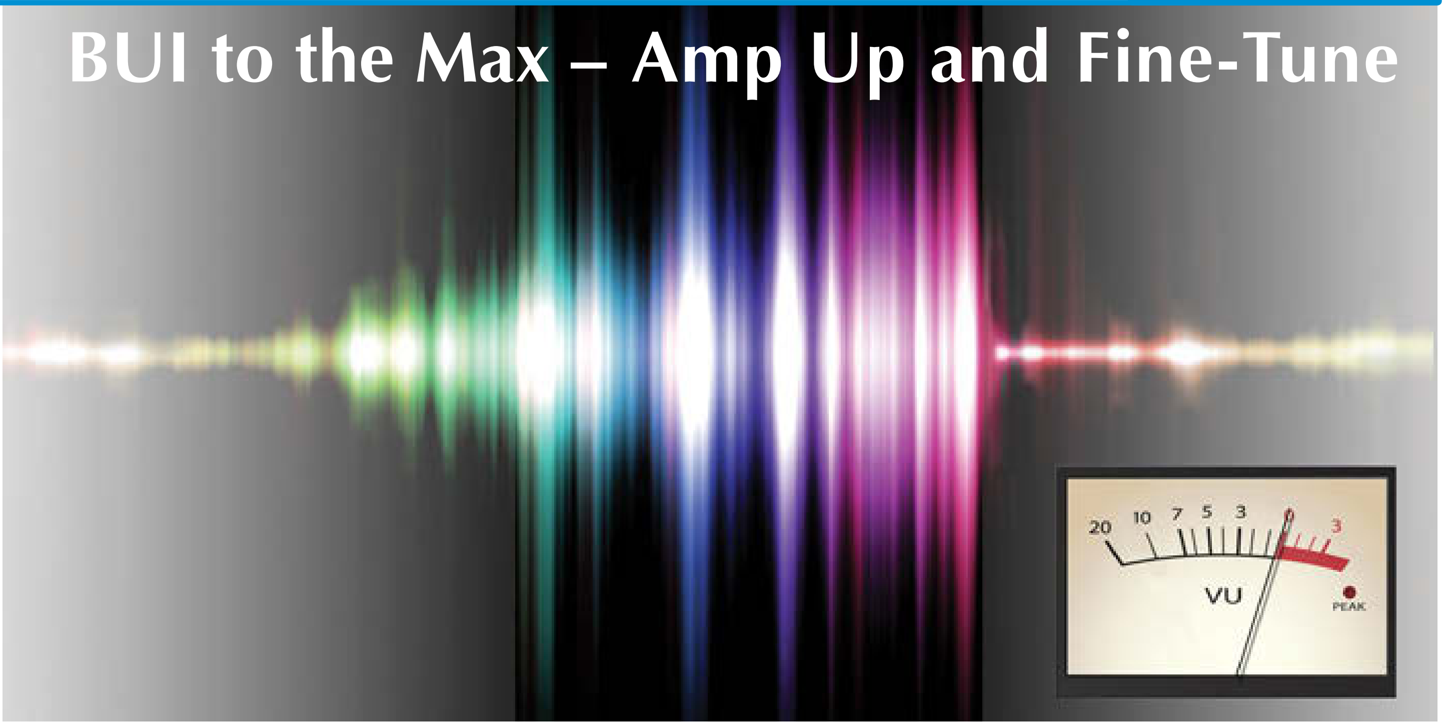
BUI to the Max – Amp Up and Fine-Tune



Most Visual PRO/5® or BBj® GUI applications are inherently BUI applications. In the minute it takes to configure an app in your BUI app server, you're good to go. But a working BUI app is just the first step. This article covers some easy things you can do to maximize the performance and fine-tune the appearance to make it more like apps that users expect of a web page and less like a transplanted desktop application.

Getting Started

BASIS has already taken the first step: BBj 13.10 previews a completely redesigned BUI theme for BBj 14. This new theme includes hundreds of aesthetic and functional tweaks to all BUI controls. On touch devices, dialogues like MSGBOX() and FILEOPEN() are better centered, and some controls have been modified to work better. For example, spinner buttons are oriented horizontally to provide larger touch targets. Even if you do nothing at all, your BUI apps already look better than ever before! Find out more details about the new defaults in *Default CSS Gets a Makeover* at links.basis.com/13css.

By Jim Douglas
Software Developer

Performance

We first talked about bandwidth and latency issues with distributed applications in *The Lessons of BASIS b-Commerce* (links.basis.com/00bcomm). Now, 13 years later, these issues are still a major consideration with client/server applications. We touched on this subject in *The Anatomy of a Web App Makeover* (links.basis.com/12webapp), which looks closely at the BUI version of the BBj download page. When we design a distributed application, the client and server work together to manipulate data, perform calculations, and present information to the user. The communication between the client and server can be broken down into three broad categories.

Category 1: Server to Client – includes methods like `BBjEditBox::setText`. The BBj application running on the server sends data to the client without waiting for a response. BBj automatically optimizes batches of operations in this category to improve performance.

Category 2: Client to Server – includes event traffic like the ON_BUTTON_PUSH event. For the most part, the client sends events to the server without waiting for any sort of response. The event object typically contains additional parameters relating to the event such as, for example, the width/height of a resized window, or the x,y location of a moved window. Because this information is delivered to the server as part of the act of delivering the event, it is immediately available to the program as soon as the event is received.

Category 3: Server to Client to Server (round trip) – includes methods in which the application queries the client for some dynamic information that cannot be cached on the server. Methods like `BBjEditBox::getText` and `BBjCheckBox::isSelected` fall into this category. When a BBj application executes a line like `name$ = editbox!.getText()`, it comes to a complete standstill while the server sends that request to the client, then waits for the client to send back the response. As a rough rule of thumb, we usually assume that each round trip like this takes at least 100 ms (1/10th of a second), but it can easily be double that time, especially on a mobile device. And it gets worse: If the application sends several messages to the client that don't require a response (Category 1), followed by a single message that requires a response (Category 3), the application must wait for the client to process all pending messages, then respond to the final message that requires a response. In some cases, this can add hundreds of milliseconds to the delay. If that round trip hadn't been introduced, the client would have been able to continue working through processing those backlogged messages while the application running on the server moved on to new work.

We can significantly improve application responsiveness by avoiding category 3 (synchronous round trips) as much as possible. For example, we can change the ON_LIST_CLICK event handler for a BBjListBox from this:

```
on_list_click:
    event! = bbjapi().getSysGui().getLastEvent()
    listbox! = event!.getControl()
    index = listbox!.getSelectedIndex(); rem ' goes to the client
    rem -- do something here ---
return
```

to this:

```
on_list_click:
    event! = bbjapi().getSysGui().getLastEvent()
    index = event!.getSelectedIndex(); rem ' gets value from event
    rem -- do something here ---
return
```

In the first version, BBjListBox::getSelectedIndex forces a round trip back to the client, typically introducing a delay of 100 milliseconds or more. In the second version, BBjListClickEvent::getSelectedIndex retrieves the value that was delivered to the server as part of the original event.

To see the effect of eliminating round trips, run the BBjFormValidation demo (**Figure 1**) in the BBx BUI Showcase at links.basis.com/buidemos.

The program shows 14 controls of various types, with a navigator to browse through 100 records containing randomly generated data. The UI is smooth and fast; clicking the navigator buttons brings up records as fast as you can click. The two big buttons at the bottom of the form implement two different ways to retrieve the current contents of the form. The "Form Validation" button fires a BBjFormValidationEvent, which captures the current values of all user-changeable controls on the window. The "getText" button fires a BBjButtonPushEvent. In response to that event, the application queries each of the 14 controls for their current value. The difference in speed is dramatic.

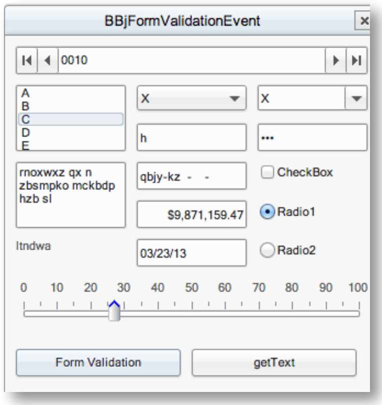
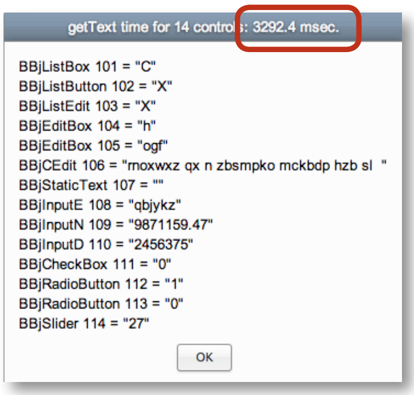


Figure 1. The BBjFormValidation demo

The getText version takes at least 1.5 to 2 seconds, but can take several seconds depending on the state of your Internet connection and the latency to the server machine. When we ran the demo here at BASIS, the getText version took more than 3 seconds, as shown in the title bar of the resulting dialog shown in **Figure 2**.



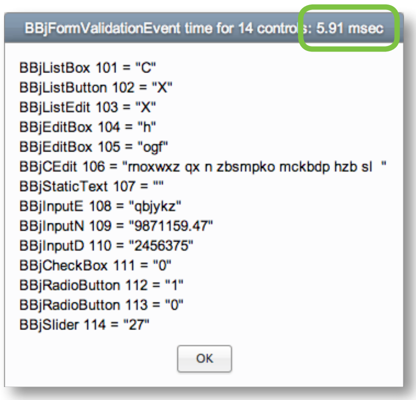
```
getText time for 14 controls: 3292.4 msec.

BBjListBox 101 = "C"
BBjListButton 102 = "X"
BBjListEdit 103 = "X"
BBjEditBox 104 = "h"
BBjEditBox 105 = "ogf"
BBjCEdit 106 = "rnoxwxz qx n zbsmpko mckbdp hzb sl  "
BBjStaticText 107 = ""
BBjInputE 108 = "qbjykz"
BBjInputN 109 = "9871159.47"
BBjInputD 110 = "2456375"
BBjCheckBox 111 = "0"
BBjRadioButton 112 = "1"
BBjRadioButton 113 = "0"
BBjSlider 114 = "27"

OK
```

Figure 2. The getText results taking 3,292.4 milliseconds

The Form Validation version (**Figure 3**) takes effectively no time. It has all the data it needs as soon as the event hits the server, so it reports all of the same information 557 times faster, in a scant 5.91 milliseconds (0.00591 seconds).



```
BBjFormValidationEvent time for 14 controls: 5.91 msec

BBjListBox 101 = "C"
BBjListButton 102 = "X"
BBjListEdit 103 = "X"
BBjEditBox 104 = "h"
BBjEditBox 105 = "ogf"
BBjCEdit 106 = "rnoxwxz qx n zbsmpko mckbdp hzb sl  "
BBjStaticText 107 = ""
BBjInputE 108 = "qbjykz"
BBjInputN 109 = "9871159.47"
BBjInputD 110 = "2456375"
BBjCheckBox 111 = "0"
BBjRadioButton 112 = "1"
BBjRadioButton 113 = "0"
BBjSlider 114 = "27"

OK
```

Figure 3. The Form Validation results taking only 5.91 milliseconds (557 times faster)

Touch Click

If you skim the source code (links.basis.com/formvalidation-code) for that form validation sample program, you might wonder about this strange line:

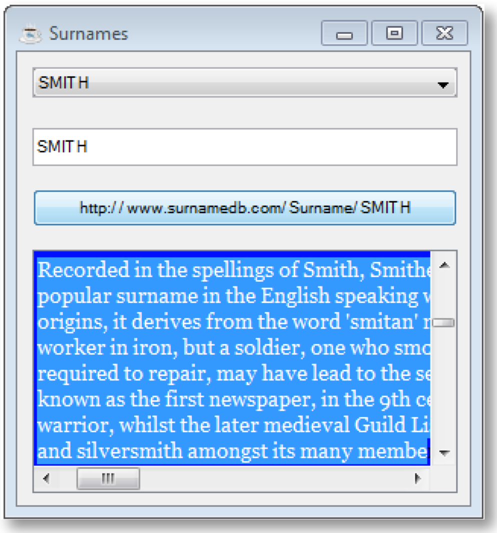
```
fast_touch_click$ = stbl("!OPTIONS","FAST_TOUCH_CLICK=TRUE")
```

In touch-oriented browsers, the user can double-tap anywhere on the screen to zoom in and out. To allow for the possibility that any given tap might be the start of a double-tap gesture, mobile browsers wait for about 300 milliseconds (0.3 seconds) before reporting that a button was clicked. Most of the time, this isn't a problem. The slight delay is a tradeoff for the added usability of being able to double-tap anywhere to zoom. But there are times when you want the user to be able to rapidly click a button and have the program respond to the click immediately.

In the default mobile browser configuration, clicking a button twice within less than 300 ms isn't interpreted as two clicks; it's interpreted as a double-tap gesture to zoom the window. When specifying the FAST_TOUCH_CLICK option, button controls (and the buttons in navigator controls) report click events immediately, eliminating that 300 ms delay. (Of course, this has the effect of disabling the standard double-tap-to-zoom behavior when the user taps directly on a button, which is why it's a developer-configurable option, as opposed to standard BUI behavior.) When selecting this option, the user is still able to double-tap to zoom anywhere else on the form, just not directly on a button.

From GUI to BUI

Let's take a small GUI app, run it in BUI, and see what we might want to do to optimize it for the browser environment. For demonstration purposes, we'll use a trivially simple application, one that prompts for a last name, or surname, and displays information about it. **Figure 4** is the original version running in GUI.

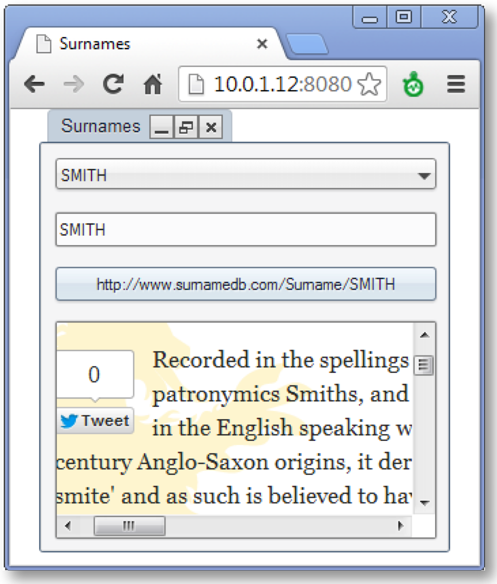


Recorded in the spellings of Smith, Smithe
popular surname in the English speaking w
origins, it derives from the word 'smitan' m
worker in iron, but a soldier, one who smo
required to repair, may have lead to the se
known as the first newspaper, in the 9th ce
warrior, whilst the later medieval Guild Li
and silversmith amongst its many membe

Figure 4. Surname sample running in GUI

Pretty basic stuff – a BBjWindow with a BBjListButton to select a surname from a list, a BBjEditBox to type a name that might not appear on that list, a BBjButton that the user can click to query the currently selected name (identified as a URL on the button), and a BBjHtmlView to show information about that name.

Figure 5 shows the same program running in BUI.



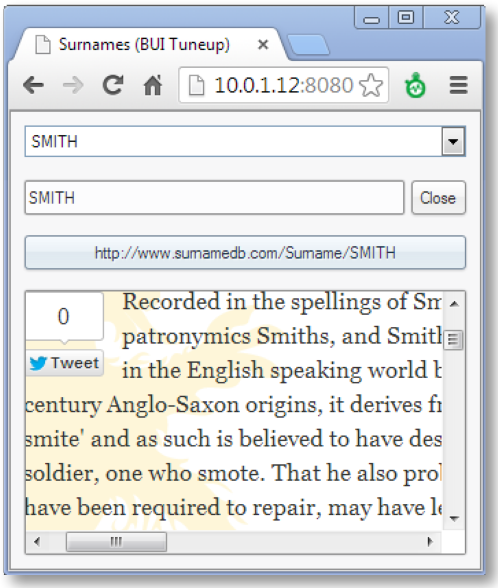
Recorded in the spellings
patronymics Smiths, and
in the English speaking w
century Anglo-Saxon origins, it der
smite' and as such is believed to ha

Figure 5. Surname sample running in BUI (Chrome)

That title bar looks a bit out of place in a web page, and it would be nice if we used all of the available space. So let's try a few tweaks:

- ☑ Create the window with no title bar; optionally add a separate Close button.
- ☑ Maximize the window to use the full browser client area.
- ☑ Use the NATIVE_BROWSER_LIST version of the listbutton; it can provide a better user experience on mobile devices.
- ☑ Size and position all of the controls based on the available space.
- ☑ Add a resize handler to resize the controls when the user resizes the browser (or changes the orientation of a mobile browser).

Figure 6 looks much more like a standard web page, and less like a desktop application running in a browser.

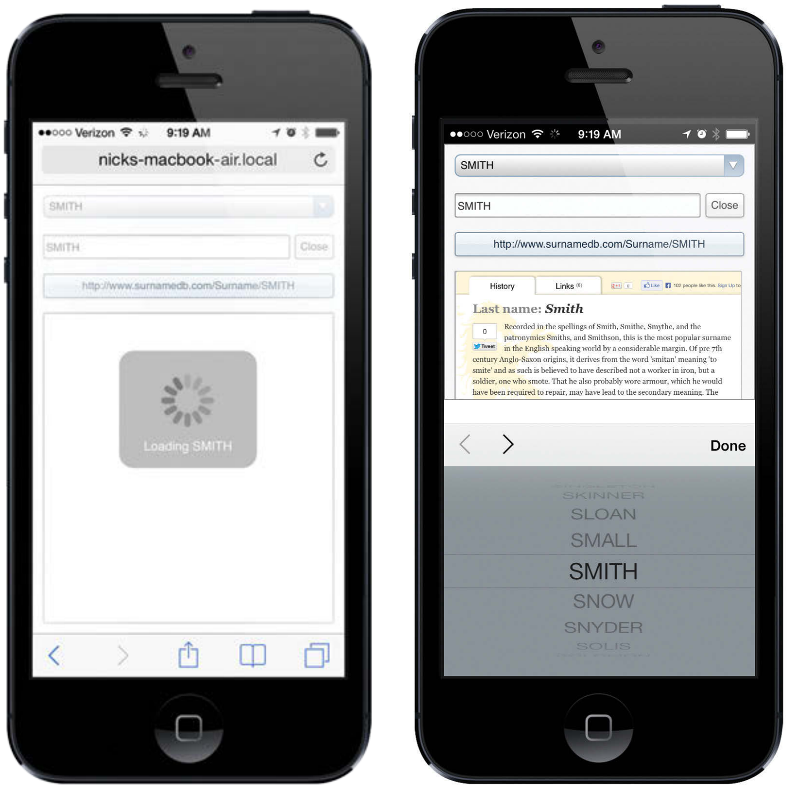


Recorded in the spellings of Sm
patronymics Smiths, and Smith
in the English speaking world b
century Anglo-Saxon origins, it derives fr
smite' and as such is believed to have des
soldier, one who smote. That he also pro
have been required to repair, may have le

Figure 6. Surname sample in BUI with some simple modifications

Now let's try it on a mobile device. This shows us one more opportunity for improvement. In desktop applications (both GUI and BUI), we can set the wait cursor to indicate that the application is temporarily busy. Mobile devices don't have cursors, so BUI offers another mechanism, the BBjBusyIndicator, to indicate that the application is busy. We can show the busy indicator when we set the wait cursor, then remove it when we reset the cursor.

The iPhone on the left in **Figure 7** shows how the busy indicator looks on a mobile device. Notice that the browser's top URL Bar and bottom Button Bar consume an appreciable amount of the already-limited screen real estate. On most mobile devices, you can add the application to the home screen with a few taps, which results in making the BUI app look and feel similar to a native application. The BUI app will then have its own icon on the home screen, will run in "standalone" mode instead of in the browser, and will take up the entire screen space except for the top



Recorded in the spellings of Smith, Smithe, Smythe, and the patronymics Smiths, and Smithson, this is the most popular surname in the English speaking world by a considerable margin. Of pre 7th century Anglo-Saxon origins, it derives from the word 'smitan' meaning 'to smite' and as such is believed to have described not a worker in iron, but a soldier, one who smote. That he also probably wore armour, which he would have been required to repair, may have lead to the secondary meaning. The

Figure 7. Surname sample on a mobile device in the browser (left) and in standalone mode (right)

status bar. The iPhone on the right in **Figure 7** shows the Surname sample running in standalone mode from the home screen. Because the NATIVE_BROWSER_LIST option was set, you choose a surname from the BBjListButton by selecting a name from the native iOS picker control in place of the BBjListButton's dropdown list.

The constrained viewport on mobile devices presents another challenge compared to desktop devices. We should take this into account and limit our display to just a few important items when designing BUI apps for deployment to mobile devices.

We can also take advantage of another new BUI feature: custom end actions. Custom end actions allow the developer or system administrator to define what happens when the user exits a BUI app.

```
click:
   htmlview!.setUrl(url$+name$)
   window!.setCursor(3)
   if busy!<> null() then
      busy!.setText("Loading " + name$)
      busy!.setVisible(1)
      bui!.setEndAction(bui!.urlAction(url$+name$))
   endif
return
```

Figure 8. Sample code to set the custom BUI end action at runtime

When the program loads the selected surname, it can tell BUI to chain to the URL for that name upon termination of the BUI app in **Figure 8.** This code lets the user load a preview of the name information into the htmlview on the BUI app page, then go to a full-page view of that same information when closing the BUI app page. This feature has many potential uses, including building full application menuing systems in BUI.

Summary

BBj's browser user interface has been around for several years. Over time it has matured, and now runs faster, looks better, and provides the application developer with several new configuration and customization options. The addition of form-level validation and more payload data in various events significantly reduces client-server round trips, speeding up the execution of remotely-deployed BUI apps and making them feel more like local applications. BASIS has also added several improvements to streamline and improve BUI apps running on mobile platforms, including native message boxes and list buttons, fast touch click detection, a built-in BBjBusyIndicator, and the ability to easily create a fullscreen app without a titlebar. Lastly, custom end actions give developers the power to determine where their BUI app should take the user after it has finished executing. BASIS' browser user interface is an exciting technology that continues to expand and improve. If you've not begun to take advantage of it yet, now is the time for you to amp up and fine-tune your BUI application. ■ links.basis.com/**13code**

- Refer to *Default CSS Gets a Makeover* at links.basis.com/13css
- Read about the BBjBusyIndicator and custom end actions in *Automate BUI Deployment With the API* at links.basis.com/13buiapi
- See more mobile-optimized sample applications in the entire *BBj BUI Showcase* at links.basis.com/buidemos
- How to add a BUI app to your mobile device's home screen:
  - iOS: links.basis.com/iphone-addicons
  - Android: links.basis.com/android-addbookmarks