data entry controls. This event is triggered when the user leaves a control. Don't assume the user is necessarily finished with a control just because you get a Lost Focus event. It could mean that the user just called up the help system to check something.

    - In a GUI program, the most reliable way for users to inform you that they are finished with a given screen is to push an **OK** button. If you have a standard data entry frame, you might want to consider automatically generating a series of buttons to handle standard functions, for example: **Update Record**, **Delete Record**, **Print Record**, etc.

- **Subroutines, Functions, and Nonexecuting Code Blocks.** These can be relatively easy to move into the GUIBuilder control file, but it's up to you to ensure that they're being used properly. Subroutines must start with a label and end with a return statement. Nonexecuting code blocks (IOLIST, DATA, and TABLE) should have an associated label so that you can refer to them from elsewhere in the program. These code blocks are written to the GUIBuilder control file with a header of `[Function (unique_name)]`. If you have standard subroutines, functions, IOLISTs, TABLEs, or other blocks of code that

should be included in all of your programs, put them in the standard include file **gb_std.cod**. This file is merged into the end of all programs generated by GUIBuilder. The standard error and escape handlers are stored in ASCII text files **gb_err.cod** and **gb_esc.cod**. If necessary, you can replace them with customized error and escape handlers.

5. Confirm that the structure of the GUIBuilder control file (**sample.gbf**) works by calling the GUIBuilder module **gb_func::gb__val_data**.

6. Generate the final program (**sample.bbx**) by calling the GUIBuilder module **gb_func::gb__build_program**.

7. At this point, the generated GUI program can be maintained within the GUIBuilder visual programming environment.