

# Using Custom AppBuilder Generator Programs

May 2008

By

Jon Bradley

The purpose of this guide is to explain the new configurability of AppBuilder's program generation process. It covers the use of BASIS provided example generator programs, and general design guidelines for creating a custom generator program.

# Using Custom AppBuilder Generator Programs

## ***Introduction***

AppBuilder, like GUIBuilder, uses a BBJ® program to generate a BBJ program based on the contents of a .gbf file. Previously, this took the form of *gb\_func*. BBJ 8.0 contains a beta introduction of support for user specifiable program generators in AppBuilder. The new generator programs access the .gbf's contents through a friendlier, documented Java object [AdvancedGBFWrapper](#). Subsequent BBJ releases will fine-tune and eventually support the custom program generators fully in 9.0. Additionally, BBJ 8.0 includes two new generator programs – *ProcessEventsBuilder.src* that creates a bare bones PROCESS\_EVENTS-based program and *LegacyPEBuilder.src* that supports some of the more commonly used variables a program generated by *gb\_func* provides.

## ***How to Select a Custom Generator Program in AppBuilder***

Select the generating program via **Tools | Options**. Next, select the **FormAppBuilder** options, and then choose the generating program from the list. Only the BASIS-provided programs appear in the list, so type in the program name if you created your own generator program.

## ***Compile Without Output Support***

Selecting any generator program other than *gb\_func* configures the AppBuilder system to respect the “Compile without output” setting in **BBJ Compiler Settings**. This causes the back end to compile the generated program to check for errors, but it does not create a .bbj file. Note that pressing the [RUN] button for the AppBuilder project runs the .src file if “Compile without output” is enabled.

## ***Design Guidelines for Creating Your own Generator Program***

Modifying the BASIS-supplied generators is strongly discouraged since the subsequent installs will overwrite them. It is better to copy the supplied generators into a new file or start from scratch. There are very few things one **MUST** do when creating a generator program. The following example, which produces a simple “hello world” program, demonstrates the necessary functionality and interface any generator program must contain.

```

10 build_program:
20   enter GBF!

40   let filename$ = GBF!.getProperty("Program Name") + ".src"
50   erase filename$,err=#next
60   string filename$
70   CHAN = unt
80   open(CHAN)filename$
90   print(CHAN)"PRINT ""Hello world""
100  print(CHAN) "escape"
110  close(CHAN)
120  exit

```

Line	Comments
10	The program must contain a label <code>build_program</code>
20	The program must enter an object. This object will be an <a href="#">AdvancedGBFWrapper</a> and the object you will use to get information about the <code>.gbf</code> file.
40-80	The program must create a program file as specified by the <b>Program Name</b> property and it must end in <code>.src</code> . This is the file the system will compile (with or without output) to check for errors. It is also the program that will run if "Compile without output" is selected.
110	Remember to close the channel of the file to which you are writing.
120	It is important not to <b>exit</b> , not <b>release</b> , so the rest of the AppBuilder backend has a chance to compile the results of your generator and report errors.

## The AdvancedGBFWrapper Class

The back-end system passes an instance of the `AdvancedGBFWrapper` class to all generators (other than `gb_func`). It provides easy access to the contents of the `.gbf` file. The `ProcessEventsBuilder.src` is a relatively simple example of usage. Refer to [AdvancedGBFWrapper](#) online.

## Moving Forward with Existing .gbf Files

The BBx® language has evolved over the years to provide natively much of the functionality the `gb_func` generator program supplied via BBx® code. `PROCESS_EVENTS` handles the dispatch of events automatically, removing the need for a `ReadRecord` loop to dispatch events to the correct handler. All BBj events have a corresponding object that contains the relevant information. This allows you to write more readable code and removes the need to use the notify strings.

If your `.gbf` files do not take advantage of any of the code or variables generated by `gb_func` (ie: `gb__event$`) and does not make use of the `.cod` files, then your program *should* build and run using any of the BASIS-provided generator programs. The `LegacyPEBuilder.src` provides support for some of the more commonly used variables: `gb__args$`, `gb__args`, `gb__sysgui$`, `gb__sysgui`, `gb__win$`, `gb__event$`, and `gb__notice$`.