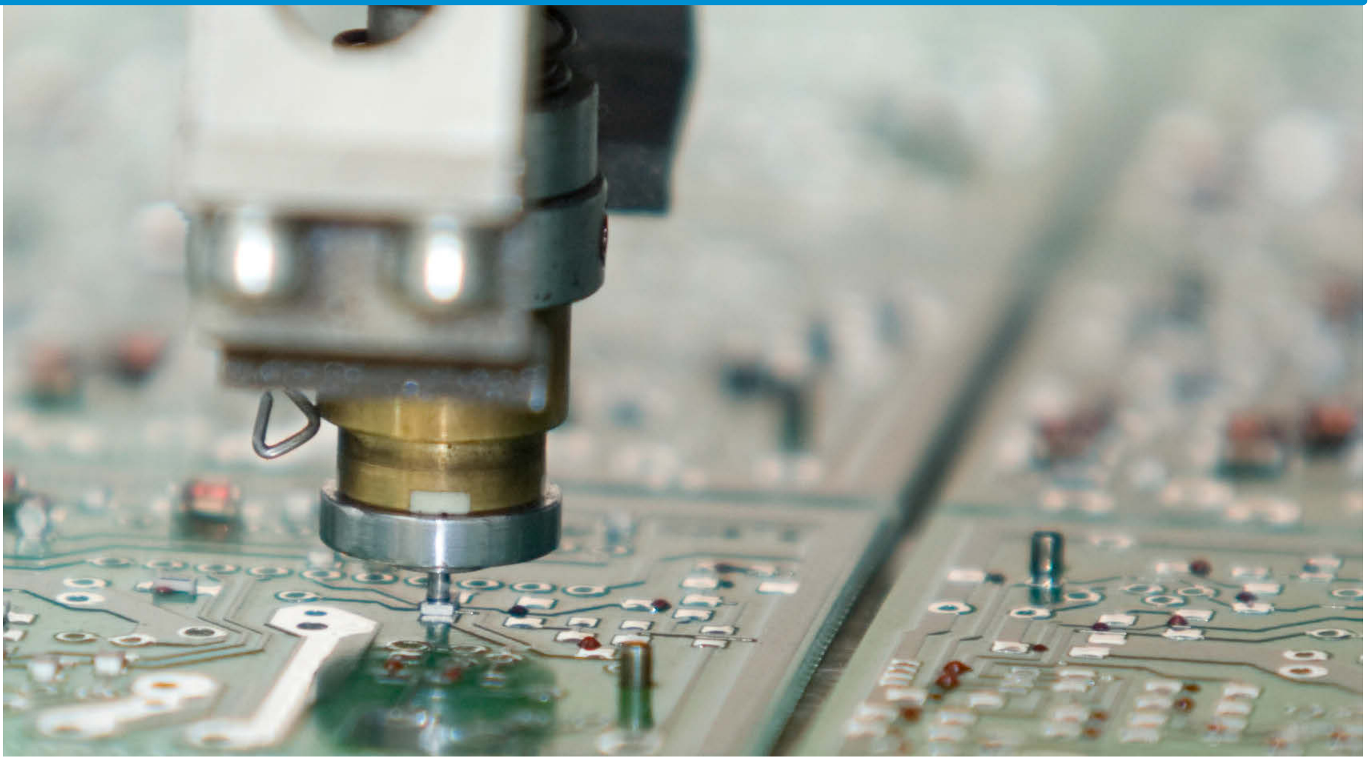


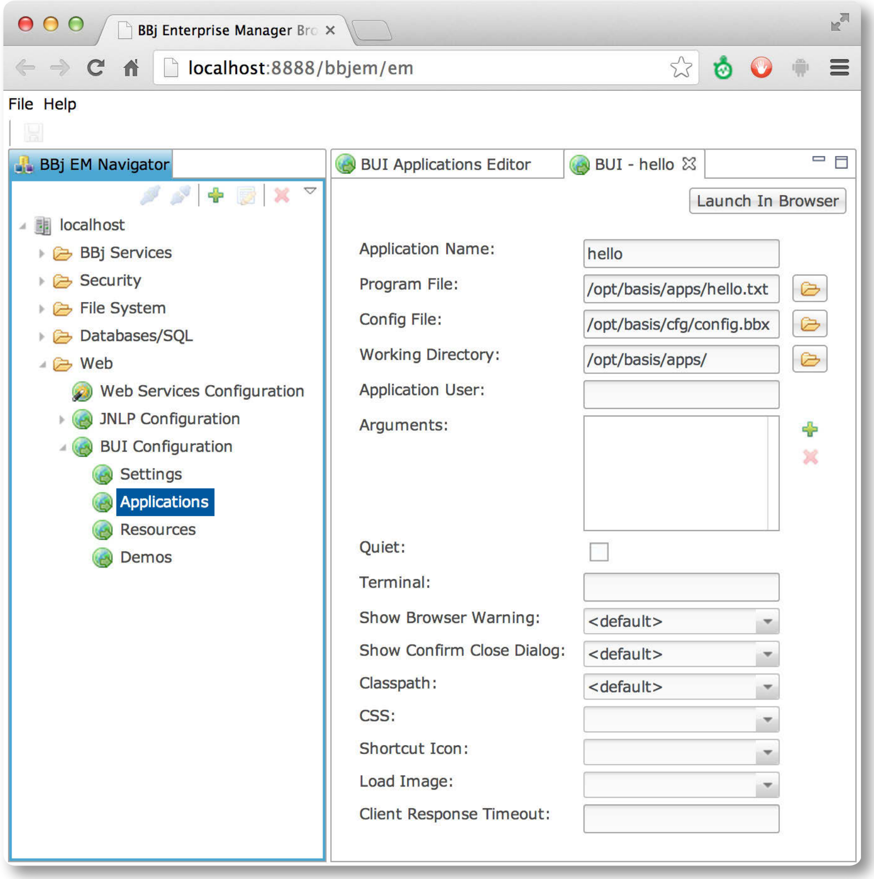
# Automate BUI Deployment With the API

Getting your BBj® app running as a BUI web app is a simple matter of spending a few minutes in Enterprise Manager to register the application, define runtime parameters, and (optionally) define a custom CSS file and browser or home screen icon. As of BBj 14.0, BUI programs come with even more optional configuration properties, including load images (a super-fast replacement for your traditional splash screen), and custom end actions that define what the browser does when the BUI app releases. As more configuration parameters become available, developers are expressing the desire to automate this process by programmatically registering and configuring BUI apps. This article looks at how the updated BUI API handles this with ease, allowing developers to register, modify, and manage every aspect of their web-enabled BUI apps.

*By Jim Douglas*
*Software Developer*

## Getting Started With BUI

The easiest way to get started with BUI is to manually configure a BUI App in Enterprise Manager. See the example in **Figure 1**.



| | |
|---|---|
| Application Name: | hello |
| Program File: | /opt/basis/apps/hello.txt |
| Config File: | /opt/basis/cfg/config.bbx |
| Working Directory: | /opt/basis/apps/ |
| Application User: | |
| Arguments: | |
| Quiet: | ☐ |
| Terminal: | |
| Show Browser Warning: | <default> |
| Show Confirm Close Dialog: | <default> |
| Classpath: | <default> |
| CSS: | |
| Shortcut Icon: | |
| Load Image: | |
| Client Response Timeout: | |

**Figure 1.** A sample BUI App definition in Enterprise Manager

The Enterprise Manager interface is perfect for getting started, and for casually defining a few BUI apps from time to time. However, BUI also has an extensive API for publishing and managing apps. Using the API, you can build automated systems to publish selected apps to the BUI Web Server and update them as needed with new code, CSS, images, and configuration changes. This isn't new; the *BBj BUI: Getting Started* (links.basis.com/buiguide) document has always included a small "Hello World" app (**Figure 2**), along with a sample program that publishes it to the BUI Web Server using the BUI API (**Figure 3**). With the exception of a few small samples like this one, the API documentation was sparse. However, extensive documentation, including full samples, is now available at links.basis.com/bbjappserver and, new for BBj 14.0, links.basis.com/bbjbuimanager.

## BUI Development

The BUI API becomes very handy during the development cycle. When you associate image and CSS resources with a BUI app, the app server takes a snapshot of those resources from the moment when they are added. It is very likely that you'll need to adjust any custom CSS associated with a BUI app several times before you are happy with it. When you change your CSS file, you need to inform the app server about the change. That can be done through Enterprise Manager, but it can become tedious when you are iteratively testing CSS tweaks. The easiest approach is to write a small update program that you run every time you edit your CSS file or change a registered image, refreshing the app server before retesting your app.

```
rem ' hello.txt
sysgui = unt
open (sysgui)"X0"
bbjapi! = bbjapi()
sysgui! = bbjapi!.getSysGui()
window! = sysgui!.addWindow(100,100,225,75,"Hello",$00090003$,$$)
window!.setCallback(bbjapi!.ON_CLOSE,"EOJ")
hello! = window!.addButton(1,25,25,75,25,"Hello!")
hello!.focus()
hello!.setCallback(bbjapi!.ON_BUTTON_PUSH,"msgbox")
goodbye! = window!.addButton(2,125,25,75,25,"Goodbye!")
goodbye!.setCallback(bbjapi!.ON_BUTTON_PUSH,"eoj")
process_events
eoj:
release
msgbox:
  i = msgbox(info(1,4),64,fnmode$(info(3,6)))
  hello!.focus()
return
def fnmode$(mode$)
  if mode$="0" then return "Fat Client"
  if mode$="1" then return "Thin Client"
  if mode$="2" then return "Java Applet"
  if mode$="3" then return "Java Web Start"
  if mode$="4" then return "JavaBBjBridge"
  if mode$="5" then return "BUI (Browser)"
  return mode$
fnend
```

**Figure 2.** A small "Hello World" BUI app

```
rem ' publish.txt
rem ' This assumes that the sample is on the desktop; adjust as necessary
path$ = ""
path$ = env("HOME",err=*next) + "/Desktop/"
if path$="" then path$ = env("HOMEDRIVE") + env("HOMEPATH") + "/Desktop/"
app$ = "hello"
source$ = "hello.txt"
bbjHome! = System.getProperty("basis.BBjHome")
config$ = bbjhome! + "/cfg/config.bbx"
appServer! = bbjapi().getAdmin("admin","admin123").getWebAppServer()
appConfig! = appServer!.makeEmptyAppConfig()
appConfig!.setProgramName(path$ + source$)
appConfig!.setConfigFile(config$)
appConfig!.setWorkingDirectory(path$)
appConfig!.setInterpreterUser(System.getProperty("user.name"))
app! = appConfig!.buildApplication(app$)
appServer!.unpublish(app$,err=*next)
appServer!.publish(app!)
```

**Figure 3.** A short BUI API program that publishes the sample "Hello World" app

**Figure 4** shows an example of this type of program.

```
rem ' update.txt
app$ = "sample"
dir$ = dsk("") + dir("")
css$ = dir$ + app$ + ".css"
img$ = dir$ + app$ + ".png"

appServer! = bbjapi().getAdmin("admin", "admin123").getWebAppServer()
app! = appServer!.getApplication(app$)
appConfig! = app!.toAppConfig()

url! = fnStaticResource!(css$,"text/css")
appConfig!.setStyleSheet(url!)

url! = fnStaticResource!(img$,"image/png")
appConfig!.setLoadImage(url!)

app! = appConfig!.buildApplication(app$)
appServer!.unpublish(app$,err=*next)
appServer!.publish(app!)
print "Updated app ",app$
stop

def fnStaticResource!(filename$,mimetype$)
  iterator! = appServer!.getStaticResources().iterator()
  while iterator!.hasNext()
    resource! = iterator!.next()
    if resource!.getSourceFileName()<>filename$ then continue
    if resource!.getMimeType()<>mimetype$ then continue
    resource!.setSourceFileName(filename$)
    return resource!
  wend
  return appServer!.addStaticResource(filename$,mimetype$)
fnend
```

**Figure 4.** An update program that refreshes a BUI app definition via the API

## New Features

In addition to adding full documentation for the existing BUI API, BBj 14.0 also adds several new features, most of which are accessible through both the interactive Enterprise Manager interface and the programmatic API.

## Load Image

The default load image still displays this animated blue and white progress bar,

but you can now define a customized default load image, and custom load images for individual apps. The load image displays almost instantly as the page loads, instead of waiting until after the interpreter associated with the BUI app is up and running. Because it displays so quickly, it is a perfect way to present the user with an application's splash screen or any other customized 'Loading' indicator.

## End Action

We've had many requests for developer-defined application termination actions in place of the default of just replacing the application in the browser with a "Click to reload application" message. As of BBj 14.0, custom end actions are definable, both at the global default level and for individual apps.

There are several kinds of end actions. You can continue to clear the browser window (the original and still default end action), but show a custom message in place of the default message. Or you can chain to another BUI app, perhaps one that acts as a main menu, or chain to a particular URL. For more flexibility, BBjBuiManager even allows for changing the end action dynamically in a running BUI app. In all cases, the end action takes effect when the app terminates, typically by executing a RELEASE statement.

## Busy Indicator

Traditional desktop applications typically indicate that they are busy by setting the wait cursor. While this still works with BUI apps in desktop browsers, mobile browsers don't have cursors so they need some other way to let the user know that they are busy. The most common way to indicate that a browser-based application is temporarily busy is to dim the screen and display an animation with a short message to let the user know what is happening. Developers can now easily accomplish this with the new Busy Indicator, as the code in **Figure 6** demonstrates.

```
busy! = bbjapi().getBuiManager().getBusyIndicator()
busy!.setText("Busy...")
busy!.setVisible(1)
rem -- do a bunch of work --
busy!.setVisible(0)
```

**Figure 5.** Code to create, customize, and display a BUI busy indicator

When the code sets the busy! object **visible**, the user sees the animated busy indicator shown here to the right.

## Summary

In the past, publishing and managing your BUI apps required a great deal of interaction with Enterprise Manager's BUI Configuration screens. To streamline and help automate this process, BBj now offers a robust API that can do everything Enterprise Manager does – from first-time registration of a BUI app to modifying every available configuration parameter. In addition, developers can implement new features such as the customizable load image, dynamic end action, and the busy indicator interactively or programmatically. The API can save you a lot of time and gives your mousing hand a much-needed rest. Give it a try today! ■

links.basis.com/**13code**

- Read *BBj BUI: Getting Started* at links.basis.com/buiguide
- For more on the BBjBusyIndicator and custom end actions, see *BUI to the Max – Amp Up and Fine-Tune* at links.basis.com/13bui