

# UI Customization

---

**This document is confidential and may not be distributed without the permission of Be Inc.**

---

copyright © 2000 Be Incorporated

# Customizing Bela

## Table of Contents

<b>Introduction</b>	<b>9</b>
<b>1 Programming Languages and Formats</b>	9
<b>2 Sample Configuration</b>	10
<b>3 Directory Structure</b>	10
3.1 Sharing Files Between Configurations	10
<b>4 Configuration Variables</b>	10
4.1 The LANGUAGE Variable	10
4.2 The RESOURCES Variable	10
<b>5 Localization</b>	11
<b>6 The Entry Point Files</b>	11
<b>7 The Content Area</b>	11
7.1 Creating the Content Frame	12
7.2 Special Treatment of the Content Frame	12
7.2.1 References to the Top Frame from within HTML	12
7.2.2 References to the Top Frame from within JavaScript	12
<b>8 Alerts</b>	12
<b>Design Considerations</b>	<b>13</b>
<b>1 The Audience</b>	13
<b>2 The Browser</b>	13
<b>3 Controls</b>	13
3.1 Labels	14
<b>4 Panes</b>	14
<b>5 Feedback and Alerts</b>	15
<b>6 Settings panels</b>	15
<b>7 Technical Considerations</b>	16
<b>8 Recommended Reading</b>	16

**JavaScript Layout Concepts . . . . . 17**

<b>1 Prerequisites</b>	17
<b>2 Fundamental Concepts</b>	17
2.1 Abstract the Creation of HTML Markup into HTML-Generating JavaScript Objects	17
2.2 Layout Objects: LayoutFrames and LayoutBags	18
2.3 Features and Modes	18
2.3.1 Features	18
2.3.2 Modes and Mode Clusters	19
2.3.3 The be_refresh() Function	20
2.3.4 Activation and Deactivation Hooks	20
2.3.4.1 Activation/Deactivation Function Vetoes	21
2.4 Hiding and Showing Layout Objects	21
2.5 Sizes	22
<b>3 A Larger Example: The Flag Demo</b>	23
3.1 What Does the Flag Demo Do?	23
3.2 Where to Start?: The ui_custom.js File	23
3.3 Auxiliary Files	24
3.3.1 The buttons.html File	25
3.4 The ui_custom.js Functions	27
3.4.1 initUIDefines()	28
3.4.2 initUIState()	28
3.4.3 initUILayout()	29
3.4.3.1 Placing your UI in the BeIA Global Namespace	33
3.4.3.2 Getting the Desired Left to Right Layout at the Top Level	33
3.4.3.3 Controlling the Layout Elements	34

**JavaScript Layout Engine . . . . . 37**

<b>1 Introduction</b>	37
1.1 Layout Objects: LayoutFrames and LayoutBags	37
1.2 Visibility, Features and Modes	37
1.2.1 Zero-Pixel Hiding	37
1.2.2 Features	38
1.2.3 Modes and Mode Clusters	38
1.3 Sizes	38
<b>2 Class and Function Reference</b>	38
2.1 Notation	38
2.2 Layout Classes	39
class LayoutBag	39
class LayoutFrame	40
2.3 Global Layout-Related Functions	42
Feature-Related API Functions	42
Mode API Functions	42
Miscellaneous Related Functions	43

**The Toolbar . . . . . 45**

<b>1 Changing the Appearance of the Toolbar</b>	46
1.1 Changing the Appearance of Toolbar Buttons	46
1.2 Changing Toolbar Layout and Other Appearance Properties with the “toolbar.html” File	47

<b>Bookmarks</b>	<b>49</b>
<b>1 How the Bookmarks Interface Works</b>	49
<b>2 Changing the Appearance of the Bookmarks Interface</b>	50
2.1 Changing the Appearance of Bookmark Controls	50
2.1.1 Names and Locations of the Bookmark Controls Image Files	50
2.2 How to Change the Look of Buttons—Advanced	51
2.3 Changing Bookmark Layout and Other Appearance Properties with the “index.html” File	51
<b>3 Preconfiguring the Favorites List</b>	52
3.1 “Favorites” File Format	52
<b>Buttons</b>	<b>55</b>
<b>1 An Example of Button Images in Action</b>	55
<b>2 The General Button Image Schema</b>	56
2.1 Location of Button Image Files	56
2.2 Button Image File Formats and Suffixes	56
2.3 Button Image Names	56
<b>3 Button Sizes</b>	57
<b>4 PNG vs. GIF Button Images</b>	57
<b>Alerts</b>	<b>59</b>
<b>1 The Alert Template</b>	59
1.1 A One Button Alert	59
1.2 A Two Button Alert	60
<b>2 The Alert Content</b>	60
<b>3 Invoking the Alert Script</b>	60
3.1 Through C++	60
3.2 Through JavaScript	61
3.3 Through TellBrowser	61

**Special Keys . . . . . 63**

<b>1 The Special Key Mapping File</b> . . . . .	63
<b>2 Actions</b> . . . . .	63
<b>2.1 Built-in Commands</b> . . . . .	63
<b>2.2 Shell Scripts</b> . . . . .	63
<b>2.3 JavaScript</b> . . . . .	64
<b>3 Modifying Special Keys Mappings and Functionality</b> . . . . .	64
<b>1 Browser</b> . . . . .	67
<b>1.1 Bad beos:// URL</b> . . . . .	67
<b>1.2 Bad file:// URL</b> . . . . .	67
<b>1.3 Bad http:// URL</b> . . . . .	67
<b>1.4 Unsupported Content or Scheme</b> . . . . .	68
<b>1.5 Password Request</b> . . . . .	68
<b>2 General Errors</b> . . . . .	68
<b>3 The Update Mechanism</b> . . . . .	69
<b>4 Midi</b> . . . . .	69
<b>5 Printer</b> . . . . .	69
<b>6 RealPlayer</b> . . . . .	70
<b>6.1 Clip Info</b> . . . . .	70
<b>6.2 Authentication</b> . . . . .	70
<b>6.3 Error Templates</b> . . . . .	71
<b>6.4 Error codes</b> . . . . .	71
<b>7 SmartCard</b> . . . . .	76
<b>8 Error-less Components</b> . . . . .	77

<b>User Interface Files</b> .....	<b>79</b>
<b>1 /boot/custom/cgi-bin/</b> .....	79
<b>2 /boot/custom/resources/\$LANGUAGE/</b> .....	79
<b>2.1 /boot/custom/resources/\$LANGUAGE/Alerts/</b> .....	79
<b>2.2 /boot/custom/resources/\$LANGUAGE/Bookmarks/</b> .....	80
<b>2.3 /boot/custom/resources/\$LANGUAGE/Cursors/</b> .....	81
<b>2.4 /boot/custom/resources/\$LANGUAGE/Days/</b> .....	81
<b>2.5 /boot/custom/resources/\$LANGUAGE/Errors/</b> .....	81
<b>2.5.1 /boot/custom/resources/\$LANGUAGE/Errors/template/</b> .....	82
<b>2.6 /boot/custom/resources/\$LANGUAGE/glyphs/</b> .....	82
<b>2.7 /boot/custom/resources/\$LANGUAGE/Home/</b> .....	82
<b>2.8 /boot/custom/resources/\$LANGUAGE/Intro/</b> .....	83
<b>2.8.1 /boot/custom/resources/\$LANGUAGE/Intro/WelcomeImages/</b> .....	83
<b>2.9 /boot/custom/resources/\$LANGUAGE/MediaBar/</b> .....	83
<b>2.10 /boot/custom/resources/\$LANGUAGE/Months/</b> .....	84
<b>2.11 /boot/custom/resources/\$LANGUAGE/PopUpDecor/</b> .....	84
<b>2.12 /boot/custom/resources/\$LANGUAGE/Settings/</b> .....	84
<b>2.12.1 /boot/custom/resources/\$LANGUAGE/Settings/widgets/</b> .....	84
<b>2.13 /boot/custom/resources/\$LANGUAGE/SoftKeyboard/</b> .....	85
<b>2.13.1 /boot/custom/resources/\$LANGUAGE/SoftKeyboard/key_graphics</b> .....	85
<b>2.14 /boot/custom/resources/\$LANGUAGE/Time/</b> .....	85
<b>2.15 /boot/custom/resources/\$LANGUAGE/Toolbar/</b> .....	85
<b>2.15.1 /boot/custom/resources/\$LANGUAGE/Toolbar/Images</b> .....	86
<b>3 /boot/custom/resources/scripts/</b> .....	86
<b>4 /boot/custom/sounds/</b> .....	87
<b>5 /boot/custom/special_keys/</b> .....	87
<b>6 /boot/home/config/settings/</b> .....	88





# UI Customization

## Introduction

Almost all graphical aspects of the BeIA browser—from the layout of the top level frames, to the images that are used for the cursors and toolbar buttons—can be customized. The only graphical elements that aren't customizable are scrollbars and the control objects used in Web forms (buttons, checkboxes, etc.).

This document describes the configuration environment—the languages and formats you'll use, the directory structure the browser expects, the files you must supply—in which you create your customized interface.

---

## 1 Programming Languages and Formats

All of the customizable components of the interface are defined in or loaded by HTML pages. These pages can be static or created dynamically through JavaScript and cgi-bin scripts. The customization work that you'll do will involve editing or replacing these HTML files (or the image files that they load). See “Web Browser” in the *BeIA Technical Specification* for descriptions of the languages and formats that the BeIA browser understands. Briefly, the browser supports most of HTML 4.0, JavaScript 1.3, Cascading Style Sheets (CSS1 and CSS2), and DHTML. The browser can display images in GIF, JPEG, and PNG format.

In addition to the off-the-shelf languages and formats, BeIA provides these native extensions:

- The `beos` JavaScript object provides some simple program-launching and message-sending functions. The `beos` JavaScript object is described in the *BeIA JavaScript* chapter.
- The `beos.binder` JavaScript object represents the Binder, a tree of global data that lets you communicate with the browser and other BeIA services. For example, you use the `beos.binder` object to ask for information about the machine's Internet connection. The Binder is described in the *BeIA Binder* chapter.
- The JavaScript Layout Engine is a software kit of JavaScript classes and functions that you use to layout the frames and other elements of your UI. The JavaScript Layout Engine is described in the *JavaScript Layout Concepts* and *JavaScript Layout Reference* sections of this chapter.

**Note:** The JavaScript Layout Engine is the latest addition to the BeIA UI tools. Its adoption makes some of the following documentation obsolete. The general principles layed out here still apply, but the tools through which you build the interface are much simpler and much more organized than the rest of this introductory section implies.

- A browser plug-in API (C++) is provided. This API lets you create “modules” that are loaded and executed by the browser. << The plug-in API isn't available yet. >>

## 2 Sample Configuration

A sample configuration is included with BeIA. The sample provides a directory structure that contains all the files that are needed to create a complete user interface.

The sample configuration comprises all files that you'll find in the `/boot/custom/resources` directory, as described in the next section.

---

## 3 Directory Structure

The BeIA file system contains the directory `/boot/custom/resources`. Each configuration that you create (i.e. each independent set of configuration files) must be placed in a separate subdirectory of `/boot/custom/resources`.

For example, let's say you have localized configurations for English, French, and German. To keep the configurations separate, you would put them in separate subdirectories (by convention, languages are represented by ISO two-character names):

```
/boot/custom/resources/en  
/boot/custom/resources/fr  
/boot/custom/resources/de
```

### 3.1 Sharing Files Between Configurations

You can use symbolic links to share files between configurations.

---

## 4 Configuration Variables

There are two configuration variables: `LANGUAGE` and `RESOURCES`.

### 4.1 The LANGUAGE Variable

To tell the browser which configuration subdirectory you want it to use, you set the `LANGUAGE` variable to the name of the subdirectory (en, en/Finance, Finance/en, etc.). Currently, the `LANGUAGE` variable is set when the user chooses a language in the Language Settings panel, and its value is brought out as a global value through the Binder.

### 4.2 The RESOURCES Variable

The browser defines the `RESOURCES` variable and initializes it to

```
/boot/custom/resources/$LANGUAGE
```

The `RESOURCES` value is prepended to all relative pathnames that are referred to from HTML pages that are loaded by the browser. For example, in a tag such as this...

```
<A HREF="Alerts/alert1.html">
```

... the HREF points to the file `/boot/custom/resources/$LANGUAGE/Alerts/alert1.html`.

The `LANGUAGE` component is evaluated every time `RESOURCES` is evaluated. Thus, if you update `LANGUAGE`, all subsequent references to `RESOURCES` will be aware of the change.

You can refer to [RESOURCES](#) anywhere you want in your own HTML code.

Keep in mind that the [RESOURCES](#) is “owned” by the browser. You never set it yourself—in other words, you must never send a message to beos telling it to set the value of [RESOURCES](#).

---

## 5 Localization

Currently, BeIA doesn’t provide any tools to help you create the *content* of your localized interface. But here’s what you need to do:

- Create a new project by copying and renaming an existing subdirectory of `/boot/custom/resources`.
- Translate all the hard-coded strings in your project’s files. The “User Interface Files” and “Errors and UI Strings” files should help you locate most of the files that contain text. Note that a number of graphics contain text as well.
- Set the [LANGUAGE](#) variable to point to your localized project, or edit the `/boot/custom/resources/$LANGUAGE/Settings/Languages.html` file to allow the user to choose your localized version. If your system supports more than one language, you’ll need to edit the `Languages.html` files in each project under `/boot/custom/resources`.

---

## 6 The Entry Point Files

When the BeIA browser is launched, it loads one of three HTML files:

- `$RESOURCES/firstboot.html` is loaded when the BeIA device is booted for the first time. The file (typically) contains a welcome message and initial instructions for the first-time user.
- `$RESOURCES/login.html` is loaded on subsequent boots. It prompts the user for a name and password (for example), and generally acts as a security manager for the device. This file may also be loaded when a sleeping device is woken up.
- `$RESOURCES/index.html` is loaded after the user has gained access to the device. This is the browser’s primary entry point into your configuration: Essentially everything the browser displays during normal operation is “loaded” by this file, either directly or through referred files. In the sample configuration, `$RESOURCES/index.html` lays out the main areas that the browser displays (toolbar, “tab bar”, content area) by defining and a set of frames.

You can modify or replace the entry point files, but keep in mind that they *must* exist.

How a device decides which page to display (i.e. between the first boot page and the login page) depends on its “boot mode,” as described in “Boot Mode” in the The BeIA User Interface.

---

## 7 The Content Area

The BeIA browser takes over the entire screen. It’s assumed that you’ll want to divide this overall area into “task-specific” areas: You’ll want an area for a toolbar, another area for branding, another for advertisements, and, most important, you’ll want a content area that displays pages that are downloaded from the Web.

## 7.1 Creating the Content Frame

All browser configurations are expected to define an HTML frame for the content area. You must set the name of this frame to “\_be:content”. For example:

```
<FRAME SRC="home_page" NAME="_be:content" NORESIZE et al. '>
```

## 7.2 Special Treatment of the Content Frame

The content frame (i.e. the frame named “\_be:content”) is treated specially by the browser: A reference to the top frame, when made from an HTML file that’s loaded into the content area, resolves to the content frame. In this way, the browser prevents Web pages from accessing frames that are “above” the content frame.

### 7.2.1 References to the Top Frame from within HTML

From within HTML, references to the top frame occur when a `target` attribute is set to “\_top”. For example, let’s say the user loads a Web page that contains a statement such as this:

```
<a href="some_file.html" target="_top"> click here to refresh </a>
```

The browser traps the reference to “\_top” and replaces it with “\_be:content”. Similarly, a target of “\_parent” *directly* within the content frame is replaced with “\_be:content”.

### 7.2.2 References to the Top Frame from within JavaScript

In JavaScript, references to the `top` object, which represents the top frame, are replaced by references to the “\_be:content” frame. For example, if a page in the content frame tries to change the background color of the top frame thus...

```
top.document.bgColor="blue"
```

...the browser will catch the top reference and convert it to the “\_be:content” frame.

---

## 8 Alerts

When an error occurs (the browser can’t get to the network, can’t load the requested page, can’t find a printer, and so on), an error code is sent back to the browser. To report the error to the user, the browser displays an alert. In BeIA, an alert is a dynamically-created HTML page that combines an **alert template** (which defines the layout of the HTML page) with some **alert content** (the text that’s displayed in the page). The alert page is then displayed in the browser’s content area.

For more on alerts, see “Alerts.”

# UI Customization

# Design Considerations

---

## 1 The Audience

BeIA was created to run on devices that are focused on a particular subset of a traditional PC's sprawling functionality. This is a blessing for designers in that the user is specifically defined, and design solutions can cater to them directly. Of course, this requires an intimate understanding of the market for which the product is destined. Before undertaking a BeIA interface, make sure you know how the user that will ultimately make use of your product, and plan to test your work with representative subjects early and often.

---

## 2 The Browser

The technical foundation of BeIA is a combination of BeOS and a highly customized Opera browser. On startup, BeIA launches immediately into Opera, which takes over the full screen. Designing for Opera is a welcome relief for designers struggling to reconcile the ever-changing demands of competing browsers. Opera is known for rigorous adherence to HTML standards, and creators can be assured that their users will view the UI as it was intended—via a single, predictable application.

Unlike a PC-based browser, BeIA's Opera has no application-specific controls. Its HTML rendering space spans the screen edge-to-edge, allowing its controls to be designed and displayed using the same graphic resources web designers have come to take for granted. This flexibility is a double-edged sword—a HTML-based user interface raises the risk of easy confusion between control and content. Take care to choose a look and feel that will readily distinguish between the two.

---

## 3 Controls

Depending on your application of BeIA, your control set will obviously vary. Some general guidelines:

- Remember that the elegance of the Internet Appliance comes from its deliberately focused feature set. Resist the temptation to make your product a limited PC. Make it a robust appliance instead.
- Ideally, a well-focused Internet Appliance can expose all of its main controls without cluttering or confusing the user experience. Notably excepted from this guideline are secondary, seldom-used controls (such as settings), which are certainly appropriate to move to a different level (such as a different screen). Just ensure that it is easy to get to settings from the main screen, and easy to get back to the main screen from settings screens.
- If your BeIA application provides a browser interface allowing your user to view internet content, strive to balance the user's need to have clear, simple, and easy to use controls with their desire to view content quickly and efficiently. Control size should be generous enough to be comfortably used by a wide range of users with varied manual dexterity. Simultaneously, your control area should not

be so large as to usurp too much valuable screen real estate that the user would prefer to use for content viewing. Strive to limit your control-to-content area ratio to a maximum of 15%.

- BeIA is tremendously flexible from a graphic perspective. Designers are encouraged to use this to its full aesthetic advantage, but strive to maintain a reliable, functional control language. Graphically tell the user what to expect through consistent treatment of object vocabulary and state. Which elements are interactive, which provide information, and which are decorative? How do control elements behave when clicked, rolled over, or disabled? How does the user know an action has begun or completed?
- Controls that fall into natural functional groups should be associated by both proximity and appearance (similar size, shape and graphic vocabulary). When possible, further distinguish controls through hierarchy by graphically emphasizing important or frequently-used functions. (For example; “Play”, “Stop”, “Fast Forward”, and “Rewind” buttons in an MP3 appliance could have the same visual style [background and foreground color, etc.], but the “Fast Forward” and “Rewind” buttons could be slightly smaller, to emphasize the fact that they are extra, rather than basic, controls.) Conversely, beware inadvertent associations between unrelated controls that may share similar appearances or groupings.
- While considering the arrangements of your controls, keep basic graphic design principles in mind. Establish a layout grid when appropriate, and mind typographic conventions.

### 3.1 Labels

All controls need some sort of label to identify their function. These labels can be either iconographic or text-based, but keep the following caveats in mind:

- Label controls of similar hierarchical importance or function consistently. Example: Don’t label a play button with a forward triangle and a stop button with the word “Stop”
- When using text labels, keep in mind that as your UI is translated into other languages, the length of your text string could vary greatly.
- When using icons, maintain visual and stylistic similarity to avoid user confusion. This is especially true if using photo-realistic icons, which must be carefully managed to be visually consistent with each other and other UI elements.
- Remember that an icon that seems “obvious” to you may not seem so obvious to another person in a different culture (or might have an obvious meaning different than what you intended). It’s common today for photocopiers to ship with only icons on their buttons—and an accompanying “cheat sheet” that lists what those “obvious” icons actually mean. Don’t make that mistake. Go to the trouble to make sure an icon is really clear, or provide an accompanying text label.

---

## 4 Panes

In keeping with the ideal that all relevant information is always available to the user, BeIA interfaces seldom use multiple overlapping windows that may obscure content or controls. However, it’s often necessary to visually and organizationally subdivide the display into discrete areas.

To accomplish this, BeIA makes use of *panes*, which are technically analogous to frames in a Web page. Panes can be opened, closed, expanded, and minimized, via JavaScript automation or user control. The chief distinction between windows and panes is that the panes are placed alongside one another, expanding and contracting as necessary to fit onscreen. Panes never overlap.

Panes are best used in situations where the user is cross-referencing two (or more) tasks that are at least peripherally related, or which make sense to do in parallel, i.e.:

- Listening to an MP3 while browsing the web (having the volume control and playlist always handy might be desirable.)
- Adding a bookmark to the “Favorites” list, while continuing to display part of the web page being bookmarked, for context.

Using panes to show two completely unrelated tasks clutters the screen and is visually confusing. For example, there is little reason to display both an e-mail pane and a web browser pane at the same time. The user will most likely be doing one or the other...why clutter the screen with an unused application?

It may be tempting to use panes to allow simultaneous display of more than one web page, but given the limited display area of most BeIA devices, this is probably not a good idea in general. BeIA *will* show two web browser panes under certain conditions. When a website spawns a child HTML page (via JavaScript), Wagner will split the screen to show the two pages (parent and child) at the same time. The child is a read-only HTML display—the navigation bar and history still apply only to the parent. If the user clicks a link in a child pane, the link loads as the parent at the top level. The child pane stays onscreen until it is dismissed by the user via a close control. This sort of multipane display can be useful if, say, the child is a “Table of Contents” or similar listing.

---

## 5 Feedback and Alerts

As they become familiar with your interface, users will take comfort in the reassurance that feedback provides. Strive to make your UI fast and efficient so users make an immediate connection between action and consequence. When a user begins an action that will take some time, provide progress information or other feedback so the user still realizes that their request is being addressed.

When more directed feedback is required, it is often necessary to provide alerts with more weight than passive feedback can provide. In these circumstances, use non-modal alerts whenever appropriate (in other words, if the alert has no bearing on what the user is currently doing, open it in another pane and let the user continue to work on his task). Modal alerts are appropriate when the user needs to address something immediately, or when the user must proceed through a set series of actions in order.

---

## 6 Settings panels

Keeping to the idea that Internet Appliances should not be miniature PCs, minimize the number of settings a user must deal with, and keep these controls “intuitive”. Almost everyone can easily understand a slider control labeled “*Volume*”, and almost everyone will want to have control of the volume of their machine. On the other hand, fields for data like “*Primary DNS*” and “*Gateway*” are reasons people don’t want to deal with PCs. Eliminate such fields if you can.

Much of the design of settings panels is simply common sense, but common sense from the point of view of the user—not necessarily from the point of view of you, a technically literate engineer. Here are some things to think about:

- Consolidate like items, split disparate items. For example, it may make sense to present audio and video controls on the same page if they will fit (they are related), but to present them in different areas labeled “*Audio*” and “*Video*”, because much of the time, a person will not be interested in both; an MP3 user is not going to be interested in the video options. On the other hand, splitting network settings into “*TCP/IP Settings*” and “*Firewall Configuration*” may seem logical from your point of view, but not from the user’s; they neither know nor care what TCP or a firewall are. From the point of view of both the user and your own technical support department, it may be much better to divide such options into “*Basic Setup Options*” and “*Advanced Setup Options (Don’t worry about these at first)*”.

- Don't overload a settings panel by putting too many things on it. It may be tempting to fit all settings onto one panel just because you can, with the idea that quick access to all settings will make it easier for the user to adjust what they want. However, this is true only if the user changes their settings a lot—which is unlikely. In the typical case that the user changes settings infrequently, an “all-in-one” settings panel will be intimidating, and will not save any appreciable time.
- It may make sense to have a setting appear on more than one panel. Having a setting labeled “Volume” in both the “Audio Playback” panel and the “Video Playback” panel is easy to understand and useful.

---

## 7 Technical Considerations

- When possible, use “real text” rather than text embedded into graphics. Text that is part of a graphic cannot be easily internationalized (you'll have to do a different version of the graphic for each language you ship in), will not gracefully upgrade to different screen sizes, and requires redoing the graphic for every change of style in your UI. Also, graphics take up a good deal of memory space.
- Allow for the fact that an Internet Appliance device has very limited resources compared to a typical PC, in terms of both storage and speed. Keep your code compact by concentrating on a core feature set, and do performance testing to ensure adequate speed on your target device.
- Use C++ plugins for situations where the JavaScript isn't fast enough. You may also wish to use C++ for complex plugins even if speed isn't an issue—JavaScript is not well-suited for large pieces of code.

---

## 8 Recommended Reading

Norman: *Design of Everyday Things*, *The Invisible Computer*

Cooper: *The Inmates are Running the Asylum*

Jef Raskin: *The Humane Interface*



# UI Customization

# JavaScript Layout Concepts

BeIA-based devices utilize HTML to present data on-screen, resulting in a great deal of flexibility in how data, pictures, and other elements can be displayed. However, working with HTML directly (and in particular, generating HTML dynamically, to give the appearance of a flexible and responsive user interface) can be difficult. To solve this problem, BeIA includes a JavaScript library which permits you to build and work with interface layouts much more easily than would be the case when using HTML directly. The following document, along with its companion liblayout reference document, describes the use of this library.

---

## 1 Prerequisites

In order to use the layout engine, you'll need a reasonable familiarity with JavaScript, including the semantics of JavaScript classes and object-oriented programming with JavaScript. You'll also find a knowledge of HTML frames and framesets to be useful, though such knowledge is not strictly required.

---

## 2 Fundamental Concepts

### 2.1 Abstract the Creation of HTML Markup into HTML-Generating JavaScript Objects

A standard method of creating Web pages is to write the HTML content which defines them, either by hand, or using some sort of tool which writes the HTML for you as you manipulate a user interface. In either case, you generate the HTML statically; once written, it is never changed.

BeIA functions differently. It generates much of the HTML it uses dynamically, using JavaScript code embedded in “stub” HTML files. For example, if the BeIA browser displays a page showing a set of controls by referencing a file called `Controls.html`, it's likely that much or all of the HTML defining the layout and contents of that page is not contained directly in `Controls.html`, but that instead, `Controls.html` simply contains a piece of JavaScript code which writes out the appropriate HTML every time the `Controls.html` document is referenced. This means that the HTML code provided by `Controls.html` can (potentially) change every time the file is displayed, so as to reflect changes in the system state.

However, directly writing JavaScript code which outputs appropriate HTML is even more tedious and error-prone than directly writing HTML. To solve this, the BeIA layout engine abstracts the idea of onscreen HTML display structures (frames and framesets) into JavaScript objects. Instead of working with HTML markup tags, you simply create a JavaScript object of an appropriate type, and insert it into a data structure. For example, you might create a JavaScript object which represents a vertically split HTML frameset, and then give it two child objects which each represent an HTML frame. Once you've created this structure, it is the responsibility of the objects in the structure to write out the HTML needed for proper onscreen rendering—you never see or work with HTML code. This approach works well because the JavaScript code needed to define such structures is much more concise than the amount of HTML

such code generates. In addition, your code will be checked for many errors by the JavaScript interpreter, which also helps to identify mistakes.

## 2.2 Layout Objects: LayoutFrames and LayoutBags

A *layout object* is a JavaScript object which represents some element of a web page layout. The BeIA JavaScript layout package currently offers two types of layout objects, each defined as a JavaScript class. A `LayoutFrame` object represents a single rectangular HTML frame displaying some sort of content, such as a page of text. A `LayoutBag` object is a collection of `LayoutFrames`; it represents an HTML frameset, and displays the contents of its child `LayoutFrames` either side-by-side, or stacked one on top of the other.

## 2.3 Features and Modes

A great deal of the UI flexibility of the BeIA system depends on the ability of the layout library to quickly and easily hide or show `LayoutFrame` or `LayoutBag` objects. For example, in some versions of BeIA, a “Bookmarks” frame is always present in the browser window, but is only visible if the user requests it. This approach is more reliable and easier to program for than actually modifying the data structure of the browser window to include a bookmarks frame.

The visibility of layout objects can be set directly via their associated visible property, but in general, you will probably find it more convenient to control visibility of objects with *features* and *modes*.

### 2.3.1 Features

*Features* are effectively boolean variables whose state is linked to the visibility of one or more layout objects. Often a feature is used to control whether or not a particular feature (in the usual sense of the word) of the UI is visible to the user, hence the name.

For example, let's say that you have a settings screen for setting up some aspect of your device, and that contained within that screen is a subframe called *advancedSettings*, an instance of `LayoutFrame`. You'd like to provide the user the option of clicking a piece of text to show the advanced settings frame. Using features, here's how you could do this:

- 1) In your HTML text for the main content of the settings window, insert something like

```
<H2 ONMOUSEUP="beos.globals.features.Open("settings:advanced");  
beos.realTop.be_refresh()>Show Advanced Settings</H2>
```

This ensures that whenever the user click on the heading “Show Advanced Settings” in the main settings area, the feature “settings:advanced” will have its associated boolean value set to true. The name of the feature is arbitrarily chosen by you; `Open()` is a method in the layout library.

- 2) The relevant portions of the main body of your code will look something like this:

```
// settingsFrame is a LayoutFrame object you've defined earlier in your code;  
// the AddChild method returns its argument as a convenience, so the following line of  
// code leaves the new frame stored in the advancedSettings variable.  
var advancedSettings = settingsFrame.AddChild(new LayoutFrame(...));  
...configure the contents of advancedSettings if necessary...  
  
// Create the new feature to control whether advancedSettings is visible or not.  
Disregard  
  
// the null arguments for the time being.  
beos.globals.features.Add("settings:advanced", null, null);
```

```
// Now, set how the new feature controls its associated object or objects; in this
// case, we state the that given layout object should be visible when the feature
// is true, and invisible otherwise.

CoupleLayoutObjectToFeature(advancedSettings, "settings:advanced", true);

// Finally, ensure that the features is initially false, so the "Advanced Settings" frame
// is initially
// hidden.

beos.globals.features.Close("settings:advanced");
```

Once the above setup has been accomplished, you can (as the code associated with ONMOUSEUP does) cause the advanced settings frame to appear or disappear simply by manipulating the feature, and then calling a screen redraw with `beos.realTop.be_refresh()`.

Using features to control the visibility of screen frames confers a number of advantages:

- A single feature can control the visibility of multiple layout objects. This frees you from remembering to change the visibility of each layout object manually.
- A feature decouples the conceptual aspect of changing the UI in some way from the changes that are made, permitting you to easily alter the changes that take place without which causes these changes.
- The provision of activation and deactivation hooks (discussed below) means that changes to accomplish tasks in a way which would be much more difficult if your code were wired to your input-related (i.e. ONMOUSEUP and similar) code.
- With the right feature names, features can also contribute greatly to internal

### 2.3.2 Modes and Mode Clusters

A feature is effectively a variable with two states, true or false; elements of the UI are visible or invisible dependent on the state of this variable. (In common use, they are visible when the feature is true and invisible when false, but that is not required; if the third argument to `CoupleLayoutObjectToFeature` is false, the UI element will be visible when the feature is false and invisible when it is true.)

A *mode cluster* is just a generalization of a feature; instead of having only two values, a mode cluster can take any one of a set of values, each of which is called a *mode*. The set of modes associated with a mode cluster is defined by you, the programmer. The mode-related part of the API is slightly different than the feature-related part, but the differences are strictly a result of the fact that a mode cluster can take one of many user-defined mode values, while a feature can take one of two predefined boolean values.

As a very quick idea of how a mode cluster would be created and used in code, consider the following:

```
// The BookmarkBag is going to contain three frames, only one of which will be
// visible at any one time, depending upon the setting of an associated mode cluster.
// someHigherLevelBag is assumed to be a containing LayoutBag previously defined.

var bookmarkBag = new LayoutBag(...); someHigherLevelBag.AddChild(bookmarkBag);

// Create frames that show the bookmarks at high, medium, and low resolutions,
// and add them to the bookmark bag.

var highresBookmarks = new LayoutFrame(...); bookmarkBag.AddChild(highresBookMarks);
var mediumresBookmarks = new LayoutFrame(...); bookmarkBag.AddChild(mediumresBookMarks);
var lowresBookmarks = new LayoutFrame(...); bookmarkBag.AddChild(lowhresBookMarks);

// Create a new mode cluster called "bookmarks:zoomlevel".
```

```
beos.globals.modes.AddCluster("bookmarks:zoomlevel");

// Allowable values for this cluster will be "low", "medium", and "high".
beos.globals.modes.AddMode("bookmarks:zoomlevel", "low", null, null);
beos.globals.modes.AddMode("bookmarks:zoomlevel", "medium", null, null);
beos.globals.modes.AddMode("bookmarks:zoomlevel", "high", null, null);

// The highresBookmarks frame should be visible only when the "bookmarks:zoomlevel"
// mode cluster is set to "high", invisible otherwise. A similar comment applies to the
// other
// two frames.

CoupleLayoutObjectToMode(highresBookmarks, "bookmarks:zoomlevel", "high");
CoupleLayoutObjectToMode(mediumresBookmarks, "bookmarks:zoomlevel", "medium");
CoupleLayoutObjectToMode(lowresBookmarks, "bookmarks:zoomlevel", "low");

// Start off with the bookmarks showing the highest resolution view.
beos.globals.modes.Set("bookmarks:zoomlevel", "high");
```

The code above sets up part of a user interface which has the ability to display the user's bookmarks in one of three ways; a high-resolution display (which might list all of the bookmarks), a medium-resolution display (perhaps showing only the frequently-accessed bookmarks), and a low-resolution display (which might show things only at the top level of the bookmark hierarchy, not those contained in folders, for example.) This is done by having a bookmark LayoutBag (i.e. HTML frameset) which contains all three of these views; the "bookmarks:zoomlevel" mode cluster is set up so that only one of the bookmark frames is visible at any given time.

Within each of the bookmark frames would be a button or other control to change the zoom level; for example, the highresBookmarks frame might display a button called "Zoom Out", when pressed, this button would simply execute the code:

```
beos.globals.modes.Set("bookmarks:zoomlevel", "medium");
```

As far as the user is concerned, the apparent effect would be that the high-resolution view of the bookmarks is replaced with the medium-resolution view.

### 2.3.3 The be\_refresh() Function

One thing that modes and features do not handle is automatic screen redraws. After changing a mode or feature, you'll need to call `beos.globals.realTop.be_refresh()` to have the display redraw, and reflect the change in the UI. This is done so that you can perform a number of mode and/or feature requests at the same time and then redraw the screen only once, greatly diminishing the computational load, and increasing the responsiveness of your device. It may be tempting to put a `be_refresh()` call after every mode or feature change, but (unless perhaps you're debugging) don't do it. Only call `beos.globals.realTop.be_refresh()` once all of your mode and features changes have been made for a particular input event.

### 2.3.4 Activation and Deactivation Hooks

You may have wondered, in the sections above, about the presence of "null" arguments in methods for creating new features and modes, i.e. in lines like

```
beos.globals.features.Add("feature:something", null, null);
```

or

```
beos.globals.modes.AddCluster("toplevel:choice");
```

```
beos.globals.modes.AddMode("toplevel:choice", "choice1", null, null);
```

These arguments may be used to pass in functions which will be called upon activation or deactivation of a mode or feature, if you need to perform certain tasks when a feature or mode is changed. In both the Add() method (for features), and the AddMode() method (for modes), the first null above can be replaced with an *activation function*, and the second null can be replaced with a *deactivation function*.

In both features and modes, an activation function (if provided) is called when the mode or feature is about to become active, and a deactivation function (if provided) is called with the mode or feature is about to become inactive. A feature becomes active when its value is set to true, and becomes inactive when its value is set to false. A mode becomes active when it becomes the value for its associated mode cluster, and becomes inactive when it is the value for its mode cluster, and then that mode cluster changes to a different value.

**Note:** Both activation and deactivation functions are called *before* the value of their associated feature or mode cluster has been changed to its new value. This is done for the reasons outlined below.

Both activation and deactivation functions are called *before* the value of their associated feature or mode cluster has been changed to its new value. This is because activation/deactivation functions do more than just perform some work when a feature or mode change takes place; they may actually *veto* that feature or mode change, as outlined in the subsection below.

#### 2.3.4.1 Activation/Deactivation Function Vetoes

Any function used as an activation or deactivation function should return a boolean value. The normal return value is true; this indicates that the function has accomplished what it needs to do, and the mode or feature change can proceed. If the function returns a value of false, this indicates that for some reason the change in the value of the associated feature or mode cluster should not be changed as was requested by the `beos.globals.features.Open(...)`, `beos.globals.features.Close(...)`, or `beos.globals.modes.Set(...)` call which resulted the activation/deactivation function being called. If the library code receives a value of false from an activation or deactivation function, it will abort the mode or feature change.

As an example of this, consider the following code:

```
...  
beos.globals.modes.Set("toplevel:screen", "bookmarks");  
...
```

and also assume that immediately before this line executes, the “toplevel:screen” mode cluster has the value “browser” as its mode. Execution of the line above will then cause the following chain of events.

1. If it exists, the deactivation function associated with mode “browser” in mode cluster “toplevel:screen” is executed; if the result is true, go on to the next step, otherwise abort. (Of course, if there is no such deactivation function, this step is skipped.)
2. If it exists, the activation function associated with mode “bookmarks” in mode cluster “toplevel:screen” is executed; if the result is true, go on to the next step, otherwise abort. (If there is no such deactivation function, this step is skipped.)
3. Set the value of the “toplevel:screen” mode cluster to “bookmarks”.

## 2.4 Hiding and Showing Layout Objects

You can make `LayoutFrame` and `LayoutBag` objects visible or invisible on the screen through the use of their boolean `visible` property; when `visible` is set to true the associated layout object will appear on the screen, and when `visible` is false, the associated object will not appear. In general, you will

probably want use modes or features to control when your object becomes visible or invisible; see See “Features and Modes” on page 18..

However, simply hiding a layout object is not quite the end of the story. There are actually two ways in which a layout object may be hidden; they both appear the same to the user, but have different consequences for the internal state of the JavaScript data they represent. It’s important to understand these differences; to explain them fully, we’ll need to take a brief look at how BeIA renders HTML code from layout objects.

Consider a `LayoutBag` object *M*, which contains two children, *A* and *B*. When both are displayed, they are displayed side-by-side, with *A* on the left and *B* on the right. However, at various times, *B* may be hidden.

Due to the dynamic nature of BeIA, there is no fixed HTML which defines *M*, *A*, and *B* on the screen; instead, whenever *M* and its contents are displayed, the required HTML code is generated dynamically. So, when *A* and *B* are both visible, a display of *M* will result in the generation of HTML code which defines a frameset containing two frames, one for *A* and one for *B*.

Now, what happens when *B* is hidden? The obvious answer is to simply change the generated HTML so that it defines a frameset containing a single frame, corresponding to *A*. Any mention of *B*, and the web page it displays, will be omitted from the generated HTML. This is the “standard” method of hiding a frame (or frameset).

In general, the standard hiding method works well, but there are some instances where it will cause problems. Consider the case where the page referenced by (displayed in) *B* contains JavaScript code which defines data objects or code which is needed by *A*, even when *B* is not visible. Under standard hiding, *B* (and therefore its web page) is not referenced in the generated HTML code, and the semantics of JavaScript mean that any data or function definitions in that web page are not loaded—they simply do not exist. Any attempt by JavaScript code in *A*’s web page (or from anywhere else) to access such data or functions will result in an error, and cause the system to fail or misbehave.

To solve this problem, another method of hiding is available, called *zeroPixelHiding*. To use this method of hiding for a `LayoutBag` or `LayoutFrame`, you simply ensure that when creating that `LayoutBag` or `LayoutFrame`, you provide a property of `zeroPixelHiding` with a value of `true` in the properties argument object. Once this is done, you can hide and show the object in the regular manner (e.g. by using features or modes: See “Features and Modes” on page 18.), and the visual effect will remain the same as with standard hiding. However, the HTML that is generated for an object hidden using `zeroPixelHiding` will be somewhat different than HTML generated for an object hidden in the standard manner; instead of any mention of that object being omitted from the generated HTML, the object made invisible with `zeroPixelHiding` (or more correctly, it’s underlying displayed web page, as given in its “location” property) will still be mentioned in the HTML, but will be displayed in a screen area of zero dimensions, making it invisible. However, because the object’s web page is mentioned in the HTML, that web page will be loaded, and any JavaScript objects or functions defined therein will be available.

This may seem abstract, so let’s take a quick look at a basic example that appears in BeIA; bookmarks. In general, the layout object which displays a user’s list of bookmarks is hidden; it is only made visible when the user requests it, typically by clicking on the “Favorites” button. However, the user has, at any time, the option of adding the site currently being browsed to the list of bookmarks, which means that the bookmark data structure, defined within the bookmarks HTML code, must always be available. To ensure this, the bookmarks frame is hidden using `zeroPixelHiding`, rather than standard hiding.

## 2.5 Sizes

Various layout constructors or functions require you to specify a size. Generally, you can give a size in one of several ways:

- Specify a number, which denotes a size (width or height) in screen pixels.

- Specify a string of numbers, such as "10%"; this specifies that the size should be a percentage of whatever is available in the appropriate dimension (width or height).
- Specify the size as "\*"; this signifies that whatever space is free in the appropriate dimension is allocated to this size request.

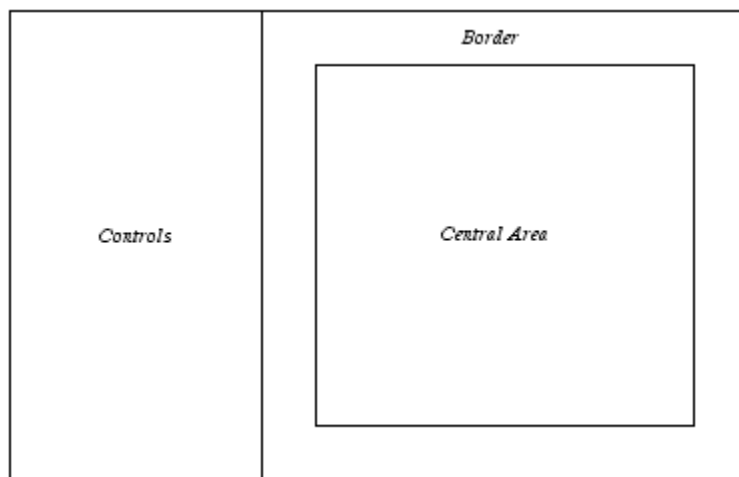
---

### 3 A Larger Example: The Flag Demo

Let's take a look at how the layout library is actually used. We'll do this through a simple example, which I've called the flag demo because it produces onscreen pictures similar in appearance to some flags.

#### 3.1 What Does the Flag Demo Do?

The flag demo displays a screen that looks something like this:



The left part of the screen will be static, and contains some text headings which the user can click on to change the appearance of the right part of the screen. The right two-thirds or so of the screen displays a border which may be made visible or invisible by means of a feature (see xxx), and a central area which may be set to display one of various possible content sources by means of a mode (see xxx). In the FlagDemo example, the central area displays either solid red, solid green, solid blue, or a mix of all three (like a tricolor flag).

#### 3.2 Where to Start?: The ui\_custom.js File

The first question we need to ask is, where do we put our code? (More correctly, where do we put the main body of our code? As we'll see, we'll have a small amount of HTML and JavaScript in files created specifically for this example.) BeIA is a big system, but fortunately, you need only worry about one file. All UI customization starts with xxx/ui\_custom.js, which is loaded automatically as part of the BeIA boot sequence, after all of the "core" functionality of BeIA has been loaded.

ui\_custom.js is composed of a minimum of three functions (of course, you may define further functions of your own there); these functions are initUIDefines(), initUILayout(), and initUIState(), which are called one after the other during the boot process. You may put anything you want into these functions, but their intended usage is as follows:

- `initUIDefines()`: Used for the definition and creation of those objects which are not layout dependent on the pre-existence of layout objects. In practice, this often means the definition of features, and specialized objects you create yourself.
- `initUILayout()`: Used to create the layout objects that define your UI, and to link them to those objects. This is where you'll create all of your `LayoutBag` and `LayoutFrame` objects, and also where you'll specify what is to be controlled by features or modes.
- `initUIState()`: Used to initialize the state of your UI to a consistent starting value. Often, these functions simply consist of a number of calls to `beos.globals.features.Open()/Close()` or `beos.globals.features.Toggle()`.

In the current version of BeIA, these functions are simply invoked sequentially. You could, therefore, put all of your UI code in (say) `initUIDefines`, leaving the other two functions empty. However, there are two good reasons for following the guidelines above:

1. Future versions of BeIA may perform “behind-the-scenes” work between these function calls, and could rely on code being written to the above guidelines.
2. Splitting up your UI code in the manner outlined above can make the code much easier to understand and modify. In particular, having all of the feature and mode cluster definitions in `initUIDefines()` makes it easy to understand the various states your UI can assume, and having the state initialization code in `initUIState()` makes it easy to change the initial state of your UI without hunting through the (typically much larger) `initUILayout()` function.

**Warning:** Even if you do not use one of the above functions, you must include them with an empty body. BeIA will not start unless all three of the above functions are included in `ui_custom.js`.

### 3.3 Auxiliary Files

Not all of the code we'll write will be in `ui_custom.js`. When you develop using BeIA's layout functionality, you'll often find it convenient to define the “lowest level” frames of your UI using HTML, and `FlagDemo` is no exception. First, let's identify what lowest-level frames are needed for `FlagDemo`.

We need a frame containing the control text, and the JavaScript code necessary to have that text react to mouse clicks. This code will go into a file called “`buttons.html`”. We'll get back to this in a bit.

We need frames for the border elements, i.e. the left, right, top, and bottom parts of the border. Each of these frames is simply a blank HTML page with a background color of magenta. Since the HTML for each of these border parts is identical, we can use the same HTML file for all border parts! We'll call this file “`border.html`”. Its contents are very simple; just

xxx

Similarly, we need frames displaying backgrounds of red, green, and blue, so we can display these colors in the central part of the display. These files are almost identical to the “`border.html`” file above. We'll call them “`red.html`”, “`green.html`”, and “`blue.html`”. Below is the text of “`red.html`”. “`green.html`” and “`blue.html`” differ only in their background color.

Each of the above files will be referenced by a “location” property of one or more `LayoutFrame` objects, to incorporate the files' HTML into the final display.



### 3.3.1 The buttons.html File

The buttons.html file has to do two things:

1. Provide controls by which the user can affect the state of the BeIA system.
2. Link these controls logically so that activating them does actually cause the desired state change.

The first of these can be done in any number of ways; HTML buttons, clickable images, and so on. To keep this part of it simple (since layout, and not controls, are our focus here), we'll just use HTML text fragments with the ONCLICK attribute set, as our controls. In a real user interface, you'd probably want to do something more visually appealing.

```
<html>
<body>
<script language="javascript">
    function toggleBorder() {
        if (beos.globals.features.Get("flagdemo:border")) {
            beos.globals.features.Close("flagdemo:border");
        } else {
            beos.globals.features.Open("flagdemo:border");
        }
        beos.realTop.be_refresh();
    }

    function showColor(color) {
        beos.globals.modes.Set("flagdemo:mode", color);
        beos.realTop.be_refresh();
    };
</script>

<H2 ONMOUSEUP="toggleBorder()">Border</H2>
<br><br><br>

<H2 ONMOUSEUP="showColor('red')">Red</H2>
<p>

<H2 ONMOUSEUP="showColor('green')">Green</H2>
<p>

<H2 ONMOUSEUP="showColor('blue')">Blue</H2>
<p>

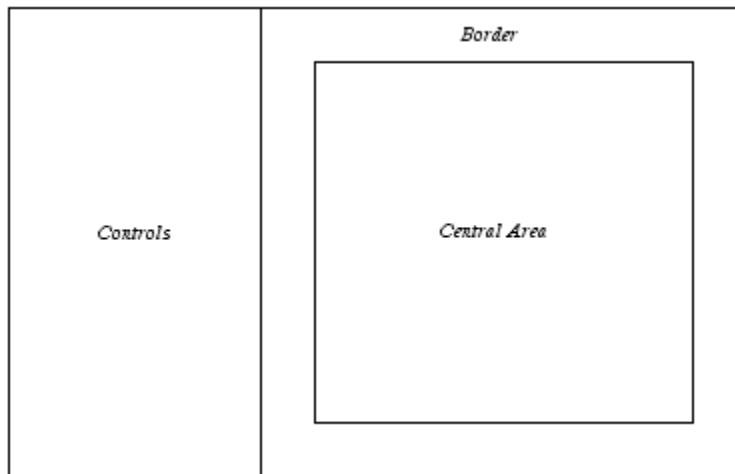
<H2 ONMOUSEUP="showColor('all')">All</H2>
<p>
```

</body>

## Designing the Layout

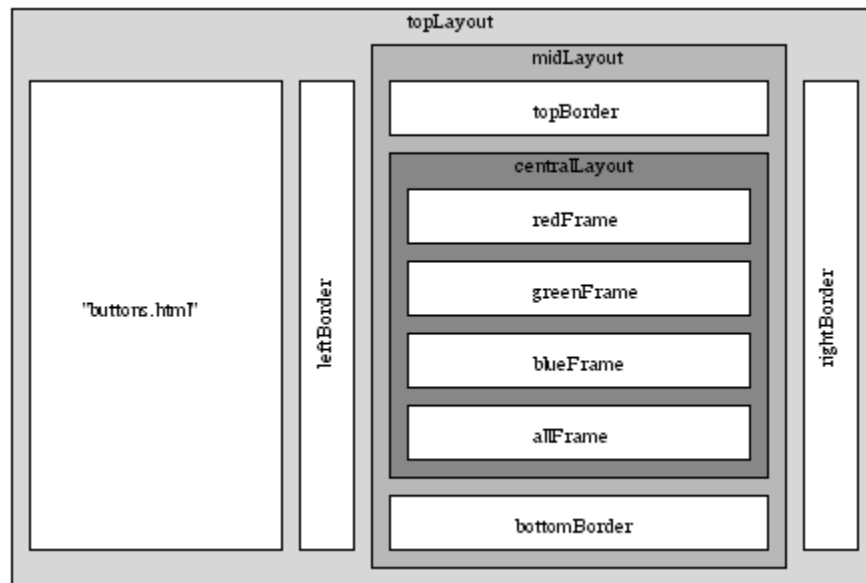
The next step to achieving our desired application is designing the layout, using `LayoutBag` and `LayoutFrame` objects. The easiest way to do this is to sketch out the layout ahead of time on a pad of paper, and figure out how everything “fits together”. We do this below for the `FlagDemo` example.

Our desired UI layout is something like this:

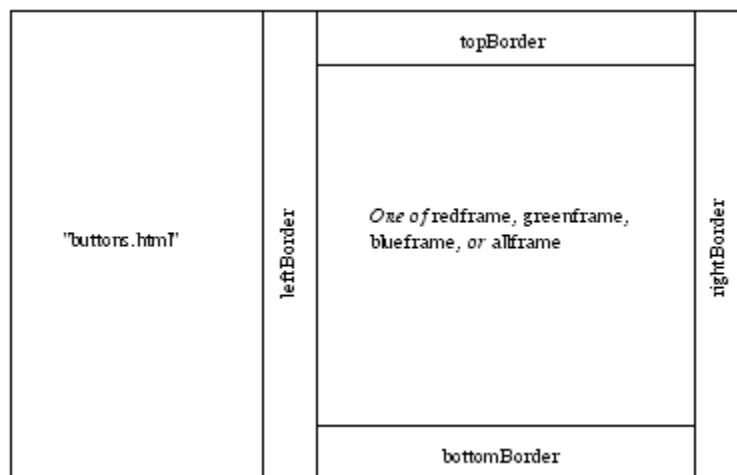


This contains a mix of frames; some are beside one another, while others are above/below one another (Note that the border must be split into four rectangular subframes in order to be drawn using frames). All of this layout is achieved using `LayoutBag` objects; however, `LayoutBag` objects correspond to HTML framesets, and as a result, must have either a vertical orientation (i.e. the borders between subframes go up and down, meaning the subframes are arranged side-by-side), or a horizontal orientation, with the subframes above and below one another. To achieve a display with a mixture of side-by-side and over-under frame arrangement, we need to use nested `LayoutBag` objects.

One possible layout is shown below. The `LayoutBag` containers are displayed in various shades of gray, while the `LayoutFrame` objects are shown in white. (“allFrame” is actually a `LayoutBag` containing three further frames, but we display it here as a frame to avoid cluttering up the diagram with too much detail).



When the UI is used, the LayoutBags will be sized to exactly fit their enclosed LayoutFrames; the “extra space” they take up around the borders in the above diagram was included only to show the nesting. As well, in the central area, only one of redframe, blueframe, greenframe, or allframe will be visible at a given time. As a result, the final look will be something like this:



Which is, of course, exactly what is desired.

### 3.4 The ui\_custom.js Functions

It's finally time to look at the code necessary to achieve our desired UI. This is one place where the benefits of splitting the code in ui\_custom.js into the three functions `initUIDefines()`, `initUILayout()`, and `initUIState()` become obvious; we can easily talk about the “guiding logic” of the UI, without getting into the details of the layout until this logic has been presented.

### 3.4.1 initUIDefines()

In `initUIDefines()`, we define the features and modes that will control the interface, without specifying what specific parts of the interface each will control—that task is left for `initUILayout()`. `initUIDefines()` is a relatively short function:

```
function initUIDefines() {  
    /* Define a "feature" (see the layout library documentation) called  
    "flagdemo:border". When this feature is true, the border will  
    be shown; when the feature is false, the border will be  
    invisible. */  
    beos.globals.features.Add("flagdemo:border", null, null);  
  
    /* Define a "mode cluster" (see the layout library  
    documentation) called "flagdemo:mode". The setting of this  
    mode will determine what is shown in the central area  
    of the screen. */  
    beos.globals.modes.AddCluster("flagdemo:mode");  
    /* After defining the mode, we need to define what values that  
    mode cluster can assume. In this case, the mode cluster can be one of the four  
    values "all", "red", "green", or "blue". */  
    beos.globals.modes.AddMode("flagdemo:mode", "all", null, null);  
    beos.globals.modes.AddMode("flagdemo:mode", "red", null, null);  
    beos.globals.modes.AddMode("flagdemo:mode", "green", null, null);  
    beos.globals.modes.AddMode("flagdemo:mode", "blue", null, null);  
}
```

We define one feature, “flagdemo:border”, which will control the visibility of the border area, and one mode, “flagdemo:mode”, which can take one of the four values “red”, “green”, or “blue”, “all”. These four possible values of the mode determine if the UI’s central area displays a solid primary color, or a sublayout showing all three primary colors simultaneously.

**Warning:** The mode and feature namespaces (i.e. the namespaces which in the above contain the names “flagdemo:border” and “flagdemo:mode”) are shared across all parts of a BeIA system. In order to avoid potential name clashes, you should choose names of features and mode clusters carefully. Our recommendation is to use a two-part name for each feature or mode, as was done above; “flagdemo:border” names a feature called “border” within the “flagdemo” part of the UI. Of course, if your needs are particularly complex, you could use names with yet another part, such as “mydepartment:mysection:myfeature”.

### 3.4.2 initUIState()

Let’s skip over `initUILayout` for a moment, and look at the last of the three `ui_custom.js` functions to be executed; `initUIState()`. `initUIState()` has a particularly simple task. It simply ensures that all of the features, modes, and other state variables defined in `initUIDefines()` and `initUILayout()` are set to a consistent, desired starting value. The code for our version of this function is as follows:

```
function initState() {

    /* Initialize the "flagdemo:border" feature to a known
    initial value; in this case, we choose to Close() (that is, hide)
    the border */
    beos.globals.features.Close("flagdemo:border");

    /* Ensure that "flagdemo:mode" is set to an initial
    known state; in this case, we elect to have the example start up
    with the "red" flag showing. */
    beos.globals.modes.Set("flagdemo:mode", "red");

    /* Apply above changes to all layout objects */
    beos.globals.topLayout.Refresh();
}
```

This function has just three lines of code; the first line specifies that the border area should initially be hidden, the second line specifies that the central area should initially display a solid red frame, and the third line simply causes the “top frame” of the entire interface, and everything in it, to be refreshed so as to reflect the new layout.

**Warning:** Always remember to terminate your `initUIState()` function with the line “`beos.globals.topLayout.Refresh()`”. If you do not do this, you may get anomalous behavior, such as some frames or suframes not being displayed correctly when your UI first loads.

### 3.4.3 `initUILayout()`

`initUILayout()` is by far the largest function in our `ui_custom.js` file, as is normally the case with BeIA UIs. This code isn’t particularly complex, it simply has to specify a great many details.

We won’t go through the entire function line by line. Instead, we’ll look at some specific parts of the function, to illustrate particular points about use of the layout library. These points are discussed in the subsections of this section, below. Before we get into those subsections, here is the complete code if `initUILayout()`, for your reference:

```
function initUILayout() {

    /* Define a top layout bag, and make it the top
    by setting it as the value of "beos.globals.topLayout" */
    var topLayout = beos.globals.topLayout
        = new LayoutBag(MakeObject(
            "name", "_be:topLayout",
            "orientation", LayoutBag_SPLIT_VERTICAL
        ));

    /****** CONTROLS *****/

    /* The "buttons" (actually, just HTML headings with their
```

```
ONCLICK property set to a JavaScript function) go into the left
side of the "topLayout" area */
topLayout.AddChild(new LayoutFrame(MakeObject(
    "name", "buttons",
    "location", "file://$RESOURCES/flagdemo/buttons.html",
    "size", "100",
    "visible", true,
    "scrolling", true
)));

/***** BORDER *****/

/** Define the left and right parts of the border. "midLayout", between
them, will contain everything else, including the top and bottom parts
of the border */
var leftBorder = topLayout.AddChild(new LayoutFrame(MakeObject(
    "name", "left_border",
    "location", "file://$RESOURCES/flagdemo/border.html",
    "size", "25",
    "visible", true,
    "scrolling", false
)));

var midLayout = topLayout.AddChild(new LayoutBag(MakeObject(
    "name", "midLayout",
    "orientation", LayoutBag_SPLIT_HORIZONTAL,
    "size", "*"
)));

var rightBorder = topLayout.AddChild(new LayoutFrame(MakeObject(
    "name", "right_border",
    "location", "file://$RESOURCES/flagdemo/border.html",
    "size", "25",
    "visible", true,
    "scrolling", false
)));

/* Now, define the top and bottom parts of the border. */
var topBorder = midLayout.AddChild(new LayoutFrame(MakeObject(
    "name", "top_border",
    "location", "file://$RESOURCES/flagdemo/border.html",
```

```
        "size", "25",
        "visible", true,
        "scrolling", false
    ));

    var centralLayout = midLayout.AddChild(new LayoutBag(MakeObject(
        "name", "flag",
        "orientation", LayoutBag_SPLIT_VERTICAL,
        "size", "*"
    )));

    var bottomBorder = midLayout.AddChild(new LayoutFrame(MakeObject(
        "name", "bottom_border",
        "location", "file://$RESOURCES/flagdemo/border.html",
        "size", "25",
        "visible", true,
        "scrolling", false
    )));

    /***** CENTRAL AREA *****/

    /* The centralLayout is the main display area in the screen. It will,
    at any one time, display one of a red "flag", a green "flag", a
    blue "flag", or a tricolor "flag". These are defined as "redframe",
    "greenframe", "blueframe", and "allframe". */
    var redframe = centralLayout.AddChild(new LayoutFrame(MakeObject(
        "name", "red",
        "location", "file://$RESOURCES/flagdemo/red.html",
        "size", "*",
        "visible", true
    )));

    var greenframe = centralLayout.AddChild(new LayoutFrame(MakeObject(
        "name", "green",
        "location", "file://$RESOURCES/flagdemo/green.html",
        "size", "*",
        "visible", true
    )));

    var blueframe = centralLayout.AddChild(new LayoutFrame(MakeObject(
        "name", "blue",
        "location", "file://$RESOURCES/flagdemo/blue.html",
```

```
        "size", "*",
        "visible", true
    ));

    /* Since "allframe" has multiple sections--three different subframes,
    each displaying a different background--it has to be defined as
    a LayoutBag. After it is defined, its three subframes are created
    and inserted into it. */
    var allframe = centralLayout.AddChild(new LayoutBag(MakeObject(
        "name", "allcolors",
        "orientation", LayoutBag_SPLIT_HORIZONTAL,
        "size", "*"
    )));

    allframe.AddChild(new LayoutFrame(MakeObject(
        "location", "file://$RESOURCES/flagdemo/red.html",
        "size", "*",
        "visible", true
    )));

    allframe.AddChild(new LayoutFrame(MakeObject(
        "location", "file://$RESOURCES/flagdemo/green.html",
        "size", "50%",
        "visible", true
    )));

    allframe.AddChild(new LayoutFrame(MakeObject(
        "location", "file://$RESOURCES/flagdemo/blue.html",
        "size", "*",
        "visible", true
    )));

    /* Tell each of the four parts of the border that they should appear
    when (and only when) the "flagdemo:border" feature is "true". */
    CoupleLayoutObjectToFeature(leftBorder, "flagdemo:border", true);
    CoupleLayoutObjectToFeature(rightBorder, "flagdemo:border", true);
    CoupleLayoutObjectToFeature(topBorder, "flagdemo:border", true);
    CoupleLayoutObjectToFeature(bottomBorder, "flagdemo:border", true);

    /* Link the defined modes so that they control the visibility
    of the flag frames. In this case, the redframe will be visible when
    the mode named "flagdemo:mode" has the value "red"; the greenframe
    will be visible when the "flagdemo:mode" mode has the value "green";
```



```
and so on. */
CoupleLayoutObjectToMode(redframe, "flagdemo:mode", "red");
CoupleLayoutObjectToMode(greenframe, "flagdemo:mode", "green");
CoupleLayoutObjectToMode(blueframe, "flagdemo:mode", "blue");
CoupleLayoutObjectToMode(allframe, "flagdemo:mode", "all");
}
```

### 3.4.3.1 Placing your UI in the BeIA Global Namespace

One part of the above code that is easy to overlook, but that is critical to the proper functioning of your UI, is placing the layout object defining the UI into a location where BeIA expects to find it. This is done by the lines

```
var topLayout = beos.globals.topLayout
    = new LayoutBag(MakeObject(...
```

near the top of the `initUILayout()` function. BeIA always looks in `beos.globals.topLayout` for the layout object it will display to the screen.

As a side note, it may seem that, given the above piece of knowledge, the easiest way to change from one screen in your UI to another would be to reassign the value of `beos.globals.topLayout`. If you do this, you'll be causing yourself more trouble than you need, and ignoring the convenience of the layout library. A better way to do this is to define layout objects for all of your top-level screens, place them all into a single layoutBag which goes into `beos.globals.topLayout`, and then define a mode cluster and modes to switch between the top-level screens.

### 3.4.3.2 Getting the Desired Left to Right Layout at the Top Level

Now let's take a look at the first phase of actually constructing the layout. If you refer to diagram xxx, you'll see that the top-level layout bag ("topLayout") contains four child layout objects; in left to right order, they are the buttons area, the "leftBorder" object, the "midLayout" object, and the "rightBorder" object. In order to get `topLayout` to display its children in left-to-right order (vs. top-to-bottom), we simply need specify in its constructor that it has an orientation of `LayoutBag_SPLIT_VERTICAL`:

```
var topLayout = beos.globals.topLayout
    = new LayoutBag(MakeObject(
        "name", "_be:topLayout",
        "orientation", LayoutBag_SPLIT_VERTICAL
    ));
```

Once that's done, the first object added as a child to `topLayout` will appear on its left side; the next object added will appear just to the right of the first object; and so on. If you look through the code for `initUILayout()` above, you'll find the following statements which add children to `topLayout`, in this order:

```
topLayout.AddChild(new LayoutFrame(MakeObject(
    "name", "buttons",
    "location", "file://$RESOURCES/flagdemo/buttons.html",
    "size", "100",
    "visible", true,
    "scrolling", true
)));
```

```
...
var leftBorder = topLayout.AddChild(new LayoutFrame(MakeObject(
    "name", "left_border",
    "location", "file://$RESOURCES/flagdemo/border.html",
    "size", "25",
    "visible", true,
    "scrolling", false
)));

var midLayout = topLayout.AddChild(new LayoutBag(MakeObject(
    "name", "midLayout",
    "orientation", LayoutBag_SPLIT_HORIZONTAL,
    "size", "*"
)));

var rightBorder = topLayout.AddChild(new LayoutFrame(MakeObject(
    "name", "right_border",
    "location", "file://$RESOURCES/flagdemo/border.html",
    "size", "25",
    "visible", true,
    "scrolling", false
)));
```

Once you've added an object to a layoutBag, you can't change its position in that layout bag.

The layouts for midLayout and centralLayout are constructed similarly (though with different orientations), as is the layout for the "allFrame" object, which in spite of its name, is a layoutBag rather than a layoutFrame.

Note that a layoutBag does not need to be fully initialized before being added to another layout bag. In the above code, "midLayout" is added to topLayout before midLayout has had an children of its own added. You could add the children of midLayout at any time after it is created, either before or after midLayout is placed in topLayout.

### 3.4.3.3 Controlling the Layout Elements

With the layout fully defined, only one thing remains to do in initUILayout(), to produce our full application. We need to couple the visibility of various parts of the layout to the mode and feature we defined in initUIDefines() earlier, which are in turn controlled by the controls in "buttons.html". To do this takes just a few lines of code; one for each of the four border parts, and one for each of "redframe", "greenframe", "blueframe", and "allframe". The relevant code appears right at the bottom of initUILayout, and is as follows:

```
/* Tell each of the four parts of the border that they should appear
when (and only when) the "flagdemo:border" feature is "true". */
CoupleLayoutObjectToFeature(leftBorder, "flagdemo:border", true);
CoupleLayoutObjectToFeature(rightBorder, "flagdemo:border", true);
```

```
CoupleLayoutObjectToFeature(topBorder, "flagdemo:header", true);
CoupleLayoutObjectToFeature(bottomBorder, "flagdemo:header", true);

/* Link the defined modes so that they control the visibility
of the flag frames. In this case, the redframe will be visible when
the mode named "flagdemo:mode" has the value "red"; the greenframe
will be visible when the "flagdemo:mode" mode has the value "green";
and so on. */
CoupleLayoutObjectToMode(redframe, "flagdemo:mode", "red");
CoupleLayoutObjectToMode(greenframe, "flagdemo:mode", "green");
CoupleLayoutObjectToMode(blueframe, "flagdemo:mode", "blue");
CoupleLayoutObjectToMode(allframe, "flagdemo:mode", "all");
```

As you can see from the feature coupling, a single feature (or mode) can control any number of elements in the layout. The converse is not true; if you attempt to have more than one feature or mode cluster controlling a single layout object, you may get unexpected behavior. In addition, a layout object may be linked to at most one mode in a mode cluster. For example, you might be tempted to try the following (incorrect) way of getting a “tri-color flag” defined in the central area:

```
/* EXAMPLE OF INCORRECT PROGRAMMING! */
CoupleLayoutObjectToMode(redframe, "flagdemo:mode", "red");
CoupleLayoutObjectToMode(greenframe, "flagdemo:mode", "green");
CoupleLayoutObjectToMode(blueframe, "flagdemo:mode", "blue");
CoupleLayoutObjectToMode(redframe, "flagdemo:mode", "all");
CoupleLayoutObjectToMode(greenframe, "flagdemo:mode", "all");
CoupleLayoutObjectToMode(blueframe, "flagdemo:mode", "all");
```

The intent here is to have the red subframe displayed when the mode is “red” or “all”, the green subframe displayed when the mode is “green” or “all”, and likewise for the blue subframe. Theoretically, then, when the mode is “all”, the red, green, and blue subframes would all be displayed. However, under the current implementation of the layout library, this *will not* work. Each of the red, green, and blue frames may be controlled by only one mode in “flagdemo:mode” mode cluster.



# UI Customization

# JavaScript Layout Engine

---

## 1 Introduction

The following constitutes the reference documentation for the JavaScript layout functions found in the “liblayout.js” javascript code file, included in the standard BeIA distribution. You can find detailed examples and explanations of the use of these functions in the JavaScript Layouts Concept Document, included in the optional Concepts documentation package.

### 1.1 Layout Objects: LayoutFrames and LayoutBags

A *layout object* is a JavaScript object which represents some element of a web page layout. The BeIA JavaScript layout package currently offers two types of layout objects, each defined as a JavaScript class. A `LayoutFrame` object represents a single rectangular HTML frame displaying some sort of content, such as a page of text. A `LayoutBag` object is a collection of `LayoutFrames`; it represents an HTML frameset, and displays the contents of its child `LayoutFrames` either side-by-side, or stacked one on top of the other.

By using layout objects, you can avoid the need of working directly with HTML in JavaScript. Layout objects will automatically generate all of the HTML needed to correctly display onscreen.

### 1.2 Visibility, Features and Modes

A great deal of the UI flexibility of the BeIA system depends on the ability of the layout library to quickly and easily hide or show `LayoutFrame` or `LayoutBag` objects. For example, in some versions of BeIA, a “Bookmarks” frame is always present in the browser window, but is only visible if the user requests it. This approach is more reliable and easier to program for than actually modifying the data structure of the browser window to include a bookmarks frame.

The visibility of layout objects can be set directly via their associated `setVisible()` method, but in general, you will probably find it more convenient to control visibility of objects with *features* and *modes*. See below.

Note that in order for changes to be reflected onscreen, you will need to issue a `beos.realTop.be_refresh()` command.

#### 1.2.1 Zero-Pixel Hiding

When layout objects are made invisible, the normal method of doing this is to simply avoid generating the HTML which would display them onscreen. From the point of view of the browser, it starts displaying a page of HTML which contains a reference to (for example) a frame called *A*, and then is passed a different page to display, which is the same as the first page except that all mentions of *A* have been removed. *A* ceases to exist.

However, if *A* contains JavaScript code which is used by other elements of the interface, it may be necessary to hide *A* but still maintain a reference to it in the HTML, to ensure that it (and its

accompanying JavaScript code) are loaded. This is accomplished by specifying that a layout object should be hidden using *zeroPixelHiding*, in the object's constructor. See the reference for the `LayoutBag` and `LayoutFrame` constructors for details, and see the Concepts documents for further explanation and examples.

### 1.2.2 Features

*Features* are effectively boolean variables whose state is linked to the visibility of one or more layout objects. Often a feature is used to control whether or not a particular feature (in the usual sense of the word) of the UI is visible to the user, hence the name. You can link a number of layout objects to a feature, and then simply change that feature to hide or show those layout objects. See the `CoupleLayoutObjectToFeature()`, `beos.globals.features.Add()`, and related functions for details.

### 1.2.3 Modes and Mode Clusters

A *mode cluster* is just a generalization of a feature; instead of having only two values, a mode cluster can take any one of a set of values, each of which is called a *mode*. The set of modes associated with a mode cluster is defined by you, the programmer. The mode-related part of the API is slightly different than the feature-related part, but the differences are strictly a result of the fact that a mode cluster can take one of many user-defined mode values, while a feature can take one of two predefined boolean values.

See the `CoupleLayoutObjectToMode()`, `beos.globals.modes.Add()`, and related functions for details.

## 1.3 Sizes

Various layout constructors or functions require you to specify a *size*. Generally, you can give a size in one of several ways:

- Specify a number, which denotes a size (width or height) in screen pixels.
- Specify a string of the form “*number%*”, such as “10%”; this specifies that the size should be set to the given percentage of whatever is available in the appropriate dimension (width or height, depending on the context).
- Specify the size as “\*”; this signifies that whatever space is free in the appropriate dimension should be allocated to this size request.

---

## 2 Class and Function Reference

### 2.1 Notation

JavaScript is an untyped language, in that variables, function arguments, and function return results may be of any type. However, most functions below expect arguments of a certain “type”, e.g. numbers, objects of a certain JavaScript class, or strings of a certain format. In order to provide this “type” information, the parameters in the functions below, and the return values of the functions, are “typed” using notation akin to that found in most typed language, i.e. the return type of a function (if any) is indicated by a “returns” note, and the type of a parameter is indicated by putting the type before the parameter. This is not part of the JavaScript syntax. For example,

```
Foo(char c, int n) returns string
```

indicates a function named `foo`, which takes a character as its first argument and an integer as its second argument, and returns a string. You may need to exercise a certain amount of judgment in interpreting what types mean; there is no “character” type in JavaScript, so what is actually meant by “char” is just a single-character string.

## 2.2 Layout Classes

---

### class `LayoutBag`

`new LayoutBag(Object properties)`

Constructor to create a new `LayoutBag`. The single argument is a JavaScript object which specifies the initial properties of the layout bag as a set of name/value pairs. The recognized property names, and their possible associated values, are as follows:

- **orientation**: one of global constants `Layout_Bag_SPLIT_HORIZONTAL` or `Layout_Bag_SPLIT_VERTICAL`. Defaults to `Layout_Bag_SPLIT_HORIZONTAL`. A value of `Layout_Bag_SPLIT_HORIZONTAL` means that frames in the bag will be separated by horizontal splits, and will thus be one above the other; conversely, a value of `Layout_Bag_SPLIT_VERTICAL` means the frames will be separated by vertical splits, and will thus be side by side.
- **name**: a string, the global name of this `LayoutBag`. You should provide this if you want to be able to look up your `LayoutBag` in the global display hierarchy by name (using the `FindChild()` method).
- **visible**: true if the bag (or more correctly, the HTML frameset it represents) is visible, false if invisible, defaults to true.
- **size**: initial size of this `LayoutBag`. Exactly what this size depends on the orientation of the `LayoutBag`; a `LayoutBag` whose orientation is `Layout_Bag_SPLIT_VERTICAL` will assume that it should fill as much horizontal area (width) as available, and hence will use size to set its height, and conversely for a `LayoutBag` whose orientation is `Layout_Bag_SPLIT_HORIZONTAL`. Note, however, that `LayoutBags` will do a certain amount of resizing automatically to accommodate their context, so you may find it more useful to leave `LayoutBag` sizes undefined for the most part, and instead specify the sizes of `LayoutFrames`. See “Sizes” on page 38. for notes on what constitutes a “size”.
- **framesetArgs**: If present, included verbatim in the `<FRAMESET...>` markup tag that is generated whenever this `LayoutBag` object is rendered to HTML. For example, if you specified the `framesetArgs` property as the string `“ONLOAD=“foo()”`, then the frameset tag for this `LayoutBag` would be rendered as `<FRAMESET ONLOAD=“foo()”...>`. Use this property if you are comfortable with HTML frameset markup, and want to accomplish something that can’t be done through any of the other `LayoutBag` properties.
- **zeroPixelHiding**: true or false, depending on the hide method being used. See “Visibility, Features and Modes” on page 37.

`AddChild(LayoutObject child)` returns *child*

Add a new layout object (i.e. a `LayoutFrame` or a `LayoutBag`) as a child of this `LayoutBag`. The order in which children are added determines their position onscreen; children which are added first will be displayed at the top or left of the `LayoutBag`’s frameset (depending on the orientation of the `LayoutBag`), children added later will be displayed lower or to the right.

This function returns the argument `child` as a convenience so that you can write code like

```
var theNewFrame = someLayoutBag.AddChild(new LayoutFrame(...));
```

and then go on to set properties of the newly created frame, such as mode or feature coupling.

`FindChild(string name)` returns layoutobject

Look up by name a layout object (LayoutBag or LayoutFrame) contained in this LayoutBag and return it. Returns null if no such object is found.

`RemoveChild(string name)`

Remove the child layout object with the given name.

`SetSize(size s)`

Set the size of this LayoutBag. (The initial size can also be specified via the size property of the argument to the constructor.) See “Sizes” on page 38.

`GetSize()` returns size

Retrieve the size of this LayoutBag. See “Sizes” on page 38.

`SetVisible(boolean visibility)`

Set the visibility of this LayoutBag. Note that features and modes often provide a more convenient way of managing visibility: See “Visibility, Features and Modes” on page 37. Also note that if a LayoutBag is coupled to a feature or mode, setting the visibility directly may not work as desired.

`IsVisible()` returns boolean

Returns the visibility setting of this LayoutBag.

`DeleteAllChildren()`

Remove all child layout objects of this LayoutBag.

---

## class LayoutFrame

new `LayoutFrame`(Object *properties*)

Constructor to create a new LayoutFrame. The single argument is a JavaScript object which specifies the initial properties of the layout bag as a set of name/value pairs. The recognized property names, and their possible associated values, are as follows:

- `location`: The URL this frame displays, e.g. “file:///RESOURCES/picture.html”. [See xxx for notes on using URLs to reference internal pages in BeIA.](#)
- `name`: The name of this frame, used by various functions to find or perform operations by name.
- `size`: Initial size of the frame. The dimension this size refers to depends upon the orientation of the enclosing LayoutBag. If the orientation is such that this LayoutFrame is stacked vertically (i.e. above and/or below its sibling frames, meaning that the enclosing LayoutBag has an orientation of `Layout_Bag_SPLIT_HORIZONTAL`), then this property will set the height of the frame; in the other orientation, this property will set the width of the frame.
- `visible`: true if this frame is visible, false otherwise, defaults to true.
- `scrolling`: true if this frame should add scrollbars when the content is too large to display in the allocated screen area, false if the frame should never display scrollbars. Defaults to true.
- `frameArgs`: If present, the string passed in via this property is inserted verbatim into the <FRAME> HTML tag. This is similar to the `framesetArgs` property of LayoutBag, see the notes for that property.



- `zeroPixelHiding`: true if this frame should be hidden using *zeroPixelHiding*, false if it should be hidden using the “normal” method of hiding. See “Visibility, Features and Modes” on page 37 for a discussion of the implications of this.

`MakeLayoutFrame`(string *name*, URL location, boolean scrolling, size size, boolean visible) returns `LayoutFrame`

This function is a quick way of making layout frames, until the JavaScript interpreter in the Opera browser supports JavaScript 1.3 anonymous object creation (i.e. objects specified using a “{name:value,...}” syntax. The current version of Opera does not support this.) This function simply creates an object with appropriately named attributes, fills in those attributes with the given arguments, and then calls `new LayoutFrame()` on the object and returns the new layout frame. Arguments are:

- `name`: the name of the layout frame, passed in as the name attribute of the object given to `new LayoutFrame()`.
- `location`: the initial URL the layout frame will display, passed in as the location attribute of the object given to `new LayoutFrame()`.
- `scrolling`: determines if scrollbars will be shown in the new layout frame, passed in as the scrolling attribute of the object given to `new LayoutFrame()`.
- `size`: determines the initial size of the new frame, passed in as the size attribute of the object given to `new LayoutFrame()`.
- `visible`: determines if the new frame will initially be visible or invisible, passed in as the visible attribute of the object given to `new LayoutFrame()`.

For more details on these arguments, refer to the documentation for the associated attribute in the `LayoutFrame` constructor.

`SetSize`(sizeString *s*)

Set the size (width or height, depending on orientation of the enclosing `LayoutBag`) of this frame. See “Sizes” on page 38 for an explanation of what constitutes a legal size string.

`GetSize()` returns sizeString

Get the size (width or height, depending on orientation of the enclosing `LayoutBag`) of this frame. See “Sizes” on page 38 for an explanation of what constitutes a legal size string.

`SetVisible`(boolean *visibility*)

Set the visibility of this frame; true for visible, false for hidden. You do not normally work with layout object visibility directly: See “Visibility, Features and Modes” on page 37.

`IsVisible()` returns boolean

Returns the visibility of this frame; true if visible, false if hidden. Note that `IsVisible()` may return true but the frame might still be hidden on screen, if one of its ancestor `LayoutBags` is hidden. You do not normally work with layout object visibility directly: See “Visibility, Features and Modes” on page 37.

`SetLocation`(URLString *location*)

Set the location of this frame (the URL-formatted string it displays).

`GetLocation()` returns URL

Get the URL this frame is currently displaying.

## 2.3 Global Layout-Related Functions

---

### Feature-Related API Functions

```
beos.globals.features.Add(string featureName, function activationFunction, function  
                           deactivationFunction)
```

Create a new feature, with the given name, activation function (may be null), and deactivation function (may be null.) After creating a feature and associating it (via `CoupleLayoutObjectToFeature()`) with one or more layout objects, you should call either the `beos.globals.features.Open()` or `beos.globals.features.Close()` function on that feature's name, to assume the feature and its dependencies are in a consistent state.

```
CoupleLayoutObjectToFeature(LayoutObject obj, string featureName, boolean visibleState)
```

Set dependencies so that the given layout object (either a `LayoutFrame` or a `LayoutBag`) will be visible only when the named feature is in the given *visibleState*. After creating the given feature and using this function to create dependencies, you should call `beos.globals.features.Open()` or `beos.globals.modes.Close()` with the feature name, to ensure your data structures are in a consistent state.

```
beos.globals.features.Open(string featureName)
```

Set the boolean value associated with the named feature to true. This will cause the activation function of the feature (if any) to be called. The activation function should normally return true; if it returns false, the `Open()` will be aborted, and the value of the feature will remain what it previously was.

**Note:** Calling `Open()` on a feature that already happens to be set to true will result in an execution of the feature's activation function.

```
beos.globals.features.Close(string featureName)
```

Set the boolean value associated with the named feature to false. This will cause the deactivation function of the feature (if any) to be called. The deactivation function should normally return true; if it returns false, the `Close()` will be aborted, and the value of the feature will remain what it previously was.

**Note:** Calling `Close()` on a feature that already happens to be set to false will result in an execution of the feature's deactivation function.

```
beos.globals.features.Get(string featureName) returns boolean
```

Return the boolean value currently associated with the named feature.

---

### Mode API Functions

```
beos.globals.modes.AddCluster(string clusterName)
```

Create a new mode cluster under the given name. You will then need to use `beos.globals.modes.AddMode()` to define mode values which this new mode cluster may assume.

```
beos.globals.modes.AddMode(string clusterName, string modeName, function  
                           activationHook, function deactivationHook)
```

Define *modeName* as one of the mode values that the mode cluster named by *clusterName* can assume. The *activationHook* and *deactivationHook* functions may be null; if one or both are provided, they will be called immediately before mode changes. (The activation hook for a mode will be called immediately

before the associated mode cluster is set to that mode; the deactivation hook for a mode will be called immediately before the associated mode cluster is set to some *other* mode.) *activationHook* and *deactivationHook* should normally return true; they can return false to indicate that the mode change should not be carried out.

```
CoupleLayoutObjectToMode(LayoutObject obj, string clusterName, string modeName)
```

Set dependencies so that the given layout object (either a `LayoutFrame` or a `LayoutBag`) will be visible only when the named mode cluster has the mode value indicated by *modeName*. After creating the mode cluster and associated modes and using this function to create dependencies, you should call `beos.globals.modes.Set()` with this cluster name and an appropriate mode name, to ensure your data structures are in a consistent state.

```
beos.globals.modes.Set(string clusterName, string modeName)
```

Change the mode associated with the named mode cluster to the named mode. This will cause the deactivation function of the previous mode (if any) to be called, and then the activation function of the new mode (if any) to be called, before the cluster's mode value is actually changed. If either of those functions returns false, the mode change will be aborted.

**Note:** Setting a mode cluster to the mode it already happens to be in will still cause that mode's activation/deactivation functions to be called.

```
beos.globals.modes.Get(string clusterName) returns string
```

Return the name of the mode the named cluster is currently set to.

---

## Miscellaneous Related Functions

```
beos.realTop.be_refresh()
```

This function needs to be called so that any changes made in the displayed frames or framesets will actually be drawn to the screen. For example, you can do any number of mode or feature changes, URL changes, or calls to `setVisible()` methods, but until you explicitly issue a `beos.realTop.be_refresh()` command, none of those changes will be reflected on the screen. This allows you to make many changes but perform only one (computationally expensive) redraw.



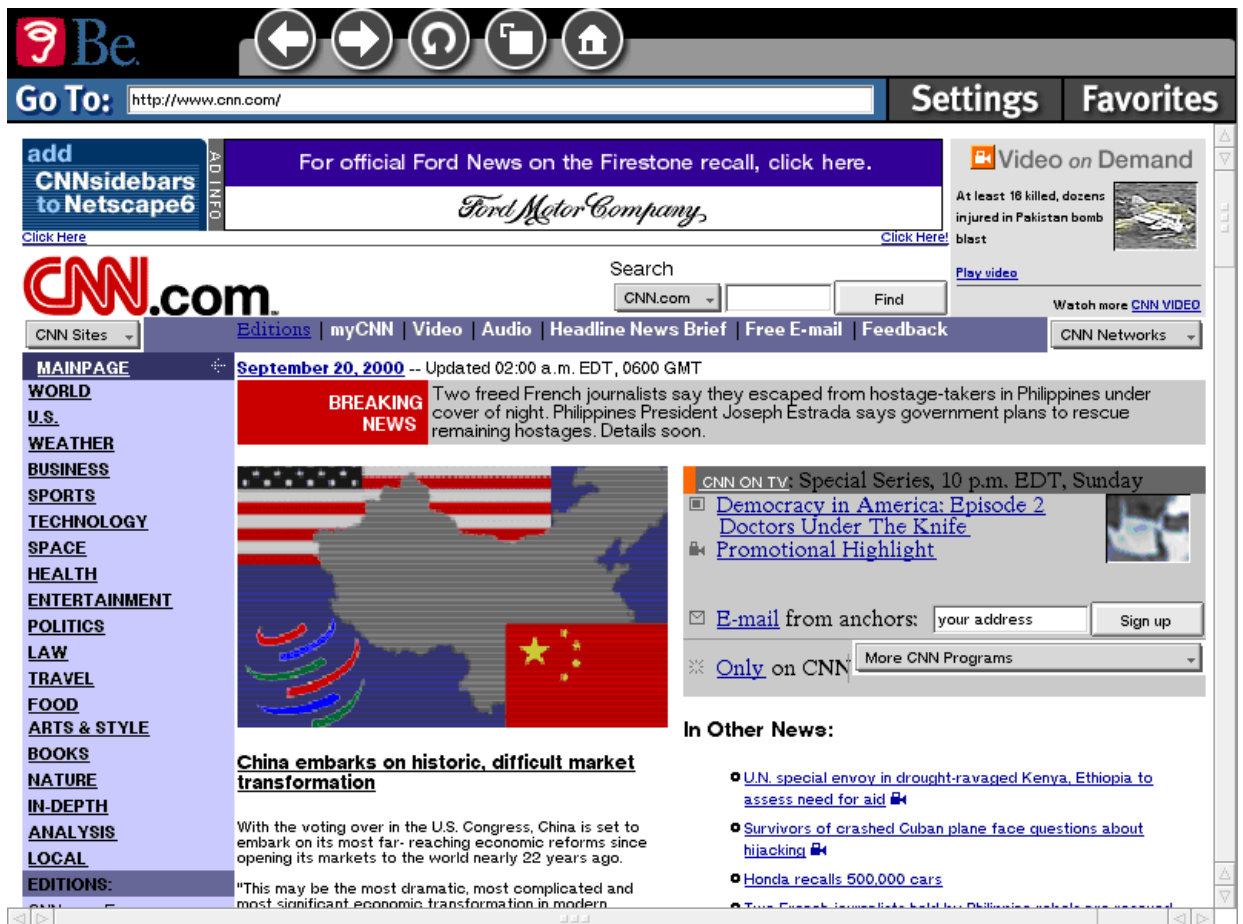
# UI Customization

## The Toolbar

The toolbar (or navigation bar) of the BeIA interface is the bar which appears at the bottom of the BeIA screen, and lets the user control the browser. A typical toolbar might appear as shown below:



Within the BeIA interface, the toolbar is always present in the screen, so that the user always has onscreen controls available for navigating the web:



You can modify aspects of the toolbar in various ways, as described in the sections below.

---

## 1 Changing the Appearance of the Toolbar

The appearance of the toolbar may be changed in one of two general ways:

- You can use different images for the toolbar controls. This will affect not only the way buttons are “normally” displayed, but also the way they are highlighted.
- You can edit `$RESOURCES/Toolbar/toolbar.html` to change the layout of the toolbar, and the appearance of those parts of it not defined by images.

### 1.1 Changing the Appearance of Toolbar Buttons

**Note:** See [xxx](#) for a general discussion of changing button appearances in BeIA.

Like other parts of the BeIA interface, the appearance of the toolbar buttons is defined by images associated with each button. In the case of the toolbar, these images are stored in `$RESOURCES/Toolbar/images` (see [xxx](#) for details of how the `$RESOURCES` environment variable is resolved to a file path), under the following filenames.

- `home.gif`, `home-over.gif`, `home-down.gif`
- `forward.gif`, `forward-over.gif`, `forward-down.gif`
- `back.gif`, `back-over.gif`, `back-down.gif`
- `goto.gif`, `goto_active.gif`, `goto_active-over.gif`, `goto_active-down.gif`
- `reload.gif`, `reload-over.gif`, `reload-down.gif`
- `magnify.gif`, `magnify-over.gif`, `magnify-down.gif`
- `stop.gif`, `stop-over.gif`, `stop-down.gif`
- `favorites.gif`, `favorites_active.gif`, `favorites_active-down.gif`, `favorites_active-over.gif`

The alternative versions of each button (eg., the `-over`, and `-down` versions of the `forward` button) define the appearance of the button when the mouse cursor is over the button, and when the button is being clicked. For button which have an explicit disabled state, such as the `goto` button, the `_active` part of the image name denotes an image which may be used when the button is clickable, while the base name (such as `goto.gif`) denotes the button in its disabled state. In functional terms, the `home`, `forward`, `back`, `reload` and `stop` buttons are the standard controls found on any browser, clicking on the `magnify` button switches the content frame between various levels of magnification, clicking on the `favorites` button toggles the bookmark display open or closed, and clicking the `goto` button causes the browser to go to whatever page is entered in the URL text box shown in the toolbar. The `status` images simply provide feedback as to the connection status of the device.

While these image files are currently GIF images, they may be converted in the future to PNG images, in which case the file names will be of the form `imagename.png`.

**Note:** The dimensions of the button images are hardcoded in the HTML layout code within `$RESOURCES/Toolbar/toolbar.html`, using the [HEIGHT](#) and [WIDTH](#) attributes of the [IMG](#) markup tag. If you replace the default button images with other images of differing dimensions, you will need to change the corresponding [HEIGHT](#) and [WIDTH](#) values within the HTML markup. See [section 1.2](#) for details of the `toolbar.html` file.

## 1.2 Changing Toolbar Layout and Other Appearance Properties with the “toolbar.html” File

The basic layout of the toolbar, along with some other properties such as background color, are determined by the HTML code in `$RESOURCES/Toolbar/toolbar.html`. The following are the changes you are most likely to wish to make.

- The toolbar elements are laid out in an HTML table, and by editing this table, you can rearrange the order of the elements, or change the layout in more significant ways.
- The entry box for user-typed URLs is just a standard HTML text input field, and you can use any tags applicable to this field to change its properties.
- The background color of the toolbar is set by the `BGCOLOR` attribute of the `BODY` markup tag. You can change this to alter the color of the toolbar behind the buttons.

**Note:** Button images are, of necessity, square, and the default button images achieve the appearance of round buttons by filling in the corners of the images with the background color. Thus, if you wish to change the background color used with these buttons, you will need to alter the button images also.

Refer to any recent HTML reference for details on editing the above HTML elements. Aside from the above points, there is little in the way of “simple” changes you can make within the toolbar—functional changes (altering the behavior of toolbar controls, or adding new controls) requires an understanding of JavaScript, and sometimes of the BeIA “plugin architecture” which supports the browser.





# UI Customization

## Bookmarks

The bookmarks interface is the list of folders and bookmarks, and associated behavior, that is presented to the user when he or she wishes to add, delete, rename, or otherwise manage web site bookmarks.

You can do a number of things to modify aspects of this interface:

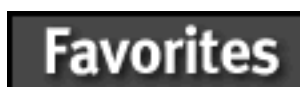
- The appearance of controls can be altered by providing alternate images for them; see **2.1** Changing the Appearance of Bookmark Controls
- Colors and fonts used in the bookmarks interface can be changed by editing the HTML file defining the general layout of the interface. See section **2.3** Changing Bookmark Layout and Other Appearance Properties with the “index.html” File.
- A predefined list of bookmarks and bookmark folder can be provided for the user. In addition, you can mark particularly important bookmarks (such as bookmarks back to your company’s technical support site) as non-deletable. See section **3** Preconfiguring the Favorites List.

These topics are discussed in the following sections. We begin with an overview of how the bookmark interface functions.

---

## 1 How the Bookmarks Interface Works

The bookmarks interface is displayed at the request of the user, by clicking on the “Favorites” button displayed in the BeIA toolbar:



When such an event occurs, the browser is presented with an internal HTML page which displays the bookmarks interface within a frame. The presentation of the bookmarks interface within this frame is defined by the `$RESOURCES/Bookmarks/index.html` file.

The bookmarks interface consists of several buttons, and a list of bookmarks and bookmark folders. The buttons are implemented using HTML `IMG` tags and event attributes which cause appropriate JavaScript calls to be invoked in response to mouseovers or clicks on the buttons. The list of bookmarks and folders is not implemented by HTML/JavaScript, but by a BeIA-provided C++ plugin, which allows editing of names and URLs, selection of parent folders via a popup menu, and a few other things. You cannot modify the C++ code for this plugin; however, the plugin does respect various attributes of the HTML `<EMBED>` tag which embeds the plugin into the bookmark interface, and editing these attributes will allow you to change the appearance of the editing list.

## 2 Changing the Appearance of the Bookmarks Interface

The appearance of the bookmark interface may be changed in one of two general ways:

- You can use different images for the bookmark interface controls. This will affect not only the way buttons are “normally” displayed, but also the way they are highlighted.
- You can edit `$RESOURCES/Bookmarks/index.html` to change the layout of the bookmarks interface, and the appearance of those parts of it not defined by images.

### 2.1 Changing the Appearance of Bookmark Controls

Like most parts of the BeIA user interface, the appearance of the bookmarks interface can easily be changed by providing a different set of graphical images for use as the controls of the interface. Although some of the bookmark controls are not standard buttons, most still use graphics to change their appearance in response to user actions, and so their appearance can be changed by substituting the desired graphics images, as with standard buttons. The bookmark control images are all located in `$RESOURCES/Bookmarks/index.html`.

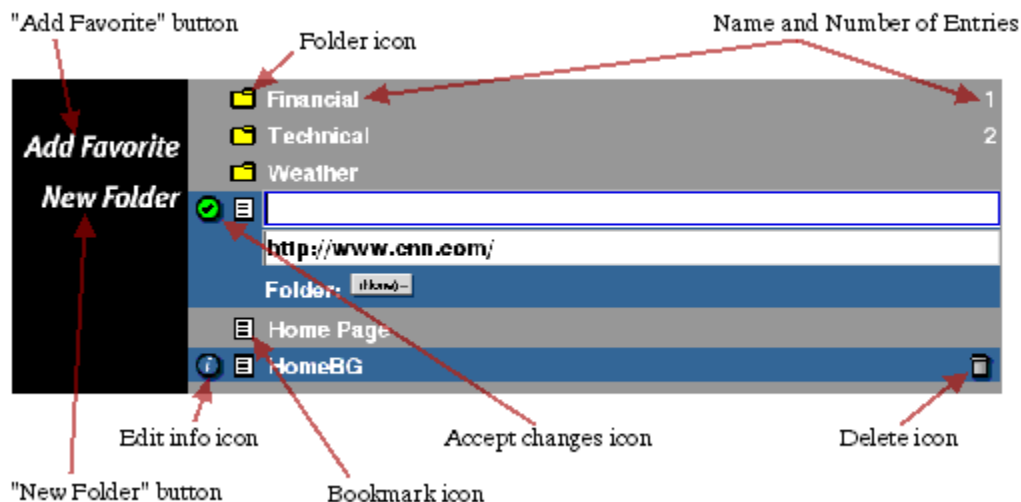
**Note:** For a description of how to change the appearance of standard buttons, see [xxx](#).

#### 2.1.1 Names and Locations of the Bookmark Controls Image Files

The appearance of virtually all of the bookmark-related controls can be changed in this manner described above. Below is a diagram assigning names to the different controls; following it is a list of the image files associated with each control. Simply replace the images with ones of your own design, to change the appearance of a given control.

Except where noted, all of the image files mentioned will be found in `$RESOURCES/Bookmarks`.

**Note:** Some of these images may be in PNG (Portable Network Graphics) format; if this is the case, you must provide any substitute images in this same format. If your graphics program does not save in PNG format, you can use BeOS (desktop version) to convert from any supported graphics format (a BeOS translator is available for GIF images, though it does not ship with BeOS) to PNG format, which is supported by a built-in BeOS translator.



- **"Add Favorite" button:** `addbookmark.gif`, `addbookmark_over.gif`, `addbookmark_active.gif`

- **“New Folder” button:** `addfolder.gif`, `addfolder_over.gif`, `addfolder_active.gif`
- **Folder icon:** `folder.png`, `folder_closed.png`
- **Edit Info icon:** `info.png`, `info_over.png`, `info_on.png`, `info_onover.png`, `info_outside.png`
- **Bookmark icon:** `bookmark.png`
- **Delete icon:** `delete.png`, `delete_over.png`, `delete_outside.png`
- **Accept changes icon:** this is just one of the image alternatives for the **Edit Info icon**.
- **Name and Number of Entries:** The appearance of these text elements depends on which font is used by the C++ bookmark plugin. This font can be changed by editing the `listfont` attribute of the `<EMBED>` definition in the `$RESOURCES/Bookmarks/index.html` file. See section **2.3**

**Note:** If you change the appearance of the bookmarks elements, you may also want to change the appearance of the toolbar button which toggles the bookmarks open and closed. See [xxx](#) for information on changing the appearance of toolbar elements.

## 2.2 How to Change the Look of Buttons—Advanced

The current “look behavior” of buttons (i.e. the use of separate images for normal, mouse-over, and pressed states) is determined by the JavaScript code found in `$RESOURCES/scripts/buttons.js`. If you wished to achieve more advanced visual effects with buttons—such as, for example, having highlighting fade in and out, rather than simply appearing and disappearing when the mouse cursor moves over a button—you could conceivably modify this code.

## 2.3 Changing Bookmark Layout and Other Appearance Properties with the “index.html” File

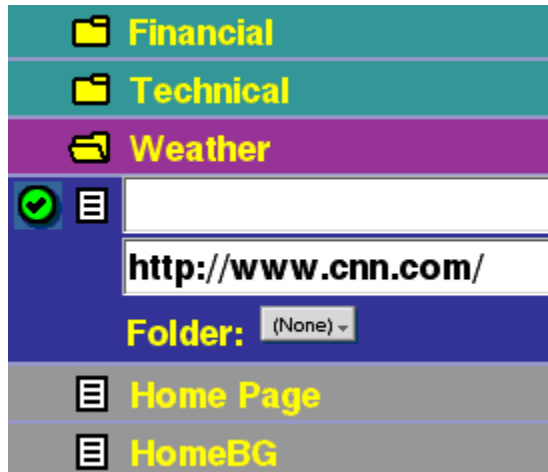
The basic layout of the favorites list, along with some other properties such as background color, highlight color, text font, and so forth, are determined by the HTML code in `$RESOURCES/Bookmarks/index.html`. Through judicious editing of this file, you can change certain basic visual aspects of the appearance of the favorites list.

**Note:** `$RESOURCES/Bookmarks/index.html` also contains a significant amount of code which affects the behavior of bookmarks; this is primarily JavaScript code, and event tags associated with control elements, which define JavaScript functions to be called when certain events take place. You should not modify such code.

There are three places within the `index.html` file you might want to make changes:

- The `<body...>` markup tag which starts the body of the HTML code defines the default background color attribute for the bookmarks list and controls. You will want to change this attribute if you elect to change the color scheme. Various other attributes associated with `<body>` in the HTML standard may also be applied.
- A `<table...>` definition defines the layout of the bookmark buttons at the left of the favorites list. (The “Add Favorite” button and the “New Folder” button: See section **2.1.1** ) You can change this table to change the layout of the buttons. In addition, if you use your own set of graphics for the button images (See section **2.1** ), and these graphics differ in size from the default button graphics, you should alter the `WIDTH` and `HEIGHT` attribute values of the corresponding `IMG` markup tags within the table, to reflect this difference.
- The `<EMBED...>` definition which constitutes approximately the last one-third of `index.html` is used to define the appearance of the C++ plugin which provides the bookmark list and list management interface. The plugin uses attributes associated with the `<EMBED>` tag to set many of its visual properties. Under the current version of BeIA, all of these attributes control various colors in the favorites list, except for the `listfont` attribute, which defines the font used in the favorites list. [The `buttonfont` attribute is nonfunctional and will at some point be removed.]

The various colors controlled by the <EMBED> attributes should be reasonably apparent from their names. If in doubt, simply play around with the hexadecimal color attribute values, restart your browser (to clear the cache), and go to the favorites screen to view the result. Here is the aesthetically disastrous result I got by changing some values at random:



---

### 3 Preconfiguring the Favorites List

You may wish to ship your BeIA units with a set of bookmarks and folders preconfigured for the convenience of the user. All bookmark data resides in the file `/boot/home/config/settings/Favorites`. You can generate or modify this file in one of two ways:

- Simply browse to the desired sites on a working BeIA device, using the BeIA interface to create and add folders and bookmarks as desired. Then copy the resulting `Favorites` data file into your distribution filesystem.
- Edit the `Favorites` data file by hand. This is more error-prone (make sure to check, after doing this, that the bookmarks you've entered work properly from the BeIA interface), but it does let you do some things which cannot be done by simply using the BeIA interface to generate the `Favorites` file. The format of the `Favorites` file is discussed below.

#### 3.1 “Favorites” File Format

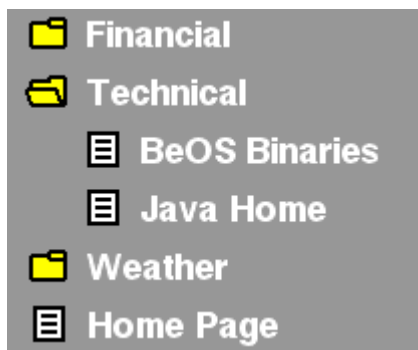
Before going into the details of the `Favorites` file format, you should understand a few of the assumptions made by the BeIA favorites interface:

- Folders will never be nested within folders; “folders of favorites” can only appear at the top level of the favorites list, and can only contain bookmarks.
- A user's list of favorites will be presented as an alphabetized list of folders, followed by an alphabetized list of top-level bookmarks. Within each folder, all of the bookmarks in that folder will be shown alphabetically. This ordering is defined in the C++ bookmark plugin (see See section 1 ), and cannot be changed except by changing the plugin, or using a different plugin. As a result, the order of entries in the `Favorites` file is not necessarily the order in which they will appear to the user.

With that in mind, here is a sample `Favorites` file:

```
@Favorites 1
Folder Financial
    Favorite Yahoo
    URL=http://www.yahoo.com
Folder Technical
    Favorite BeOS+Binaries
    URL=http://www.bebits.com/
    Favorite Java+Home
    URL=http://java.sun.com/
    NoDelete=true
Folder Weather
    Favorite Home+Page
    URL=file://$RESOURCES/Home/start.html
```

and here is the the resulting favorites list:



Even if the user passes the cursor over the “Java Home” entry, no trash can symbol will be shown in that entry, and so the user cannot delete “Java Home”. This is the result of the occurrence of the tag `NoDelete=true` in the Favorites file, under the “Java Home” entry. This will be discussed in more detail shortly.

These are the rules for the Favorites file format:

- The Favorites file begins with the line “@Favorites 1”.
- The remainder of the Favorites file is a list of favorite entries. Each such entry may define either a folder or a bookmark.
- If an entry defines a folder, it will consist of a single line, starting with the word “Folder”, and with the remainder of the line giving the name the user will see associated with the folder. The bookmark entries in the folder follow immediately after, and all of the lines which make up the folder’s bookmarks begin with a `<tab>` character.
- If an entry defines a bookmark, it consists of multiple lines. The first line of a bookmark entry starts with the word “Favorite” (including a leading `<tab>`, if the bookmark is contained within a folder), followed by the name the user will see for the bookmark. The next line is of the form “URL=location”, where location is a fully defined web address (including the correct protocol identifier at the beginning, such as “http:”, “ftp:”, etc.) Any further lines are optional, and set additional properties for the bookmark. At this time, the only additional option is a line of the form “NoDelete=true”, which will prevent the user from removing the bookmark.
- The names associated with folders or bookmarks must be given in an “HTML-escaped” format, i.e. as if they were text to be displayed in an HTML document. This means that “+” should be used in place of spaces in the names, and that various other characters associated with HTML markup must be given via an HTML escape sequence, rather than entered directly in the name. See any introductory HTML book for details.

- Any lines in the Favorites file which are associated with a top-level folder or bookmark should be entered with no leading whitespace. Any lines associated with a bookmark contained in a folder should be entered with a single leading <tab> character.
- The last line of the Favorites file should end with a linefeed, otherwise it may not be read in correctly by the BeIA system.

# UI Customization

## Buttons

Almost all buttons in BeIA have an appearance which is provided by a pre-rendered image or images. By substituting your own button images in place of the default images provided in BeIA, you can change the appearance of the interface, without writing a single line of code. This section discusses the details of this process.

---

### 1 An Example of Button Images in Action

To see how button images work in the actual BeIA interface, consider the example of the “forward” button in the toolbar. Here’s a screenshot showing the **forward** button (and a bit of the surrounding buttons, for context) in its “normal” state:



This screen image is produced by drawing a dark blue background in the HTML frame the **forward** button will appear in, and then, in the position of the **forward** button, drawing the image found in the BeIA file `/boot/custom/Resources/en/Toolbar/images/forward.gif`. As it happens, the image found in that GIF file is just this:



When the mouse cursor moves over the position of the **forward** button, a slightly different image is used to draw the button on the screen; this image is found in `/boot/custom/Resources/en/Toolbar/images/forward-over.gif`, and looks like this:



On screen, this produces the appearance that whenever the mouse moves onto the button, the green circle around the button turns red. If the user then clicks the mouse button, a third image, `/boot/custom/Resources/en/Toolbar/images/forward-down.gif`, is used for as long as the user holds down the mouse button. In the case of the **forward** button, it looks exactly the same as the `forward-over.gif` image, but it is a separate image and could be different:



Together, these three files constitute define the appearance of the **forward** button. By replacing them with images of your own design, you can change the appearance of the forward button in almost any way you wish.

---

## 2 The General Button Image Schema

### 2.1 Location of Button Image Files

In the example above, the four button images were `forward.gif`, `forward-over.gif`, and `forward-active.gif`, all in the folder `/boot/custom/Resources/en/Toolbar/images`. In general, the images for the buttons you might be interested in will be in different folders depending on what part of the interface the buttons are associated with. So, the images for all of the buttons associated with the toolbar are in `$RESOURCES/Toolbar/images`, while the images for the bookmarks list buttons are in `$RESOURCES/Bookmarks`. `$RESOURCES` is an environment variable which expands out into the proper path for resources in a given locale—see [xxx](#) for details. For full descriptions of button image locations and names, see the BeIA technical documentation describing the relevant part of the BeIA interface.

### 2.2 Button Image File Formats and Suffixes

Button images may be either GIF images (with a `.gif` suffix) or PNG images (with a `.png` suffix). Most buttons used in BeIA use GIF images, but in the future, it is expected that BeIA will use only PNG button images, due to the advantages associated with this format. See the section below on PNG images for details. If you replace BeIA button images with images of your own design, you should replace PNG images with PNG images, and GIF images with GIF images.

### 2.3 Button Image Names

For a given button, the names of the images for that button generally follow the same pattern. If the button is named `button`, then the following image files, or files with similar names, may be present:

- `button.gif` (or `.png`): The “normal” button image.
- `button-over.gif` (or `.png`): The image used when the mouse is over the button.
- `button-active.gif` (or `.png`): The image used when the button is being pressed.

If the button has a disabled state, a different naming scheme may apply:

- `button.gif` (or `.png`): The “disabled” button image.
- `button_active.gif` (or `.png`): The button in its enabled (clickable) state.
- `button_active-over.gif` (or `.png`): The image used when the mouse is over the button.
- `button_active-down.gif` (or `.png`): The image used when the button is being pressed.

For historical and technical reasons, slightly different naming schemes may apply to various buttons. When in doubt, simply inspect the button image, and check on a running IAD device to find out when that button image is used (i.e. when the mouse is over the button, clicking the button, etc.)



You need only replace files that exist, to change the appearance of a button; you do not have to supply images (such as a “disabled” image) that are not in the default BeIA distribution.

---

### 3 Button Sizes

For efficiency reasons when drawing onscreen, the sizes of the BeIA buttons are all “hard-coded” into the HTML which defines the user interface elements the user sees. When using other images for buttons, we suggest you use images that are exactly the same dimensions (pixel width by pixel height) as the button images you are replacing. If you do this, you will not need to alter any HTML code.

If you decide to use button images that differ in dimension from the default BeIA buttons, you will need to alter the button size in the HTML code defining that part of the interface. The technical documentation describing the details of each part of the interface will direct you to the appropriate HTML file or files. As an example, though, consider this snippet of code from the HTML file which defines the layout of the toolbar (depending on the version of BeIA you are working with, the toolbar HTML file may not contain exactly this text, but it will have something similar):

```
<TABLE BORDER=0 WIDTH=800 CELLPADDING=0 CELLSPACING=0>
  <TR>
    <TD WIDTH=110>
      <IMG onClick="top.be_goto_home();" SRC="logo.gif" HEIGHT=57 WIDTH=110 BORDER=0>
    </TD>
    <TD WIDTH=12 HEIGHT=49 NOWRAP VALIGN=TOP></TD>
    <TD WIDTH=41><!--Back Button-->
      <IMG NAME="back" SRC="back.gif" WIDTH=41 HEIGHT=41 BORDER=0
        ONMOUSEOVER="doOverSound('back');"
        ONMOUSEOUT="doOut('back');"
        ONMOUSEDOWN="doDown('back');"
        ONMOUSEUP="if(doUp('back')) top.be_back();">
    </TD>
    <TD WIDTH=12 HEIGHT=49 NOWRAP VALIGN=TOP></TD>
    <TD WIDTH=41><!--Forward Button-->
      <IMG NAME="forward" SRC="forward.gif" WIDTH=41 HEIGHT=41 BORDER=0
        ONMOUSEOVER="doOverSound('forward');"
        ONMOUSEOUT="doOut('forward');"
        ONMOUSEDOWN="doDown('forward');"
        ONMOUSEUP="if(doUp('forward')) top.be_forward();">
    </TD>
  </TR>
</TABLE>
```

This HTML code defines part of the table which makes up the toolbar (each button is in its own table cell, and cells are also used for spacing.) The definitions of three buttons—the **home**, **back**, and **forward** buttons—are shown. In each case, the sizes of the associated button images are given by the values of the **WIDTH** and **HEIGHT** attributes of the **IMG** markup tags. If you change the sizes of the button images, you must also change the values associated with these attributes. Note that for any given button, such as the **forward** button, the sizes of all images for that button (i.e. the “normal” image, the -over image, etc.) must be the same.

**Note:** Depending on the part of the interface you are working with, using differently sized buttons may require more complex changes than this, due to the fact that if button images grow too large, you may need to change their layout on the screen so the user can see everything. Making buttons smaller is less likely to cause problems than making them larger.

---

### 4 PNG vs. GIF Button Images

Although most of the button images used in BeIA are currently in the GIF image format, it is expected that in the future, Be, Inc. will shift to using PNG images for buttons throughout BeIA, due to the alpha transparency ability of the PNG format. This section briefly discusses this feature.

Consider again the default button image for the toolbar's forward button:



We'll assume that the intended interpretation is various shades of gray over a white background. Note that the background is actually part of the image; this means that the background of the button must be coordinated with the background color of the surrounding HTML frame. In particular, this means that simply changing the background color of the toolbar necessitates not only a change to the HTML code defining the toolbar's default background color, but also requires redoing all of the button images so that they use an appropriate background color. This is because GIF images do not support transparency. (More correctly, GIF images do not support transparency sufficiently well to produce a good appearance under most circumstances—pixels in a GIF image cannot be partially transparent, and so cannot be antialiased smoothly against the background color).

However, the PNG format does support full alpha transparency. Using PNG format, the above button image could be defined in such a way that what appears white could be defined as transparent, with the edges of the gray areas defined as partially transparent to achieve a smooth, antialiased appearance regardless of the background. The white background would not appear in this PNG image, and you could easily change the background of your HTML frame without changing any of the button images.

**Note:** To produce button images with transparency, you will need to use an image editor or paint program which supports transparency and which can save transparency information to PNG files.

# UI Customization

## Alerts

Alerts and dialog panels are HTML pages that are created on-the-fly by a cgi-bin script (the **alert script**). The script creates the new HTML page by merging an **alert template** (a “text-less” HTML file) with some **alert content** (the text that will appear on the page).

You can’t customize the **alert script**, but you can provide your own **alert templates** and **alert content**. In the sections below, we look at the format of the alert template and alert content, and describe how to invoke the **alert script**.

This document considers “alerts” and “dialog panels” to be the same thing, and refers to all such entities as “alerts.”

## 1 The Alert Template

An **alert template** is an HTML file that’s used as the basis for an alert that’s displayed by the browser. You can supply as many different templates as you want, and they can be as explicit (i.e. as hard-coded) as you want.

Typically, an alert template defines the form of the alert—how many buttons, where the alert content is positioned, how the reply is handled—without hard-coding any actual text. The alert content (i.e. the text that the alert displays) is supplied as “arguments” to the template; a template accepts as many as ten text arguments. These arguments are referred to, within the template, by the variables `%0` through `%9`. (How you supply the arguments is discussed under “**3 Invoking the Alert Script**,” below.)

### 1.1 A One Button Alert

As a simple example, here’s a template that displays a centered line of text and a titled “button” (i.e. a link):

```
content-type: text/html
<TITLE></TITLE>
<BODY BGCOLOR=0000ff TEXT=ffffff LINK=ff0000 VLINK=ff0000 ALINK=ffbfbf>
<P ALIGN=CENTER>%0<BR>
<A HREF=
  "file://$SCRIPTS/htmlalertreply?i32-msgid=0x00000000&i32-
  realwhat=0x00000000">%1</A>
```

The template expects two content arguments, which it refers to as `%0` and `%1`. When passed the arguments “File not found.” and “Okay”, for example, the constructed alert would look something like this:

<<illo>>

The file reference in the last line...

```
"file://$SCRIPTS/htmlalertreply?i32-msgid=0x00000000&i32-realwhat=0x00000000"
```

... points to a cgi-bin program that's invoked when the user clicks the **Okay** link. The parts of the reference are:

- `$SCRIPTS` is a global variable that points to the directory that contains cgi-bin programs. For rules governing this variable, see `<x>`.
- `htmlalertreply` is the cgi-bin script that's invoked when the link is clicked.
- `msgid` is a unique 32-bit integer identifier for the alert panel.
- `realwhat` is a 32-bit integer that encodes the “create a new alert” request.

## 1.2 A Two Button Alert

`<<TK>>`

## 2 The Alert Content

`<<TK>>`

## 3 Invoking the Alert Script

There are three ways to invoke the **alert script**: Through C++ code (the `TellTellBrowser()` function), through JavaScript code, and through the **TellBrowser** command line program.

### 3.1 Through C++

To invoke the **alert script** through C++ code, you create a `TB_OPEN_ALERT` message and send it to the system through the `TellTellBrowser()` function.

The `TB_OPEN_ALERT` message has four fields. Two of the fields, “itemplate” and “template”, specify the location of the **alert template** file that you want to use. The “content” field identifies the **alert content** file you want to use. The final field, “data”, lets you pass arguments to the templates:

Field	Type code	Description
“itemplate”	<code>B_STRING_TYPE</code>	The path to the <b>alert name mapping file</b> . This is a text file that maps alert template names (delimited by whitespace) to the files that contain the templates' HTML code, in this format: <pre>template_name_1 branch/template_file_1 template_name_2 branch/template_file_2 template_name_3 branch/template_file_3 ...</pre>
“template”	<code>B_STRING_TYPE</code>	The name of the template you want to use. The “template” value must appear as a template name given in the “itemplate” file.

Field	Type code	Description
"content"	B_STRING_TYPE	The path to the <b>alert content</b> file. This is a text file (similar in form to the <b>alert name mapping file</b> ) that maps arbitrary, whitespace-delimited symbols to text strings: <pre> symbol_1 Error message one. symbol_2 Error message two. symbol_3 Error message three. ...</pre>
"data"	B_STRING_TYPE	An array of the symbols (as listed in the <b>alert content</b> file) that you want to use in your alert. The "data" array can contain as many as nine symbols. You refer to a symbol through the \$N variable, where N is the index of the symbol in the "data" array.

As an example, the following code is the BMessage definition that's used by the print server to create a "There's no printer" alert:

```

/* Define the TB_OPEN_ALERT message. */
BMessage msg_alert(TB_OPEN_ALERT);
BMessage reply;
msg_alert.AddString("itemplate", "Alerts/indirect.txt");
msg_alert.AddString("template", "print_no_printer");
msg_alert.AddString("content", "Alerts/printer.txt");
msg_alert.AddString("data", "errnoprt");
msg_alert.AddString("data", "BUY_URL");
msg_alert.AddString("data", "BUY");
msg_alert.AddString("data", "NO");
error=TellTellBrowser(&msg_alert, &reply);

```

### 3.2 Through JavaScript

<<TK>>

### 3.3 Through TellBrowser

<<TK>>



# UI Customization

## Special Keys

BeIA lets you hard-wire any of the keyboard keys to specific commands, such as “volume up”, “sleep,” or “go to *URL*”. This feature is meant to be used in keyboards that have special physical keys that are specifically designed to provide the user with “one touch” shortcuts. It’s not meant to be used to arbitrarily remap the “regular” keys on the keyboard.

This document describes how special keys are mapped to particular actions, and how you can augment or modify the default map.

---

## 1 The Special Key Mapping File

The `/boot/custom/special_keys/map` file contains the special key-to-action map. The file is plain text that you can easily modify. Each line in the file describes a single key mapping in this format:

```
key_number action [ arg ] done
```

The *key\_number* is a hardware-defined hexadecimal number that identifies the physical key. *action* describes the action that the key, when pressed, performs, as described in the next section. *arg* lets you pass an argument to the action (if required). All entries in the `map` file must end with the `done` keyword.

---

## 2 Actions

The *action* part of a `map` file entry can be one of the built-in BeIA commands, an invocation of a shell script, or an invocation of the `be_special_keys()` JavaScript function.

### 2.1 Built-in Commands

There are three built-in commands, all of which pertain to the system audio volume:

- `volume_up` turns the system audio volume up one “tick.”
- `volume_down` turns the volume down a tick.
- `toggle_mute` toggles the system audio mute.

For example, the default `map` file maps keys `0x100002`, `0x200004`, and `0x200005` to these actions:

```
0x100002 toggle_mute done
0x200004 volume_up done
0x200005 volume_down done
```

### 2.2 Shell Scripts

You can map a special key to a shell script through the `exec` action, passing the name of the shell script as an argument:

```
key_number exec script done
```

Note that you can't pass any arguments to the script through a map entry. If you want to invoke a script with arguments, create a script “wrapper” that invokes the desired script (with arguments), and then invoke the wrapper from the map file. As a convenience, BeIA provides a set of script wrappers in the `/boot/custom/special_keys` directory. The wrapper files are numbered (and named) 0 through 16. For example, the default map maps script 16 (print) to the key 0x100010:

```
0x100010 exec 16 done
```

Currently, the only scripts that BeIA provides that you'll want to use—and that you shouldn't overwrite or modify—are 13 and 16. Script 13 toggles the device's sleep mode: It puts the machine to sleep when it's awake, and wakes it up when it's asleep. Script 16, as mentioned above, prints the currently displayed page.

## 2.3 JavaScript

The top-level `$RESOURCES/index.html` file defines the function `be_special_keys()`. This function takes a single argument that represents a pre-defined operation. To invoke the `be_special_keys()` function from a map entry, pass `javascript` as the *action* and the name of the desired operation as the *arg*:

```
key_number javascript operation done
```

For example, the default map declares that key 0x200001 will navigate the browser to the previous web page:

```
0x10000e javascript back done
```

The list of available operations is given below.

Operation	Meaning
<code>back</code>	Navigates (the browser) to the previous page
<code>forward</code>	Navigates to the next page
<code>bookmarks</code>	Brings up the bookmarks list
<code>home</code>	Navigates to the device's home page
<code>settings</code>	Brings up the menu of settings panels
<code>news</code>	Navigates to the preferred news site
<code>shopping</code>	Navigates to the preferred shopping site
<code>email</code>	Brings up the email client (possibly navigating to an email site)
<code>search</code>	Navigates to the preferred search engine
<code>rocket</code>	Navigates to the preferred financial site
<code>lightbulb</code>	Navigates to the preferred entertainment site

## 3 Modifying Special Keys Mappings and Functionality

You can modify the special key mappings simply by editing the `map` file. The only thing you mustn't do is get rid of the file altogether.



To add functionality to the special keys you can:

- Create (and map to) a new shell script, possibly covering the script with a wrapper. Note that you don't have to use the numbered wrappers in the `special_keys` directory. They're provided simply as a convenience.
- Add an operation name to the `be_special_keys()` function (in `$RESOURCES/index.html`). The function, as provided by BeIA, is essentially just a switch statement that invokes a particular JavaScript function given an operation name. You can add your own operations by creating an operation name, a function to go with it, and tying them together in a new branch in the `be_special_keys()` function.



# UI Customization

## Errors and UI Strings

This document lists the error conditions (and other events) that are reported in alert panels, and the strings (including button labels) that are used to describe each condition.

Except for the browser, BeIA components report their errors through the **alerts system**. The system creates a new alert page by filling in an HTML template file (the **alert template**) with some descriptive text taken from an **alert content** file. Each text string in an alert content file is indexed by an **alert constant**. The template and content files are stored in \$RESOURCES/Alerts. See “Alerts” for more information on how alerts works.

In the sections below, the errors are grouped by BeIA component and enumerated by their alert constants. Browser error reporting is described in the browser section, immediately below.

**Note:** All descriptive text shown below is taken literally from the BeIA interface code (HTML pages, text files, etc.). Typos, grammatical errors, and confusing or inconsistent wording will be corrected.

---

## 1 Browser

The browser uses pre-defined alert files to report errors; each of these files corresponds to a particular sort of browser error, as listed below. Strings given as **text** are passed to the files as arguments.

### 1.1 Bad beos:// URL

```
Errors/beos.html
    The web page URL is invalid.
    Tell me more...
```

### 1.2 Bad file:// URL

```
Errors/file.html
    The local file filename could not be shown because the following problem occurred: message.
    Tell me more...
```

### 1.3 Bad http:// URL

```
Errors/connect.html
    A connection could not be made to the remote host host because the following problem occurred: message.
    Tell me more...
```

```
Errors/hostclosed.html
    The web site host unexpectedly closed its connection.
    Tell me more...
```

```
Errors/hostrefused.html
    The web site host refused your connection. Usually this occurs when the site is temporarily down. Try your request
```

again later.

Tell me more...

Errors/hostunknown.html

The web site **host** could not be found. If you have entered the web address manually, please make sure that you have typed it correctly.

Tell me more...

Errors/http.html

While requesting your page, the web site returned an error message: **message (code)**.

Tell me more...

Errors/timeout.html

Your attempt to access **host** failed because the server took too long to respond. This could be due to to server being too busy right now. Try your request again later.

Tell me more...

## 1.4 Unsupported Content or Scheme

Errors/content.html

This device can not display data of type **type**.

Tell me more...

Errors/scheme.html

The requested page could not be shown because the network protocol **scheme** is unknown.

Tell me more...

## 1.5 Password Request

The Errors/password.html template file is filled in and displayed when a server requests a password but doesn't supply its own password window.

You must enter a password to see this page.

Username

Password

Cancel

Login

---

## 2 General Errors

Alerts/content.txt is an alert content file that contains strings that are generally useful in an error panel. The strings are listed below by the error codes to which they correspond.

**error**

An error occurred.

**OK**

OK

**ign**

ignore

### 3 The Update Mechanism

Alerts/doupgrade.txt contains the strings that the update mechanism displays in the alert panel that's presented to the user during and after a major update (or “upgrade”). Each string is represented by a code (“[query](#)”, “[defer](#)”, “[doupgrade](#)”, etc).

Note that minor updates don't interrupt the user with alert panels. For more on the update mechanism, see “The Update Mechanism” in the *BeIA Support* chapter.

#### [query](#)

A system software upgrade is available. Would you like to install the upgrade now? If you defer, you will be given the option to upgrade again before the machine goes to sleep.

#### [defer](#)

Defer

#### [doupgrade](#)

Upgrade Now

#### [ok](#)

OK

#### [status-updating](#)

Downloading update... This may take a few minutes.

#### [status-install](#)

Installing update: -- WARNING -- interrupting your machine at this time may render it unbootable!

#### [success](#)

The upgrade was successful. Your machine will reboot automatically in 10 seconds...

#### [error-install](#)

A serious error has occurred while installing the upgrade. Initiating recovery in 10 seconds...

#### [error-verify](#)

The downloaded upgrade package is damaged. Another upgrade attempt will be made later.

#### [error-download](#)

An error has occurred while attempting to download the upgrade package. Another upgrade attempt will be made later.

#### [error-memory](#)

There is not enough memory available to perform the upgrade. Another upgrade attempt will be made later. If this problem persists, you may need to use the recovery mechanism to install the upgrade.

---

### 4 Midi

Alerts/midiErrors.txt enumerates the MIDI error codes:

#### [midioverload](#)

MIDI can't handle this many instruments.

**Note:** The Alerts/midiErrors.txt file doesn't currently exist.

---

### 5 Printer

Alerts/printer.txt enumerates the printer's error codes:

`errnoprt`

No printer connected!.

`errbusy`

Printer busy! Please wait the end of the current printing.

`errio`

The printer does not respond. Please check the connections.

`errpaper`

Paper out.

`errink`

Ink out.

`errjam`

Paper jam.

`OK`

OK

`CANCEL`

Cancel

`YES`

Yes

`NO`

No

`RETRY`

Retry

---

## 6 RealPlayer

### 6.1 Clip Info

`Alerts/realplayerclipinf.html` is the template for the RealPlayer clip information page. It contains the following (hard-coded) text:

About this Presentation:

Clip:

The fields are filled in through arguments to the template.

### 6.2 Authentication

`Alerts/realplayerauth.html` is the RealPlayer authentication page:

User Name and Password Required

User Name

Password

Submit

Cancel

## 6.3 Error Templates

Alerts/realplayer[1234].txt are error templates that are used to display RealPlayer error text, mapped from the codes listed in the next section. The error page may also display the URL of the faulty clip, the server name, and a “More info:” URL. The realplayer3.txt and realplayer4.txt files contain this hard-coded string:

More info:

## 6.4 Error codes

Alerts/realplayer.txt enumerates RealPlayer’s error codes:

**failedload**

Failed to load the 'rmacore.so.6.0' shared library.

**failedinit**

Failed to initialize - not enough memory or resources.

**cantopen**

Can't open URL.

**OK**

OK

**CANCEL**

Cancel

**YES**

Yes

**NO**

No

**RETRY**

Retry

**CLOSE**

Close

**NOINFO**

No information available

**ITITLE**

Title:

**IAUTHOR**

Author:

**ICOPY**

Copyright:

**SPACE**

‘ ‘ (single whitespace character)

**INFO**

Contact your vendor.

**PNR\_FAIL**

General error. An error occurred.

**PNR\_OUTOFMEMORY**

Out of memory. You may need to close some other applications to play this content.

[PNR\\_INVALID\\_PARAMETER](#)

Invalid parameter. Unable to process request.

[PNR\\_INVALID\\_OPERATION](#)

Invalid operation. Cannot process request.

[PNR\\_NOT\\_INITIALIZED](#)

Not initialized.

[PNR\\_INVALID\\_FILE](#)

RealPlayer cannot play this type of document.

[PNR\\_INVALID\\_VERSION](#)

Invalid file version number.

[PNR\\_DOC\\_MISSING](#)

Requested file not found. The link you followed may be outdated or inaccurate.

[PNR\\_BAD\\_FORMAT](#)

Unknown data format.

[PNR\\_NET\\_SOCKET\\_INVALID](#)

Invalid socket error.

[PNR\\_NET\\_CONNECT](#)

Connection to server could not be established. You may be experiencing network problems.

[PNR\\_BIND](#)

An error occurred binding to network socket.

[PNR\\_SOCKET\\_CREATE](#)

An error occurred while creating a network socket.

[PNR\\_INVALID\\_HOST](#)

Unable to establish a connection with the server.

[PNR\\_INVALID\\_PATH](#)

Requested URL is not valid.

[PNR\\_NET\\_READ](#)

An error occurred while reading data from the network.

[PNR\\_NET\\_WRITE](#)

An error occurred while writing data to the network.

[PNR\\_NET\\_UDP](#)

Cannot receive UDP data packets. You may wish to try the TCP data option in the Network Preferences. You may also want to configure RealPlayer to use a firewall proxy. Please contact your system administrator for more information.

[PNR\\_HTTP\\_CONNECT](#)

Could not connect to Server using HTTP

[PNR\\_SERVER\\_TIMEOUT](#)

Connection to server has timed out. You may be experiencing network problems.

[PNR\\_SERVER\\_DISCONNECTED](#)

Connection to server has been lost. You may be experiencing network problems.

[PNR\\_DNR](#)

Unable to locate server. This server does not have a DNS entry. Please check the server name in the URL and try again.



[PNR\\_OPEN\\_DRIVER](#)

Cannot open the network drivers.

[PNR\\_BAD\\_SERVER](#)

This server is not using a recognized protocol.

[PNR\\_ADVANCED\\_SERVER](#)

This version of RealPlayer G2 cannot access this server.

[PNR\\_OLD\\_SERVER](#)

Connection closed. The host's version of the RealNetworks server is too old for this client.

[SERVER\\_ALERT](#)

Server alert.

[PNR\\_DEC\\_NOT\\_FOUND](#)

File compression not supported. Cannot locate the requested RealPlayer decoder.

[PNR\\_DEC\\_INVALID](#)

The requested RealPlayer decoder is not valid.

[PNR\\_DEC\\_TYPE\\_MISMATCH](#)

Decoder type mismatch. Cannot load the requested decoder.

[PNR\\_DEC\\_INIT\\_FAILED](#)

Requested RealPlayer decoder cannot be found or cannot be used on this machine.

[PNR\\_DEC\\_NOT\\_INITED](#)

RealPlayer Decoder was not initialized before attempting to use it.

[PNR\\_DEC\\_DECOMPRESS](#)

RealPlayer was unable to decompress this content.

[PNR\\_NO\\_CODECS](#)

No codecs have been installed on your system.

[PNR\\_PROXY](#)

Proxy status error.

[PNR\\_PROXY\\_RESPONSE](#)

Proxy invalid response error.

[PNR\\_ADVANCED\\_PROXY](#)

This version of RealPlayer G2 cannot access this proxy.

[PNR\\_OLD\\_PROXY](#)

Connection closed. The proxy is too old for this client.

[PNR\\_AUDIO\\_DRIVER](#)

Cannot open the audio device. Another application may be using it.

[INVALID\\_PROTOCOL](#)

Invalid protocol specified in URL. URLs should typically start with "rtsp://", "pnm://", or "http://"

[PNR\\_INVALID\\_URL\\_OPTION](#)

Invalid option specified in URL.

[INVALID\\_URL\\_HOST](#)

Invalid host string in requested URL.

[PNR\\_INVALID\\_URL\\_PATH](#)

Invalid resource path string in requested URL.

**PNR\_GENERAL\_NONET**

Cannot find network services.

**PNR\_PERFECTPLAY\_NOT\_SUPPORTED**

Requested server does not support PerfectPlay.

**PNR\_NO\_LIVE\_PERFECTPLAY**

PerfectPlay not supported for live streams.

**PNR\_PERFECTPLAY\_NOT\_ALLOWED**

PerfectPlay not allowed on this clip.

**PNR\_MULTICAST\_JOIN**

An error occurred attempting to join multicast session.

**PNR\_GENERAL\_MULTICAST**

An error occurred accessing a multicast session.

**PNR\_MULTICAST\_UDP**

Cannot receive audio data from this multicast session.

**PNR\_SLOW\_MACHINE**

Your CPU is unable to decode this content in real time. Try selecting content that needs less bandwidth in order to receive less complex content.

**PNR\_INVALID\_HTTP\_PROXY\_HOST**

Invalid hostname for HTTP proxy.

**PNR\_INVALID\_METAFILE**

Invalid Metafile

**PNR\_NO\_FILEFORMAT**

No file-format is available to provide playback of this file type on your system.

**NO\_RENDERER**

This version of RealPlayer G2 cannot play the clip you are trying to play.

**PNR\_MISSING\_COMPONENTS**

Some components are not available to provide playback of this presentation on your system.

**PNR\_BAD\_TRANSPORT**

Bad Transport

**NOT\_AUTHORIZED**

Access Denied

**NOTENOUGH\_BANDWIDTH**

You cannot receive this content. You do not have enough network bandwidth.

**PNR\_ENC\_INVALID\_VIDEO**

The file contains an unsupported video format. The needed codec is not installed on your system.

**PNR\_ENC\_INVALID\_AUDIO**

The file contains an unsupported audio format. The needed codec is not installed on your system.

**PNR\_NOTIMPL**

This file is not supported.

**PNR\_INVALID\_REVISION**

Invalid revision number in RealAudio file.

**PNR\_UNEXPECTED**

Unexpected data. Cannot continue.

[PNR\\_NO\\_DATA](#)

There is no data waiting to be processed.

[PNR\\_RETRY](#)

Attempting to reconnect to the RealAudio server.

[PNR\\_AT\\_END](#)

Arrived at the end of the document.

[PNR\\_RECORD\\_WRITE](#)

An error occurred while recording clip to file.

[PNR\\_TEMP\\_FILE](#)

An error occurred while accessing a temp file.

[FILE\\_NOT\\_FOUND](#)

Error: File not found.

[PNR\\_REDIRECTION](#)

Client redirected to new server

[HTTP\\_CONTENT\\_NOT\\_FOUND](#)

Content not found by HTTP.

[PNR\\_EXPIRED](#)

Player license has expired. Contact your vendor.

[PNR\\_INVALID\\_INTERLEAVER](#)

Cannot locate the requested interleaver.

[PNR\\_CHUNK\\_MISSING](#)

RealAudio file is missing the requested data chunk.

[PNR\\_INVALID\\_STREAM](#)

Invalid .rm stream.

[PNR\\_UPGRADE](#)

This version of RealPlayer G2 cannot play this clip.

[PNR\\_INVALID\\_WAV\\_FILE](#)

File is not a RIFF WAV file.

[PNR\\_NO\\_SEEK](#)

Seek is not possible.

[PNR\\_ENC\\_FILE\\_TOO\\_SMALL](#)

File is too small.

[PNR\\_ENC\\_UNKNOWN\\_FILE](#)

Unknown file type.

[PNR\\_ENC\\_BAD\\_CHANNELS](#)

Invalid number of encoder channels.

[PNR\\_ENC\\_BAD\\_SAMPLESIZE](#)

Invalid encoder sample size.

[PNR\\_ENC\\_BAD\\_SAMPRATE](#)

Invalid encoder sample rate.

[PNR\\_ENC\\_INVALID](#)

Invalid encoder.

PNR\_ENC\_NO\_OUTPUT\_FILE

Output file not found.

PNR\_ENC\_NO\_INPUT\_FILE

Input file not found.

PNR\_ENC\_NO\_OUTPUT\_PERMISSIONS

Encoder output permission error.

PNR\_ENC\_BAD\_FILETYPE

Input file type not supported.

PNR\_ENC\_NO\_VIDEO\_CAPTURE

Unable to initialize the video capture device.

PNR\_ENC\_INVALID\_VIDEO\_CAPTURE

The video capture device format is unsupported.

PNR\_ENC\_NO\_AUDIO\_CAPTURE

Unable to initialize the video capture device.

PNR\_ENC\_INVALID\_AUDIO\_CAPTURE

The audio capture device format is unsupported.

PNR\_ENC\_TOO\_SLOW\_FOR\_LIVE

Not enough resources to maintain live encoding.

PNR\_ENC\_ENGINE\_NOT\_INITIALIZED

The encoding engine is not initialized.

PNR\_ENC\_CODEC\_NOT\_FOUND

The requested codec was not found.

PNR\_ENC\_CODEC\_NOT\_INITIALIZED

Codec initialization failed.

PNR\_ENC\_INVALID\_INPUT\_DIMENSIONS

Invalid input video frame dimensions

SMILDUPID

SMIL: Duplicate ID

SMILDOCERR

SMIL: Document Error

UNKNOWN

An unknown error was detected.

---

## 7 SmartCard

Alerts/smartcard.txt enumerates the SmartCard error codes:

cardunknown

Card unknown. Please insert another smartcard.

wrongway

The card is inserted the wrong way.

OK

OK

## 8 Error-less Components

The following BeIA components don't report errors:

- **Flash** doesn't report any errors.
- **Java** doesn't report any errors. However, individual Java applets may report errors by creating their own alert windows.
- **MediaPlayer** doesn't report any errors.
- **The BeIA kernel.** Kernel errors aren't reported to the user through the browser. If something bad happens in the kernel, the machine reboots itself.
- **Data decoders.** Except for the “unsupported content error” (see **1.4** Unsupported Content or Scheme), the BeIA data decoders don't report specific errors.



# UI Customization

## User Interface Files

This document lists and describes the files that make up the BeIA user interface.

---

### 1 /boot/custom/cgi-bin/

`errorgen`, `htmlalertgen`, `htmlalertreply`

cgi-bin scripts used internally by BeIA to generate error and alert pages, and to process user responses to such pages.

---

### 2 /boot/custom/resources/\$LANGUAGE/

The `LANGUAGE` value is stored in the `beos.binder.service.locale.language` Binder property.

`firstboot.html`

Welcome page that's displayed when the device is booted in firstboot bootmode (i.e. the first time the user boots the device). See "Boot Mode" in the *BeIA Support* chapter.

`index.html`

The main entry point for UI configuration, as described in the "The Entry Point Files" in the "Introduction" to this chapter.

`login.html`

Login panel that's displayed when the device is booted in normal mode. See "Boot Mode" in the *BeIA Support* chapter.

`recover.html`

"Invisible" page that's used to restore the last-visited page when the browser comes back up after a crash.

`scripts`

#### 2.1 /boot/custom/resources/\$LANGUAGE/Alerts/

*For a general explanation of the alert system, see "Alerts" in this chapter.*

`alert1.html`, `alert2.html`

Templates for basic one- and two-button alert panels.

`alert_reply.js`

JavaScript functions used by the JavaScript alert templates (`js_alert.html`, `js_confirm.html`, and `js_prompt.html`).

`content.txt`

Default alert panel text strings.

doupgrade.txt

Text strings used in the update alert panel. See “The Update Mechanism” in the *BeIA Support* chapter.

indirect.txt

Maps error codes to alert templates.

javascript.txt

Text strings for JavaScript alerts.

js\_alert.html, js\_confirm.html, js\_prompt.html

JavaScript alert panel templates.

printer.txt

Printer error text strings.

realplayer.txt

**RealPlayer** error text strings.

realplayer1.html, realplayer2.html, realplayer3.html, realplayer4.html

**RealPlayer** alert templates.

realplayer\_i.txt

Maps **RealPlayer** error codes to **RealPlayer** alert templates.

realplayerauth.html

**RealPlayer** login panel for server-requested passwords.

realplayerclipinf.html

**RealPlayer** clip info panel (clip name, length.

redirect1.html, redirect2.html

Templates for one- and two-button alerts that redirect the user to a Web site.

smartcard.txt

Smartcard error text strings.

upgradeinfo.html

Template for system upgrade alert panel. See “The Update Mechanism” in the *BeIA Support* chapter.

## 2.2 /boot/custom/resources/\$LANGUAGE/Bookmarks/

See “UI Customization” in this chapter for a description of the bookmarks UI.

addbookmark.gif, addbookmark\_active.gif, addbookmark\_over.gif

**Add Favorites** button graphic (in various modes) at the top of the Favorites list.

addfolder.gif, addfolder\_active.gif, addfolder\_over.gif

**New Folder** button graphic (in various modes) at the top of the Favorites list.

alloutside.png

bookmark.png

Bookmark icon in the Favorites list.

BookmarkDefs.js

bottom.html

bottomtab.gif

closebox.png, closebox\_active.png

Close box icon (in various modes) at the top of the Favorites list.



delete.png, delete\_outside.png, delete\_over.png

Trash can icon (in various modes) in the Favorites list.

edge.html

edit.html

folder.png, folder\_closed.png

Open and closed folder icons in the Favorites list.

gutterbg.gif

index.html

Page that defines the bookmarks settings panel.

info.html, info\_on.html, info\_onover.html, info\_outside.html, info\_over.html,

Info button icons (in various modes) in the Favorites list.

list.html

listbg.gif

top.html

toptab.gif

## **2.3 /boot/custom/resources/\$LANGUAGE/Cursors/**

cursor.png

Default cursor icon.

ibeam\_cursor.png

Icon that's displayed when the cursor is above editable text.

link\_cursor.png

Icon that's displayed when the cursor is above a hypertext link.

wait\_cursor0.png, wait\_cursor1.png, wait\_cursor2.png, wait\_cursor3.png,

Series of icons (0 through 3) that are displayed when the browser is waiting for data to download.

## **2.4 /boot/custom/resources/\$LANGUAGE/Days/**

0.gif, 1.gif, 2.gif, 3.gif, 4.gif, 5.gif, 6.gif

Icons that name (abbreviated) the days-of-the-week; 0.gif ("Sun") through 6.gif ("Sat").

## **2.5 /boot/custom/resources/\$LANGUAGE/Errors/**

*The files in Errors are HTML templates for alert panels that describe browsing errors and other situations that require the user's attention . See "Errors and UI Strings" in this chapter for more information.*

beos.html

connect.html

content.html

file.html

hostclosed.html  
hostrefused.html  
hostunknown.html  
http.html  
password.html  
scheme.html  
timeout.html

### 2.5.1 /boot/custom/resources/\$LANGUAGE/Errors/template/

erroricon.incl  
footer.incl  
haiku\_default.incl, haiku\_post.incl, haiku\_pre.incl  
title.incl

### 2.6 /boot/custom/resources/\$LANGUAGE/glyphs/

0.gif, 0x.gif, 1.gif, 1x.gif, 2.gif, 2x.gif, 3.gif, 3x.gif, 4.gif, 4x.gif, 5.gif, 5x.gif, 6.gif,  
6x.gif, 7.gif, 7x.gif, 8.gif, 8x.gif, 9.gif, 9x.gif  
Numerals against two different background colors.  
bar\_active.gif, bar\_inactive.gif  
blank.gif, blankx.gif  
checkbox\_active.gif, checkbox\_inactive.gif  
endcapleft\_active.gif, endcapleft\_inactive.gif, endcapright\_active.gif,  
endcapright\_inactive.gif  
left.gif, left\_active.gif  
radio\_active.gif, radio\_inactive.gif  
right.gif, right\_active.gif

### 2.7 /boot/custom/resources/\$LANGUAGE/Home/

background.gif  
email.gif, email-down.gif, email-over.gif  
entertainment.gif, entertainment-down.gif, entertainment-over.gif  
Home.html  
news.gif, news-down.gif, news-over.gif  
search.gif, search-down.gif, search-over.gif

## 2.8 /boot/custom/resources/\$LANGUAGE/Intro/

blankwhite.html

connectionprogress.html

introtutorial.html

regform.html

registrationpl.html, registrationpl.5.html, registrationp2.html, registrationp3.html,  
registrationp4.html, registrationp5.html, registrationp5.5.html

Regp1.html, Regp1.5.html, Regp2.html, Regp3.html, Regp4.html, Regp5.html

returnpolicy.html

termsandconditions.html

WelcomeBG.gif

Background watermark-type image for the welcome page.

welcomepage.gif

### 2.8.1 /boot/custom/resources/\$LANGUAGE/Intro/WelcomeImages/

welcomepage\_new\_06.gif, welcomepage\_new\_15.gif, welcomepage\_new\_18.gif,  
welcomepage\_new\_20.gif, welcomepage\_new\_22.gif, welcomepage\_new\_24.gif,  
welcomepage\_new\_26.gif, welcomepage\_new\_33.gif, welcomepage\_new\_35.gif,  
welcomepage\_new\_37.gif, welcomepage\_new\_42.gif

## 2.9 /boot/custom/resources/\$LANGUAGE/MediaBar/

*The Media bar is used to control the playing of sounds, animations, etc. See xxx for details.*

closebox.png

Graphic for the button used to close the media bar.

endcap\_left.png, endcap\_right.png

Left and right endcap graphics for the media bar.

intersect.png

main.html

JavaScript code that constructs a media bar page.

pause.png, pause\_active.png, pause\_over.png

Various graphics for the “Pause” button in the media bar.

play.png, play\_active.png, play\_over.png

Various graphics for the “Play” button in the media bar.

progressbar.png

Small graphic used in constructing a progress bar.

remainbar.png

Small graphic used in construction a progress bar (the part showing what’s left to play).

## 2.10 /boot/custom/resources/\$LANGUAGE/Months/

0.gif, 1.gif, 2.gif, 3.gif, 4.gif, 5.gif, 6.gif, 7.gif, 8.gif, 9.gif, 10.gif, 11.gif

Icons that name the months, from 0.gif (“January”) through 11.gif (“December”).

## 2.11 /boot/custom/resources/\$LANGUAGE/PopUpDecor/

*Window pane control.*

BottomPaneClose.gif, BottomPaneClose\_over.gif

Graphics for the button that closes the bottom pane, in a two-pane horizontally oriented display.

LeftPaneClose.gif, LeftPaneClose\_over.gif

Graphics for the button that closes the left pane, in a two-pane vertically oriented display.

PanebarHorz.html

HTML layout code for a two-pane horizontally oriented display.

PanebarVert.html

HTML layout code for a two-pane horizontally oriented display.

RightPaneClose.gif, RightPaneClose\_over.gif

Graphics for the button that closes the right pane, in a two-pane vertically oriented display.

TopPaneClose.gif, TopPaneClose\_over.gif

Graphics for the button that closes the top pane, in a two-pane horizontally oriented display.

## 2.12 /boot/custom/resources/\$LANGUAGE/Settings/

Advanced.html

DateTime.html

General.html

Languages.html

Network-ethernet.html

Network-modem.html

Printing.html

Security.html, SecurityBadPass.html, SecurityControl.html, SecurityMessage.html,  
SecurityMismatch.html, SecuritySuccess.html

security.gif, security\_active.gif, security\_over.gif

Settings.html

The main “BeIA” Settings page.

settingsicon.gif

Small graphic displayed in each settings subpage, to identify it as a settings page.

switch-ethernet.gif, switch-modem.gif

### 2.12.1 /boot/custom/resources/\$LANGUAGE/Settings/widgets/

checkbox.checked.gif, checkbox.gif

Images of a generic settings checkbox control, in various states.

`down.gif`, `down_active.gif`

Small triangular down arrow in normal and active states.

`left.gif`, `left_active.gif`

Small triangular left arrow in normal and active states.

`radiobutton.checked.gif`, `radiobutton.gif`

Images of a generic settings radio button, in various states.

`right.gif`, `right_active.gif`

Small triangular right arrow, in normal and active states.

`slider.endcap.left.filled.gif`, `slider.endcap.left.gif`, `slider.endcap.right.filled.gif`,  
`slider.endcap.right.gif`, `slider.middle.filled.gif`, `slider.middle.gif`

Images used to construct the appearance of a slider control. The slider will be entirely empty (representing, for example, volume turned down all the way), or will have some portion of the slider filled from the left (which may constitute all of the slider, if the control is set to its maximum level).

`up.gif`, `up_active.gif`

Small triangular up arrow, in normal and active states.

### 2.13 /boot/custom/resources/\$LANGUAGE/SoftKeyboard/

*The files in the SoftKeyboard directory are used by the soft (touch-screen) keyboard and foreign language input methods. For more information on the soft keyboard, see <<>>.*

`jim.html`

The HTML file that loads the plug-in for the Japanese input method.

`main.html`

The HTML file that loads the plug-in for the soft keyboard.

`qwerty.kbd`

Keyboard layout file for the soft keyboard.

`SoftKeyboardClose.gif`, `SoftKeyboardClose_over.gif`

Close button images displayed by the soft keyboard.

#### 2.13.1 /boot/custom/resources/\$LANGUAGE/SoftKeyboard/key\_graphics

`key_87_UD.png`, `key_97_UD.png`, `key_98_UD.png`, `key_99_UD.png`

Non-default key images displayed by the soft keyboard.

### 2.14 /boot/custom/resources/\$LANGUAGE/Time/

`am.gif`

Text image **AM**.

`colon.gif`

Character image : (colon).

`pm.gif`

Text image **PM**.

### 2.15 /boot/custom/resources/\$LANGUAGE/Toolbar/

`toolbar.html`

HTML code that assembles the graphics in this directory into the toolbar seen by the user.

### 2.15.1 /boot/custom/resources/\$LANGUAGE/Toolbar/Images

`back.gif`, `back-down.gif`, `back-over.gif`

The toolbar's go-back-one-page button, in various states.

`favorites.gif`, `favorites_active.gif`, `favorites-down.gif`, `favorites-over.gif`

The toolbar's favorites button, in various states.

`forward.gif`, `forward-down.gif`, `forward_over.gif`

The toolbar's go-forward-one-page button, in various states.

`goto.gif`, `goto-down.gif`, `goto-over.gif`, `goto_active.gif`, `goto_active-down.gif`, `goto_active-over.gif`, `goto_area.gif`, `goto_area-active.gif`

The toolbar's load-page button, in various states.

`home.gif`, `home-down.gif`, `home-over.gif`

The toolbar's go-to-home button, in various states.

`left_edge.gif`

`magnify.gif`, `magnify-down.gif`, `magnify-over.gif`

The toolbar's text magnification button, in various states.

`reload.gif`, `reload-down.gif`, `reload-over.gif`

The toolbar's reload-page button, in various states.

`right_edge.gif`

`settings.gif`, `settings_active.gif`, `settings-down.gif`, `settings-over.gif`

The toolbar's go-to-settings button, in various states.

`spacer.gif`

`status_busy.gif`, `status_connected.gif`, `status_offline.gif`

The connection status icon, in various states.

`stop.gif`, `stop-down.gif`, `stop-over.gif`

The toolbar's stop-loading button, in various states.

---

## 3 /boot/custom/resources/scripts/

`buildprefs.js`

Very small JavaScript fragment used when starting up.

`buttons.js`

JavaScript functions to let images act as buttons; includes such things as highlighting when the mouse passes over a button.

`network.js`

Network-related JavaScript functions; includes functions to find information about the network configuration, and glue function which request the underlying OS to make or break dialup connections.

`widgets.js`

JavaScript code to handle the behavior of widgets other than buttons, such as checkboxes and sliders.

## 4 /boot/custom/sounds/

BeBeep.wav

General beep sound.

cache.ini

List of sounds (and some default amplitude adjustments) that are loaded into RAM at boot time. Also defines the startup sound.

MouseDown.wav

Sound used when the mouse button is pressed.

MouseEnter.wav

Sound used when the mouse cursor enters a button.

MouseEnterTool.wav

MouseLeave.wav

MouseUp.wav

Sound used when a clicked button is released.

MuteToggle

Sound used when audio mute is toggled.

PageLoaded.wav

Sound used when a page finishes loading.

PowerDown.wav

Sound used when the device is powered up.

PowerUp.wav

Sound used after the machine is powered up, to let the user know it is ready to use.

VolumeDown

Sound played when the volume is adjusted, so the user can hear the new volume.

VolumeUp

Sound played when the volume is adjusted, so the user can hear the new volume.

WindowActivated

Sound used to indicate a window has been activated.

---

## 5 /boot/custom/special\_keys/

*The files in the special\_keys directory map specific keys on the keyboard to certain pre-defined operations (toggle the volume mute, put the device to sleep, etc.) For more information, see “Special Keys” in this chapter.*

0-16

Shell scripts that are designed to be invoked from the special keys. Currently, only scripts 13 (sleep/wake) and 16 (print) are used.

map

Keycodes-to-actions map for the special keys. The actions are either shell executions of one of the

files described above (or any other shell script), calls to built-in functions, or calls to the JavaScript `be_special_key()` function (defined in `$RESOURCES/index.html`).

---

## 6 /boot/home/config/settings/

The files in `/boot/home/config/settings` set the values of certain device parameters. In addition to the references given below, the files are (generally) described in “Settings Files.”

`beia-bootmode`

Defines the environment (developer, test, user, etc.) that the device will boot into. See “Boot Mode” in the *BeIA Support* chapter.

`RealNetworks_RealMediaSDK_60`

**RealPlayer** configuration and plug-in registration list.