Chef's Choice - the BUI Menu App

A common paradigm used for displaying options is the familiar menu – entrée choices at a restaurant, program choices to run from the Windows Start Menu, or a listing of tasks or modules to run from a typical Business BASIC program menu. The concept of a list of options is very effective, especially in the computer application world when programs are readily categorized and grouped by function. Users select a program to run from the menu program, which responds by replacing the menu screen with the desired application screen. When the user has finished with the selected program, control returns to the menu screen once again.

Are Menus Still the Best Choice?

A menu-based work flow is practical and ingrained in many of us, so there are good reasons to stick with it, even in a new model like web-based apps running on a mobile device. In contrast, in the traditional web page environment, we click on a link and expect it to open in a new browser window or tab. This is not a bad solution for tangentially-related content like web pages, but it gets cumbersome and confusing when dealing with a single application running in a browser. A smart phone exacerbates the problem as screen space is at a premium and navigating multiple web pages may be laborious and distracting. The goal is to streamline and simplify the user experience, especially when the program is a BUI web app running on a small-screen mobile device. So is it possible to retain the same effective menu paradigm in a BUI web app running on a smartphone? The answer is a resounding "Yes!"

BUI Web Apps are the Main Course

To illustrate the menu concept with a concrete BUI web app example, take a look at the "Phone BUI Menu App" demo. This app installs with the demos along with BBj® so anyone can view the underlying BBx® source code and CSS stylesheet. It is also published live on the Internet so anyone with a browser and Internet connection can run the web app without downloading or installing BBj. **Figure 1** is one example of the BUI Menu running on a smartphone. The code uses a fixed width for the main window and is somewhat flexible with the height based on the available screen real estate, requiring some tweaking for specific mobile devices. This accounts for the smartphone browser hiding or showing certain UI elements depending on how the user launched the app. For a more detailed discussion regarding screen layouts for specific mobile devices, see Let BUI Put Your App in Touch With Your Users on page 40.

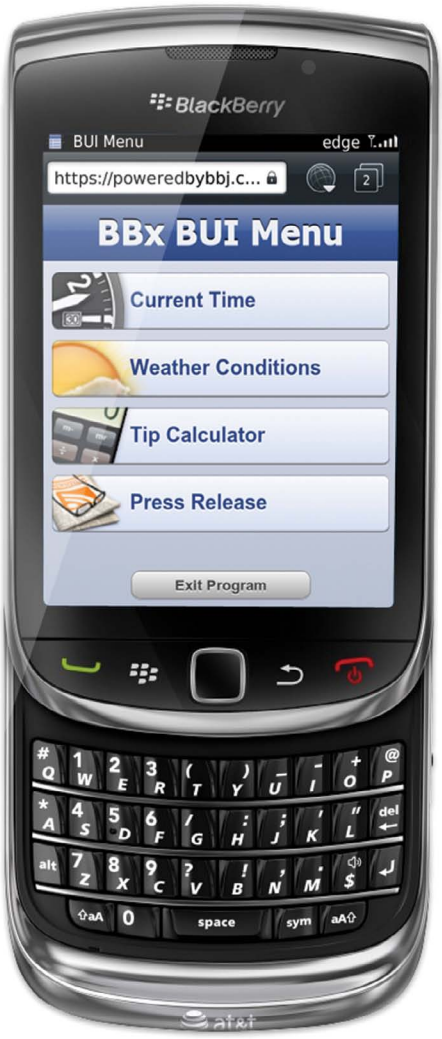

Figure 1. BBx BUI Menu running on a BlackBerry

Ordering From the Menu

To mimic the standard flow of the traditional BBx menu-driven application, the BUI app consists of a few utility programs >>

By Nick Decker
Engineering Supervisor

along with the main menu program. When the demo launches, the menu program displays the list of available utility programs in a series of vertically aligned graphical buttons.

In this simplified case, the user may launch only four programs from the menu – an HTML/CSS analog flip clock, a weather program, a restaurant bill tip calculator, and a BASIS press release viewer. The programs range in complexity from the rather simplistic clock with only a couple of controls, to the more full-featured tip calculator app that makes use of several different BBj controls. Although these are separate programs, the menu program launches each one via the SCALL function. The SCALL's parameter is of the form **`bbj <program_name>`**, meaning that the interpreter takes a shortcut and launches another instance of BBj instead of actually passing the command on to the operating system as it would for a true "System Call". This results in very fast execution of the chosen utility app and is very efficient because the SCALL consumes fewer resources compared to the similar BBjThinClient::clientExec method.

The Benefits of SCALL

Having the menu program SCALL the utility program makes a lot of sense since it happens quickly and has low overhead. But what really matters is that the chosen utility program uses the same display as the menu program. That is also desirable, as it helps to ensure that the transition between programs is as seamless as possible. In contrast, opening the chosen BUI app in another browser session results in the menu app page being pushed into the background while a new browser window opens to display the utility program. That in itself is not so disruptive, but exiting the utility app results in a nearly empty browser page showing a single link to reload the app. To get back to the menu, one would have to click on the browser's session icon on the bottom bar, tap to close the current page, slide the page carousel to the right to select the menu app page, then tap once more to go back to that web page. That whole routine is about as far from seamless as one can get, and requiring that many taps and gestures results in frustrated users and increases the potential for errors and mistakes. It turns out that there is a rather simple solution to this – maintain a single visible window regardless of which program is running.

Maintaining a Single Window

Because SCALLs are either synchronous or asynchronous, developers have a choice in how the menu program launches the utility. The concept of synchronicity – whether the main menu program waits for the SCALLed program to finish or not – has interesting ramifications for the BUI Menu demo. Because it was designed specifically to emulate a native smartphone app, the code does a few things that may be considered a bit unusual to accomplish this feat. In addition to creating the window without a title bar, the demo makes a concerted effort to ensure that only one window is visible at a time. When multiple windows are visible in a BUI program, BBj automatically displays a tabbed bar on top of the app to facilitate the act of switching windows. The tabbed title bar is very useful in normal circumstances, but does not accurately reflect a native smartphone application. Therefore, the demo must first hide the window associated with the menu program before SCALLing the selected utility. That way, the utility program can show its window normally and the tabbed window bar will not display since there is only one window visible. The trick, of course, is having the menu program show its window once again after the launched utility program has exited. If the menu program neglects to show its window, the user ends up staring at a blank screen.

Passing Control to the Utility

There are a couple of ways to ensure that only one window displays at any given time, based on which version of SCALL is employed. If the menu program uses a synchronous SCALL, execution of that program will halt until the launched utility program has exited. Therefore, the menu program hides its window, SCALLs the utility, and immediately shows its window once more. The synchronous nature of the program flow guarantees that only one window is visible at a time. However, if the menu program uses an asynchronous SCALL, it will have no way of knowing when the utility program has exited and therefore will not know when to show its window.

Although there are many ways of having the menu program and utility program communicate, such as via a socket, namespace variable, etc., it would require extra programming and would make it more difficult to add new utility programs to the menu. The extra overhead and complexity are certainly not desirable, but luckily a simpler alternative exists. The solution is to have the menu program hide its window, SCALL the utility program asynchronously, then RELEASE the BBj session. In order for this scheme to work, the utility programs must follow the same paradigm and hide their window, SCALL the main menu program, then RELEASE. This ensures that a program is running at all times and guarantees a single visible window. While the first solution is easier to understand and implement, choosing the best solution depends on the existing design and structure of your current menu system.

Summary

The BUI Menu demo is a good example of implementing a menu system for a mobile device, and illustrates a few of the special requirements and design decisions that are an important part of any mobile application development. This example is inspiring, as it shows what is possible when you take a traditional BBx menu-driven system and deploy it live on the Internet to browsers and smartphones EVERYwhere. ■

For more information
- Try the BUI Menu demo
- Read *Let BUI Put Your App in Touch With Your Users* on page 40