BIOPAC Systems, Inc. Registered to ISO 9001:2008 RESEARCH
42 Aero Camino, Goleta, CA 93117
Tel (805) 685-0066 | Fax (805) 685-0067
info@biopac.com | www.biopac.com
5.03.11

Application Note 264 Flow Control via User Interaction with a Graph File UI

Scripting functionality is available through an optional license available with Acq*Knowledge* 4.1.1 or above. The Scripting license must be authorized to access Script functionality. Scripting is available for Windows 7 or Vista and Mac OS X 10.4-10.6. Rich dialog development is supported for users with Qt SDK 4.5.0 or higher.

The Scripting license enables the Script menu, adds the Calculation channel Preset “Run Macro,” links the **BIOPAC Basic Reference** under Help, and adds Preferences for Scripting.

This application note discusses program flow control that is predicated on a user’s interaction with graphical interfaces, namely, buttons, scroll lists, and dialog boxes.

Flow Control within BIOPAC Basic Scripting (BBS) is described in the manual as follows:

> “Macro code is normally executed in linear order from the first line of a macro function to the last line of a function, one line of code at a time. Flow control operations allow for execution to be changed to a different location. This may be used to create loops, call other functions, and halt the interpreter.”

As such, Flow Control is limited to the following:

• **Goto**	- jump to a label within the script
• **Call** “fnName”	- call to a function routine
• **Runscript** “filename” “fnName”	- loads a new script
• **Halt**	- stop all macro execution
• **Break**	- stop execution of current macro function, returns control to calling macro function
• **Eval** “macro_code”	- executes a single line of macro code
• **Wait**	- suspends macro execution for a specified period of time

Given the present list of commands for flow control, there is no function/command within the BBS nor in the Acq*Knowledge* programming environment that allows the user to suspend execution of the program/script to await user interaction with a graph file, be it: selection of data within a channel, transform invocation, or the activation of UI widgets (buttons or combo boxes). This is particularly tricky when a programmer wishes to establish a loop structure of some sort that branches on user input, for instance, Continue or Cancel?

For loop constructs predicated on user input, the flow control is not constructed using individual command statements but on macro function calls. An example will illustrate.

Figure 1 shows a traditional control loop where the calling function “FindAvg” calls upon “RunEnsemble.” Within RunEnsemble the graph file is modified by the installation of UI buttons (Figure 2). Depending on which button is depressed, the averaging commences or not.

On execution and upon entering the function “RunEnsemble”, one would presume that the program would “wait” until the user has depressed a button to drive the direction of the program; either to “ProcessPeakSelection” or “Cancel.” However, there is no command to force the program to wait, and the next lines of code executed are the increment counter and the Goto command back at the calling function “FindAvg.” Even though the buttons are active and awaiting action, the program flow has left the function “RunEnsemble”. What to do?

Basically a structuring of the loop is required. Figure 3 shows how this is done.

The key idea is that the calls to the macro functions are configured into a control loop. Also key is that those functions that have UI widgets that await user action, are not followed by any executable statements, they are wholly contained within a function block. In addition, all variables used at decision points should be either global or at script level so that all functions are cognizant of the variables’ existence and value.

Function “FindAvg” serves and the entry point into the loop and function “ProcessPeakSelection” serves as the decision point and terminus of the loop. Figure 4 shows in greater detail how this “function-loop” is implemented.

Loops with this type of format will facilitate scripts that desire user interaction with graph files and UI widgets.

Function: FindAvg

```
;Find ensemble average of all files

C# = 1
Loop1:
if C# < T + 1
        Call “RunEnsemble”
        C# = C# + 1
        goto Loop1
endif
End
```

Figure 1 Loop structure with the calling function “FindAvg” calling “RunEnsemble”

Function: RunEnsemble

```
Show Toolbar
RemoveAllButtons
CreateButton “Peak is Selected”, “ProcessPeakSelection”
CreateButton “Cancel”, “GenericCancel”
Prompt “Highlight a single peak then click ‘Peak is Selected’”,”OK,””,””
Select Cursor Tool Selection

End
```

Figure 2 Called function “RunEnsemble”

Restructured Control Loop

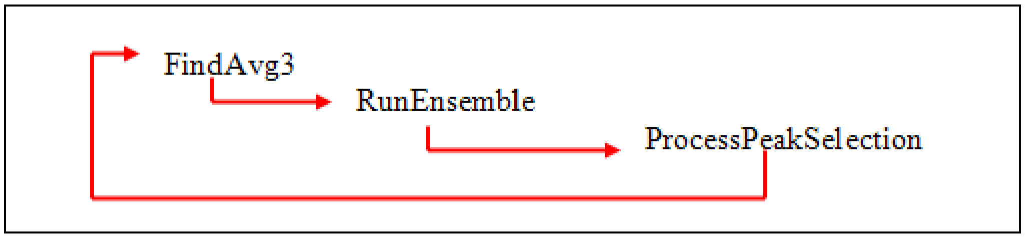

Figure 3 Restructuring of the Control Loop to allow for using interaction

Function: FindAvg

```
;Find ensemble average of all files
ChooseItemArray " Select the file",R#,"SegmentFiles","OK,"Cancel", X#

if X# = 2
        Prompt "No more files.","OK"
endif

if X# = 1
        Call "RunEnsemble"      ← The last executable line of code
endif

End
```

Figure 4a Restructuring the control loop within the calling function

Function: RunEnsemble

```
Show Toolbar
RemoveAllButtons
CreateButton "Peak is Selected", "ProcessPeakSelection"
CreateButton "Cancel", "GenericCancel"
Prompt "Highlight a single peak then click 'Peak is Selected'","OK,"","",""
Select Cursor Tool Selection

End
```

Figure 4b Called function "RunEnsemble"

Function: ProcessPeakSelection

```
; function duties
....
Prompt "Analyze another file?","OK","Cancel","",G#

if G# = 1
        Call "FindAvg"      ← Loop back to the original call function
endif

if G# = 2
        Prompt "All done…","OK"
        Halt
Endif2
End
```

Figure 4c Restructured function to act as a control loop terminus