CUSTOMIZING GUIDES USING ADOBE® FLASH® BUILDER™
Contents

Chapter 1: About Customizing Guides Using Flash Builder
Who should read this document? ................................................................. 1
Additional information ........................................................................... 1

Chapter 2: Overview

Chapter 3: Creating Flex library Projects for Custom Guides
Setting up your development environment ........................................... 3
Creating a Flex library project ............................................................... 3
The folder structure for the Flex library project ..................................... 4
Importing sample Flex library projects ............................................... 5
Building your custom Flex library project ........................................... 5
Recompiling a library project for a migrated Guide ............................... 6
What’s next? ......................................................................................... 6

Chapter 4: Creating Custom Style Sheets
Creating a style sheet SWF file using Flash Builder ................................. 7
Creating a style sheet SWF file using a command-line compiler .............. 7
Referencing custom style sheets in a Guide ......................................... 8
What’s Next? ....................................................................................... 8

Chapter 5: Creating Guide Layouts
Overview of Guide layouts .................................................................... 9
Getting started creating Guide layouts ............................................... 10
Button Bar Guide layout ....................................................................... 14
Binding To A Data Model ..................................................................... 14
What’s next? ....................................................................................... 15

Chapter 6: Creating Panel Layouts
Overview of panel layouts ................................................................... 16
Getting started creating panel layouts ............................................... 19
What’s next? ....................................................................................... 22

Chapter 7: Creating Controls
Overview of Guide controls ................................................................... 23
Field controls in Workbench ............................................................... 23
Creating a control ............................................................................... 24
Creating Guide layout controls ........................................................... 25
Creating custom field controls ............................................................ 29
What’s next? ....................................................................................... 31

Chapter 8: Custom Components Reference
Button components ............................................................................... 32
Help components ............................................................................... 33
Label components ............................................................................... 33

Last updated 10/29/2010
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Navigation components</td>
<td>33</td>
</tr>
<tr>
<td>Output components</td>
<td>34</td>
</tr>
<tr>
<td>Chapter 9: Guide CSS Reference</td>
<td></td>
</tr>
<tr>
<td>Guide CSS classes</td>
<td>35</td>
</tr>
<tr>
<td>Guide CSS properties</td>
<td>45</td>
</tr>
<tr>
<td>Index</td>
<td>59</td>
</tr>
</tbody>
</table>
Chapter 1: About Customizing Guides Using Flash Builder

Welcome to Customizing Guides Using Adobe® Flash® Builder™. This document provides information about creating custom Guide layouts, panel layouts, and controls using Adobe® Flash Builder.

Who should read this document?

This document is intended for Flash developers who are interested in learning how to create custom Guide extensions. You create custom Guide extensions to extend the Guide components shipped with Adobe® LiveCycle® Workbench 9.5 or to meet specific needs.

A knowledge of Guides, Workbench, and Flash Builder is assumed.

Additional information

Adobe has various resources about Guides focused at different audiences. To view these resources, go to the location specified in the See column in the following table.

<table>
<thead>
<tr>
<th>For information about</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>The starting point for learning about Guides. This document includes a walkthrough of</td>
<td>Creating Guides, available</td>
</tr>
<tr>
<td>creating a Guide. It also includes an example of how to render and deploy a Guide to</td>
<td>within Workbench</td>
</tr>
<tr>
<td>Adobe® LiveCycle® Workspace 9.5 by using processes created in Workbench.</td>
<td></td>
</tr>
<tr>
<td>Creating and editing Guides using the Guide Design perspective in Workbench.</td>
<td></td>
</tr>
<tr>
<td>Rendering and deploying Guides using processes created in Workbench.</td>
<td></td>
</tr>
<tr>
<td>Debugging Guides.</td>
<td></td>
</tr>
<tr>
<td>The ActionScript® classes and properties included with Adobe® LiveCycle® Enterprise</td>
<td>ActionScript Language</td>
</tr>
<tr>
<td>Suite 2.5 (ES2.5) software.</td>
<td></td>
</tr>
<tr>
<td>LiveCycle ES2.5 terminology</td>
<td>LiveCycle ES2.5 Glossary</td>
</tr>
<tr>
<td>Other services and products that integrate with LiveCycle ES2.5.</td>
<td><a href="http://www.adobe.com">www.adobe.com</a></td>
</tr>
<tr>
<td>Patch updates, technical notes, and additional information on this product version</td>
<td>LiveCycle Technical Support</td>
</tr>
</tbody>
</table>
Chapter 2: Overview

Using Flash Builder, you can go beyond the Guide customizing options available in the Guide Design perspective in Workbench. You can create Guide extensions designed to suit your specific needs. For example, a Flash developer can create a custom Guide layout control that displays Guide sections and panels in a tree structure.

The process for creating Guide extensions, whether they are Guide layouts, panel layouts, or controls, follows the same general steps:

1. Create a Flex® library project in Flash Builder (see “Creating Flex library Projects for Custom Guides” on page 3).
2. Create Guide extensions as part of the Flex library project. See one of the following:
   - “Creating Custom Style Sheets” on page 7
   - “Creating Guide Layouts” on page 9
   - “Creating Panel Layouts” on page 16
   - “Creating Controls” on page 23
3. Compile the Flex library project to an SWC file (see “Building your custom Flex library project” on page 5).
4. Import the SWC file or SWF file into the Guide Design perspective in Workbench. (See “Referencing custom style sheets in a Guide” on page 8, “Referencing your Flex library project in Workbench” on page 13, “Referencing your Flex library project in Workbench” on page 22, or “Referencing your Flex library project in Workbench” on page 30).
5. Apply the new Guide extensions to a Guide.
Chapter 3: Creating Flex library Projects for Custom Guides

You can configure your Flash development environment and to create Flex library projects for Guide extensions.

Setting up your development environment

Before you can create custom Guide layouts, panel layouts, and controls, set up your development environment. To create custom Guides, you must have the following software installed on your computer.

Minimum required software

- Flex 3.4.1 SDK
- Access to the Guides SDK available in the Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK\misc\Guides folder where Workbench is installed (by default C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK\misc\Guides).

You can download the Flex 3.4.1 SDK.

Recommended software

- Flash Builder 4 (Standard or Premium edition) or Flash Builder 4 Plug-in for Eclipse
- LiveCycle Workbench ES2

See the system requirements for Flash Builder and for Workbench.

Note: The term Flash Builder in this document refers to all versions of Flash Builder (previously Flex Builder™).

Creating a Flex library project

The first step in creating Guide extensions is to create a Flex library project in Flash Builder. When completed, the Flex library projects you create must be compiled into an SWC file for importing into the Guide Design perspective in Workbench. An SWC file is an archive file for Flex components and other assets.

Each Flex library project you create must include the following elements:

- A reference to the Guide runtime library (SWC) file.
- A folder structure with specific subfolders for each type of Guide extension. For more information about creating the correct folder structure, see “The folder structure for the Flex library project” on page 4.

In addition, you can optionally include additional SWC files that contain MXML components or ActionScript classes that you want to leverage. You can also include other assets such as image files or videos.

Create a Flex library project for custom Guides

1. Start Flash Builder.
2. Select File > New > Flex Library Project.
3 Type a project name, assign a workspace, set the Flex SDK version to the default Flex 3.4.1 version of the SDK, and then click Next.

4 Click Library Path.

5 Click Add SWC.

6 Go to the LiveCycle_ES_SDK\misc\Guides\code\libs folder where Workbench is installed. By default, the path is C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK\misc\Guides\code\libs. Add the following SWC file to the project:
   - dcruntime_library.swc

7 Select the Link Type option for the SWC file, and click Edit.

8 (Optional) Set the value of the Link Type option to External. By keeping the dcruntime_library.swc file as an external reference, you reduce the size of the compiled Guide extension (SWF file).

9 Click Finish.

After you create the Flex library project, create a folder structure to store your Guide extensions. Each type of Guide extension must be in an appropriately named project folder.

The folder structure for the Flex library project

Each custom Flex library project must include Guide extensions stored in a specific folder structure. Compile the project to an SWC file and import the SWC file into the Guide Design perspective in Workbench. After you have imported the SWC file, Guide extensions are loaded from the project folders and made available in the Guide Properties view.

The folder structure for the custom Flex library must contain the following elements:

- A top-level folder that has a unique name. The top-level folder in the basic folder structure example below is named custom.

  Important: The folder cannot be named ga.

- At least one nested subfolder that has one of the following names:
  - controls (field controls)
  - layouts (panel layouts)
  - wrappers (Guide layouts)

  Note: Do not save Guide layout controls, such as panel navigation, in these folders. Save them in any other folder.

For example, the image below illustrates a simple valid folder structure.

A basic folder structure that includes subfolders for all Guide extensions.
Creating Flex library Projects for Custom Guides

**Importing sample Flex library projects**

Included with the LiveCycle ES2.5 SDK is a pre-configured Flex project, guide_extension, that contains sample Guide MXML and ActionScript extensions. Import this project to help you start creating Guide extensions more quickly.

*Important: Try not to edit the sample files directly or modify those files that are included with LiveCycle ES2.5. It is a best practice to make uniquely named copies of the content to help you get started creating Guide extensions.*

**Import the sample Guide extensions Flex project**

1. Start Flash Builder.
2. Click File > Import > Flex Project.
3. Select Project Folder and then click Browse. Select the LiveCycle_ES_SDK\misc\Guides\samples\projects\extensions folder located where you installed LiveCycle ES2.5 (by default, C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK\misc\Guides\samples\projects\extensions), and then click OK.
4. (Optional) Choose the workspace where you want to import the new project.
5. Click Finish.

After you import the project, build it to generate the custom extension SWC file. By default, the name of the compiled SWC file is the same as the project (for example, extensions.swc). It is located in your Flex project’s bin directory. The file is located relative to the project’s Adobe® LiveCycle® Workspace 9 directory, not in the original sample project directory. Copy the extensions.swc file into a Workbench application. Add the compiled extension library to your Guide using the Guide Design perspective in Workbench.

**Building your custom Flex library project**

After you create Guide extensions, add them to LiveCycle ES2.5 applications to make the extensions available for use in Guides.

**Compile your Guide extension Flex library project**

1. Start Flash Builder.
2. If your Flex library project is set to compile automatically, proceed to step 3. Otherwise, in the Navigation view, select your project and click Project > Build Project.
3. Add the compiled project SWC file to a LiveCycle ES2.5 application using Workbench.

For more information on adding resources to a LiveCycle ES2.5 application, see Working with assets.

**Renaming project components**

If you rename MXML components or ActionScript classes, manually include the renamed files in your Flex library.

**Include renamed MXML components or ActionScript classes in a Flex library project**

1. Ensure that your Flex project is open.
2. Go to the MXML component or ActionScript class that you renamed. Right-click the file and select Include Class in Library.
3. If you did not configure Flash Builder to build automatically, manually rebuild your Flex library project.
Recompiling a library project for a migrated Guide

You can migrate a form guide that was created in LiveCycle Designer ES (version 8.x) to the Guides format in Workbench. If the form guide includes custom components, recompile the custom Flex library project file (SWC) using the Flex 3.4.1 SDK.

You need the original source files used to create the custom component. The original author can simply open the Flex library project and change the SDK that is used for compiling to the Flex 3.4.1 SDK.

Alternatively, if you have the source files, create a Flex library project, use the Flex 3.4.1 SDK, and then copy the source into the project.

After the source is compiled with the Flex 3.4.1 SDK, import the custom library file (SWC) into a Workbench application.

What’s next?

After you import the sample Guide projects, you can start creating your own Guide layouts. (See “Creating Guide Layouts” on page 9.)
Chapter 4: Creating Custom Style Sheets

Guides supports the ability to use custom style sheets (CSS) compiled as SWF files to control the appearance of Guide components at runtime. In general, creating a custom style sheet for use with Guides requires the following steps:

1. Create a CSS containing the necessary Guide CSS styles. For a list of CSS classes and properties used by Guides, see “Guide CSS Reference” on page 35.

2. Do one of the following:
   - Add a CSS file to Flex project. Right-click the CSS file and ensure that the Compile CSS to SWF option is selected (see “Creating a style sheet SWF file using Flash Builder” on page 7).
   - or -
   - Add the generated style sheet SWF file to a LiveCycle ES2.5 application in Workbench. Reference the style sheet SWF file in a Guide (see “Referencing custom style sheets in a Guide” on page 8).

Creating a style sheet SWF file using Flash Builder

You can create a style sheet SWF file for use with a Guide. Use Flash Builder to compile a CSS file to a SWF file.

You can use the CSS files as a starting point for creating a custom style sheet. The CSS files are located in the LiveCycle_ES_SDK\misc\Guides\css folder where Workbench is installed. By default, the path is C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK\misc\Guides\css.

Create a style sheet SWF file using Flash Builder

1. Start Flash Builder.
2. Select File > New > Flex Project.
3. Type a project name and assign a workspace.
4. Set the Flex SDK version to the Flex 3.4.1 version of the SDK, and then click Next.
5. Click Finish.
6. Add a CSS containing Guide styles, as well as any assets referenced (such as images), to your Flex project.
7. Right-click the CSS file and ensure that the Compile CSS to SWF option is selected.

Add the generated SWF file, which is available in the bin-debug folder of the Flex project, to a LiveCycle ES2.5 application in Workbench. For information on adding resources to a LiveCycle ES2.5 application, see “Referencing custom style sheets in a Guide” on page 8.

Creating a style sheet SWF file using a command-line compiler

You can create a style sheet SWF file for use with a Guide using the command-line MXML compiler included with the Flex SDK available with Workbench.
Customizing Guides Using Flash Builder
Creating Custom Style Sheets

Create a style sheet SWF file using a command-line compiler
1 On the machine where Workbench is installed, ensure that the latest Java Runtime Environment (JRE) is installed.
2 Click Start > Programs > Accessories > Command Prompt.
3 Navigate to the Flex_SDK\bin subfolder of the Workbench install location. By default, the path is C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\Flex_SDK\bin.
4 Type the following and then press Enter:
   mxmlc [sourcepath]/[filename].css -output=[destpath]/[filename].swf
   Where [sourcepath] is the folder location of the source CSS file, and [destpath] is the folder location for the compiled SWF file.

Add the generated SWF file to a LiveCycle ES2.5 application in Workbench. For information on adding resources to a LiveCycle ES2.5 application, see “Referencing custom style sheets in a Guide” on page 8.

Referencing custom style sheets in a Guide

After you build your Flex library project in Flash Builder, reference the compiled SWC file in the Guide Design perspective of Workbench.

Import your custom style sheet in Workbench
1 Start Workbench.
2 Add your custom style sheet to a LiveCycle ES2.5 application.
3 Switch to the Guide Design perspective, and open a Guide into which you want to incorporate the custom style sheet.
4 In the Guide Tree view, select the root node.
5 In the Guide Properties view, deselect Use default stylesheet.
6 In the Guide style option, click .
7 Go to the SWF file for your custom style sheet, and then click OK.

Your custom style sheet is now applied to the Guide. Click Preview to render the Guide.

Important: If you alter a custom style sheet (SWF file) after you add it to a LiveCycle ES2.5 application, remove the reference to the SWF file in the Guide extensions option (Guide Properties view). After you remove the reference, add it a second time for the changes to be available.

What’s Next?

In addition to style sheets, which apply styles broadly across a Guides, you can create Guide extensions that use customized styling. For more information, see “Creating Guide Layouts” on page 9, “Creating Panel Layouts” on page 16, and “Creating Controls” on page 23.
Chapter 5: Creating Guide Layouts

A Guide layout defines the visual layout and structure of a Guide that remains constant throughout a data capture session. The Guide Design perspective in Workbench includes default Guide layouts. The layouts are designed to help you quickly create Guides that are structured in visually appealing and meaningful ways. However, using Flash Builder, you can create new Guide layouts to structure the data capture experience of your end users to meet your specific needs. Guide layouts are derived from the ga.controls.wrapper class in the Guides ActionScript API. For more information about the Guides ActionScript API, see ActionScript Language Reference.

Overview of Guide layouts

In general, a Guide layout consists of a number of components that divide the rendered Guide into distinct areas:

- Guide Help
- Panel content
- Navigators
- Navigation controls
- Toolbar

The following image illustrates one example of a Guide layout structure.
Getting started creating Guide layouts

To get started creating Guide layouts, consider the following information:

- **“Creating a simple Guide layout” on page 10**
  
  Walks through creating a basic Guide layout using MXML.

- **“Referencing your Flex library project in Workbench” on page 13**
  
  Add your Flex library project to your LiveCycle ES2.5 application in Workbench to make the custom Guide layout available for your Guide.

**Creating a simple Guide layout**

Creating a simple Guide layout is an easy way to get familiar with basic concepts, including the structure and MXML definition of a Guide layout.

**Important:** Try not to edit the Guide layout files that are included with LiveCycle ES2.5. Instead, make uniquely named copies of the content to help you get started creating Guide layout extensions.
11

CUSTOMIZING GUIDES USING FLASH BUILDER

Creating Guide Layouts

Last updated 10/29/2010

Create a simple Guide layout
1 Start Flash Builder.
2 Create a Flex library project and configure it using the procedures in “Creating Flex library Projects for Custom Guides” on page 3. Ensure that you create the required folder structure in your Flex library project. Then, create Guide layouts in the wrappers folder. After you create the layouts, they are displayed in the list of Guide layouts in the Guide Design perspective of Workbench.
3 Right-click the wrappers folder and select New > MXML Component.
4 Type a unique filename. By default, the Guide Design perspective of Workbench adds a space immediately before each capital letter in the name of your component. For example, an MXML component named TabNav.mxml appears as Tab Nav in the Guide Design perspective of Workbench.
5 In the Based On list, select Wrapper.
6 (Optional) Set values for Width and Height.
7 Click Finish.
The MXML source for your new Guide layout looks like the following:

```xml
<?xml version="1.0" encoding="utf-8"?>
<Wrapper xmlns="ga.controls.*"
    xmlns:mx="http://www.adobe.com/2006/mxml">
</Wrapper>
```

Note the <Wrapper> element includes the namespace attribute xmlns="ga.controls.*". It is considered good practice to provide namespaces when you reference objects from the Guide API. Providing namespaces reduces the amount of MXML code and makes the code more readable. Updating the MXML source for the blank panel layout, the panel layout source looks like the following:

```xml
<?xml version="1.0" encoding="utf-8"?>
    xmlns:gc="ga.controls.*">
</gc:Wrapper>
```

Adding content to your Guide layout
After you create the shell of the new guide layout, add Flash Builder components using the Flash Builder Source view. Use the components that provide the behavior you are trying to achieve. In addition, you can include other Guide and Flex components to suit your specific needs.

In this example, the blank panel layout is extended to include an area for displaying panel content. It also includes some navigation buttons to move between panels and submit the Guide data.

Add the following to the blank panel layout:

- A standard Flex component for organizing objects into a vertical list. The padding settings provide a margin around the Guide content:

```xml
<mx:VBox width="100%" height="100%">
    paddingTop="5" paddingBottom="5" paddingLeft="5" paddingRight="5">
Note: Alternatively, you can create margins using the setting and value styleName="guide". This method derives the margins from the CSS values.

- The region of the Guide layout for displaying the current data entry panel and the associated layout:

```xml
<gc:PanelContent width="100%" height="100%" />
```
- A standard Flex component for organizing objects into a horizontal list:
  `<mx:HBox>`

- A button object that goes to the previous panel in the Guide. This object is inactive if the current panel is the first panel in the Guide. The `label` attribute controls the button caption text:
  `<gc:PreviousPanelButton id="previous" label="{'Back'}"/>`

  **Note:** The curly brackets and single quotation mark that enclose the button text prevent the component from using resource strings. This text is applied after the component is initialized.

- A button object that goes to the next panel in the Guide. This object is inactive if the current panel is the last panel in the Guide:
  `<gc:NextPanelButton id="next" label="{'Forward'}"/>`

- A button object that submits the Guide data. This object is inactive if the current panel is not the last panel in the Guide:
  `<gc:SubmitButton label="{'Submit Data'}"/>`

  **Note:** Specifying a value for the `label` attribute overwrites the default SubmitButton labels. The default SubmitButton labels are added based on the submission option selected in the Guide Design perspective of Workbench.

The SubmitButton component behaves differently depending on the submission options the Guide author sets for the Guide in the Guide Design perspective of Workbench. The following table outlines the behavior of the SubmitButton component for each submission option:

<table>
<thead>
<tr>
<th>Submit option</th>
<th>SubmitButton behavior</th>
<th>Default SubmitButton label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guide (Default)</td>
<td>Users must submit the data by clicking the submit button on the Guide.</td>
<td>Submit</td>
</tr>
<tr>
<td>Interactive PDF</td>
<td>Clicking the submit button opens the interactive PDF form from which the form filler submits the data by clicking the appropriate submit button. This option is available when the Guide is based on an XDP form that includes an interactive PDF form. When you select this option, a Submit From PDF button appears in the navigation button area. Clicking this button switches the view to the interactive PDF document, which must include a Submit button. The Submit From PDF button appears in all environments where the Guide can run, such as Adobe® LiveCycle® Workspace 9 or a browser. No other submit button appears in the navigation button area of the Guide or in the Workspace chrome.</td>
<td>Submit from PDF</td>
</tr>
<tr>
<td>User Button</td>
<td>Submits the data from a button that you add to a panel in the Guide. This submit button appears in all environments where the Guide can run. No other submit button appears in the navigation button area of the Guide or in the Workspace chrome.</td>
<td>N/A</td>
</tr>
<tr>
<td>Host</td>
<td>The SubmitButton component does not appear on the Guide. The hosted application, such as Workspace, must extract the data from the Guide and perform the data submission.</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Your MXML source looks like the following snippet and illustration.
Ensure the Flex library project builds with no warnings or errors. You can now move on to “Referencing your Flex library project in Workbench” on page 13.

**Referencing your Flex library project in Workbench**

After you build your Flex library project in Flash Builder, reference the compiled SWC file in the Guide Design perspective of Workbench. Creating the reference makes your Guide layout extension available.

**Import your Flex library component (SWC) in Workbench**

1. Start Workbench.
2. Add your Flex library project to a LiveCycle ES2 application.
3. Switch to the Guide Design perspective, and open a Guide into which you want to incorporate the custom Guide layout.
4. In the **Guide Tree** view, select the root node.
In the **Guide Properties** view, in the **Guide extensions** option, click 📝.

Go to the SWC file for your Flex library project, and then click OK.

Your Guide extensions are now available in the **Guide layouts** option in the **Guide Properties** view. Click **Preview** to render the Guide.

**Important:** If you edit the Flex library project after you add the project to a LiveCycle ES2.5 application, remove the reference to the project. The reference is found in the **Guide extensions** option (**Guide Properties** view). After you remove the reference, to make the changes available, add the reference again.

**What’s next?**

Try creating one of your own custom Guide layouts by starting with a new blank layout. Alternatively, use the MXML source for one of the Guide layouts included with Workbench to get started. To learn more about custom Guide layouts by walking through a larger example, see the section “**Button Bar Guide layout**” on page 14.

**Button Bar Guide layout**

To better understand how to structure Guide layouts, this document uses the Button Bar Guide layout as an example. The Button Bar Guide is included in the LiveCycle_ES_SDK\misc\Guides\code\src\ga\wrappers folder where Workbench is installed. In addition to the structure, the Button Bar Guide layout contains the following custom ActionScript variables. These variables define the physical location of the regions of the Guide for easy referencing:

- **TOOLBAR_TAB**: int (the default value is 100)
- **HELP_CENTER_TAB**: int (the default value is 200)
- **BUTTONS_TAB**: int (the default value is 9000)

In addition, the Button Bar Guide layout contains the following custom ActionScript functions:

- **createChildren()**: void
  
  Adds a new PAGE_SELECTION_CHANGE Guide event listener to the Guide layout.

- **pageChange(event:GAEvent)**: void
  
  Dispatched when a user navigates to the next section of the Guide. The Button Bar Guide layout uses a navigation control that segments the Guide according to the sections you define in the Guide tree.

**Binding To A Data Model**

Guide layouts can access items from a bound data model. For example, you could read a numerical value from a data model that acts as a record number. The record number appears in the upper-right corner of the Guide layout throughout the duration of the data capture session. Besides merely displaying data values that you get from the data model, you can also execute business logic based on those values. This logic can alter the intent of your business process.
Create a set function

Set up a function that references whenever the underlying data model item changes. The function uses the binding value of the data model item to obtain the value of the item. For example, the following MXML demonstrates a reference to the value of a data model item, empName, that stores an employee name:

```xml
<?xml version="1.0" encoding="utf-8"?>
<gc:Wrapper width="100%" height="100%"
xmlns:mx="http://www.adobe.com/2006/mxml"
xmlns:gc="ga.controls.*" >

<mx:Script>
<![CDATA[
Wrapper.instance.bindSetter("Services.employee.empName", empNameSetter);

private function empNameSetter( value:Object ):void
{
    var empName:String = value as String;  // cast to appropriate type
    If ( empName == "Tony Blue" ) {
        // do something here
    }
}
]]>
</mx:Script>

<mx:VBox width="100%" height="100%">
    <mx:PanelContent width="100%" height="100%" />
    <mx:HBox>
        <gc:PreviousPanelButton label="Back" />
        <gc:NextPanelButton label="Forward" />
        <gc:SubmitButton label="Submit Data" />
    </mx:HBox>
</mx:VBox>
</gc:Wrapper>
```

Each time the Services.employee.empName data item changes value, the named function executes with the value of the item. For example, you could save the value locally and have MXML controls use that local variable as their `dataProvider`.

What’s next?

Create your own custom Guide layouts, beginning with the simple Guide layout created in the section “Creating a simple Guide layout” on page 10. Alternatively, copy and modify one of the default Guide panel layouts. To create more complex layouts, use a default panel layout that uses scripting to create the PanelItem object (for example, VariableColumn, Repeater, or PieChartHorizontal).

For more information about the Guides ActionScript API, see ActionScript Language Reference.

To start learning about creating custom panel layouts, see “Creating Panel Layouts” on page 16, or for custom Guide controls, see “Creating Controls” on page 23.
Chapter 6: Creating Panel Layouts

A panel layout defines the visual layout and presentation of objects on a panel in the Guide tree. The Guide Design perspective in Workbench includes default panel layouts. These default layouts are designed to help you quickly create Guides with panels that are structured in visually appealing and meaningful ways. Using Flash Builder, you can create new panel layouts to structure panel content in new ways. You can also include Flex objects that take advantage of the capabilities of Flex.

Overview of panel layouts

In general, a panel layout consists of components that act as placeholders for text, objects, or Guide extensions specified in the Guide Design perspective of Workbench. The following are the most common Guide placeholder components that appear on panel layouts:

PanelItem: A placeholder object that displays an item, or a repeating sequence of items, from the Guide tree; either a field, text object, or button.


PanelTitle: A placeholder object that displays the name of the current panel. The panel title is specified in the Guide Design perspective in Workbench.

HelpPanel: Displays panel help text to a form filler. The panel help text is specified in the Guide Design perspective in Workbench.

The following image illustrates one example of how a panel layout structure reflects the common placeholder components.
A. Panel help text (HelpPanel) that a Guide author enters using the Guide Design perspective in Workbench. Depending on the Guide layout, the panel help can appear as part of the panel. I can also appear with the Guide Help text in the Help Center area of the Guide layout. B. Panel content that consists of Guide objects (PanelItem) as well as Flex components. C. Panel title text (PanelTitle) for the panel that a form author enters using the Guide Design perspective in Workbench.

The MXML source code for the panel layouts included with Workbench are available in the LiveCycle_ES_SDK\misc\Guides\code\src\ga\layouts folder where Workbench is installed. By default, the folder is C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK\misc\Guides\code\src\ga\layouts.

For example, examine the MXML source for the One Column panel layout:
The MXML code is distributed into the following sections:

- Initial namespace definitions. Namespaces are a convenient way to define shortcuts to ActionScript packages to simplify your code. Although not required, they are recommended.

- `<mx:Metadata>`
  
  This MXML block defines an icon image to display in the Components view of Flash Builder.

- `<mx:Script>`
  
  The One Column panel layout is an MXML component, and it is created declaratively in MXML. Panel layouts created declaratively require this `<mx:Script>` block to display form design objects.

- `<ga:PanelItem>` and `<gc:HelpPanel>`
  
  Placeholder components for the panel help and Guide objects respectively. Each of these components is discussed in more detail later, but understanding `PanelItem` is critical for creating custom panel layouts.

### Understanding the PanelItem class

The Guide Tree view in the Guide Design perspective in Workbench has panels where you add Guide objects and utility objects. Each of these panels can be thought of as a type of playlist. The objects on a panel are ordered in a sequence that you determine, and each object occupies exactly one space in the sequence. The PanelItem class behaves like a column in a table, where each object from the playlist occupies one slot, or cell in the table. You can control how many cells you want the column to display. By having more than one instance of PanelItem, you can create a multicolumn experience with objects from the playlist distributed across columns.
Customizing Guides Using Flash Builder

Creating Panel Layouts

You can view an MXML example of using the PanelItem class in the section “Extending the blank panel layout” on page 20. However, understanding the concept of the PanelItem class is central to understanding panel layouts, and Guides as a whole.

For examples of using the PanelItem class, view the MXML source for the panel layouts included with the LiveCycle ES2.5 SDK. The panel layouts are available in the LiveCycle_ES_SDK\misc\Guides\code\src\ga\layouts folder where Workbench is installed.

What’s next?
In the next section, you create a simple, blank panel layout to get familiar with the process of creating panel layouts as new MXML components.

Getting started creating panel layouts

Simple panel layouts do not require extensive MXML and ActionScript. These panel layouts are a good introduction to the basic principles of working with the Guide ActionScript packages and classes.

The following information is provided to get you started creating panel layouts:

- “Creating a blank panel layout” on page 19
  Walks through creating a blank panel layout using MXML.
- “Extending the blank panel layout” on page 20
  Walks through creating a One Column panel layout beginning with the blank panel layout example.
- “Referencing your Flex library project in Workbench” on page 22
  Add your Flex library project to your Guide in the Guide Design perspective in Workbench to apply your custom panel layout.

Creating a blank panel layout

Creating a simple panel layout familiarizes you with the basic concepts, including the structure and MXML definition of a panel layout.

**Important:** Try not to edit the panel layout files that are included with LiveCycle ES2.5. Instead, make uniquely named copies of the content to help you get started creating panel layout extensions.

Create a blank panel layout

1. Start Flash Builder.
2. Create a Flex library project and configure it using the procedures in “Creating Flex library Projects for Custom Guides” on page 3. Ensure that you create the required folder structure in your Flex Library Project. To display custom panel layouts in the list of panel layouts, create all panel layouts in the *layouts* folder of your Flex library project. (The list of panel layouts is displayed in the Guide Design perspective in Workbench.)
3. Right-click the *layouts* folder and select **New > MXML Component**.
4. Type a unique filename. The Guide Design perspective modifies the name of your custom panel layout when it displays it. The perspective adds a space immediately before each uppercase character and number in the name to make it easier to read. For example, an MXML component named MyPanel.mxml appears as My Panel in the Guide Design perspective in Workbench.
5 In the Based On list, select LayoutTemplate.

6 (Optional) Set the values for **Width** and **Height** to 100%.

7 Click Finish.

The MXML source for your new panel layout looks like the following:

```xml
<?xml version="1.0" encoding="utf-8"?>
<LayoutTemplate width="100%" height="100%"
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns="ga.model.*" />
</LayoutTemplate>
```

Note the `<LayoutTemplate>` element includes the namespace attribute `xmlns="ga.model.*"`. It is considered good practice to provide namespaces when you reference objects from the Guides API. Providing namespaces both reduces the amount of MXML code and increases code readability. Updating the MXML source for the blank panel layout, the panel layout source looks like the following:

```xml
<?xml version="1.0" encoding="utf-8"?>
<ga:LayoutTemplate width="100%" height="100%"
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:ga="ga.model.*" />
</ga:LayoutTemplate>
```

This panel layout project compiles with no warnings or errors. However, this panel layout in its current state does not display any objects or content. In the next section, you extend the blank panel layout to display Guide objects and Flex components.

**Extending the blank panel layout**

After you create the shell of the new panel layout, add Flex components. You add Flex components using the Flash Builder Source view based on the behavior you are trying to achieve. In addition, you can include other Guide and Flex components to suit your specific needs. However, panel layouts are not required to contain any specific Guide components. For example, you can create a panel that contains hardcoded text objects, that a user must read before filling the Guide. For example, you can include legal text or instructions.

In this example, the blank panel layout created in the section “Creating a blank panel layout” on page 19 is extended. After it is modified, it mimics the functionality of the One Column panel layout that is included with the Guide Design perspective in Workbench.

Add the following to the blank panel layout:

- `xmlns:gc="ga.controls.*"`
  - Defines the namespace for the HelpPanel class.
- `<mx:VBox width="100%" height="100%" styleName="layoutobjects" />`
  - A standard Flex component for organizing objects into a vertical list.
- `<gc:HelpPanel id="helpPanel" styleName="panelhelp" />`
  - A region of the panel layout for displaying panel help text. The `id` attribute must be set to `helpPanel`, and the `styleName` attribute must be set to the class name for the corresponding panel help CSS style.
- `<ga:PanelItem itemInstancesPerCycle="-1" repeatItemLimit="-1" width="100%" />`
  - By default, an instance of PanelItem requires two attributes: `itemInstancesPerCycle` and `repeatItemLimit`. 
The `itemInstancesPerCycle` attribute indicates the number of Guide tree items that can fill a specified column. Therefore, setting `itemInstancesPerCycle` to the default value of 1 indicates that only one Guide tree item appears in the output. The layout then moves on to the next instance of `PanelItem` to continue laying out Guide tree items. Setting `itemInstancesPerCycle` to a value of -1 indicates that additional instances of the current `PanelItem`, or column, are added until the `repeatItemLimit` value is met.

The `repeatItemLimit` attribute indicates the maximum number of Guide tree items to add to the instance of `PanelItem`. In general, to have all the items you add to the Guide tree for a particular panel to appear in the output, set `repeatItemLimit` to -1.

In addition, this example defines `PanelItem` declaratively. That is, it defines a `PanelItem` instance in MXML. As a result, the following `<mx:Script>` ActionScript is required for Guide tree objects to correctly display.

```xml
<mx:Script>
    <![CDATA[
        import mx.core.UIComponentDescriptor;
        import ga.controls.Wrapper;

        override public function get documentDescriptor( ):UIComponentDescriptor { return
            Object(this)._documentDescriptor_; }
        override public function set documentDescriptor(
            oDescriptor:UIComponentDescriptor ):void { Object(this)._documentDescriptor_ = oDescriptor; }
    ]]>
</mx:Script>
```

Ensure your MXML source looks like the following snippet.

**Extended blank panel layout**

```xml
<?xml version="1.0" encoding="utf-8"?>
    xmlns:ga="ga.model.*"
    xmlns:gc="ga.controls.*" >

    <mx:Script>
        <![CDATA[
            import mx.core.UIComponentDescriptor;
            import ga.controls.Wrapper;

            override public function get documentDescriptor( ):UIComponentDescriptor { return
                Object(this)._documentDescriptor_; }
            override public function set documentDescriptor(
                oDescriptor:UIComponentDescriptor ):void { Object(this)._documentDescriptor_ = oDescriptor; }
        ]]>
    </mx:Script>

    <mx:VBox width="100%" height="100%" styleName="layoutobjects">
        <gc:HelpPanel id="helpPanel" styleName="panelhelp" />
        <ga:PanelItem itemInstancesPerCycle="-1" repeatItemLimit="-1" width="100%" />
    </mx:VBox>
</ga:LayoutTemplate>
```

Ensure the Flex library project builds with no warnings or errors. You can now move on to “Referencing your Flex library project in Workbench” on page 22.
Referencing your Flex library project in Workbench

After you build your Flex library project in Flash Builder, reference the compiled SWC file in the Guide Design perspective of Workbench. This reference makes your panel layout extension available.

1. Start Workbench.
2. Add your Flex library project to a LiveCycle ES2.5 application.
3. Switch to the Guide Design perspective, and open a Guide into which you want to incorporate the custom Guide layout.
4. In the Guide Tree view, select the root node.
5. In the Guide Properties view, in the Guide extensions option, click [ ].
6. Go to the SWC file for your Flex library project, and then click OK.

Your Guide extensions are now available in the Panel layout option in the Guide Properties view. Click Preview to render the Guide.

**Important:** If you edit the Flex library project after you add the project to a LiveCycle ES2.5 application, remove the reference to the project. The reference is located in the Guide extensions option (Guide Properties view). After you remove the reference, to make the changes available, add the reference again.

What’s next?

Create your own custom panel layouts, beginning with either the blank panel layout example, or by copying and modifying one of the default panel layouts. For more information about the Guides ActionScript API, see LiveCycle ES2.5 ActionScript Language Reference.

To start learning about creating custom Guide layouts, see “Creating Guide Layouts” on page 9, or for custom Guide controls, see “Creating Controls” on page 23.
Chapter 7: Creating Controls

*Controls* are Guide extensions that perform specific actions. In general, a control performs only a single action. However, that action can be as simple or as complex as you require for your specific solution.

There are two types of controls: Guide layout controls and field controls.

**Overview of Guide controls**

The different types of controls can be grouped according to their intended purpose:

**Guide layout controls:** A Guide layout references Guide layout controls, and they never appear in Workbench. Examples of Guide layout controls include the navigation panel, Next and Previous buttons, the data entry panel, and the Guide Help panel.

The Guide Design perspective in Workbench provides a tremendous amount of functionality. However, some situations require you to create specific functionality based on your solution requirements. For example, you can create Guide layout controls that allow your users to quickly move to the first or last panel. These controls are useful in a large Guide. The Next and Previous buttons that appear on a Guide are Guide layout controls that let users move forwards and backwards through a Guide.

**Field controls:** Field controls are the items that you add to a Guide panel in Workbench, such as fields, buttons, and lists. In some situations, using a different Flex object makes the user experience more engaging. For example, if the original Numeric Field object is used to store a percentage value, you can map the Numeric Field to an HSlider control. Mapping to a control creates a more intuitive data entry experience for a user.

To take full advantage of custom controls in a Guide, familiarize yourself with the Guide ActionScript packages. (See [ActionScript Language Reference](#).)

**Field controls in Workbench**

In Workbench, field controls are items that you add to a Guide panel, such as fields, buttons, and lists.

In the Guide Properties view, all the field controls for a Guide are included in the Control Type > Display As list. When you select an item from that list, additional properties appear in the Guide Properties view. Each control type has different corresponding properties. For example, if you select HSlider or VSlider, the Minimum and Maximum properties appear.

When you extend an existing field control, the corresponding additional properties appear in the Guide Properties view.

<table>
<thead>
<tr>
<th>Control type in the Guide Properties view</th>
<th>Class to extend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button</td>
<td>mx.controls.Button</td>
</tr>
<tr>
<td>Checkbox</td>
<td>mx.controls.CheckBox</td>
</tr>
<tr>
<td>Checkbox List</td>
<td>ga.controls.CheckBoxList</td>
</tr>
<tr>
<td>Date field</td>
<td>mx.controls.DateField</td>
</tr>
</tbody>
</table>
Creating Controls

<table>
<thead>
<tr>
<th>Control type in the Guide Properties view</th>
<th>Class to extend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drop-down list</td>
<td>mx.controls.ComboBox</td>
</tr>
<tr>
<td>Horizontal slider</td>
<td>mx.controls.HSlider</td>
</tr>
<tr>
<td>HRule</td>
<td>ga.controls.HRule</td>
</tr>
<tr>
<td>Image field</td>
<td>ga.uiComponents.ImageField</td>
</tr>
<tr>
<td>List box</td>
<td>mx.controls.List</td>
</tr>
<tr>
<td>Nested panel</td>
<td>ga.uiComponents.PanelComponent</td>
</tr>
<tr>
<td>Numeric stepper</td>
<td>mx.controls.NumericStepper</td>
</tr>
<tr>
<td>Radio button</td>
<td>mx.controls.RadioButton</td>
</tr>
<tr>
<td>Radio button list</td>
<td>ga.controls.RadioButtonList</td>
</tr>
<tr>
<td>Static list</td>
<td>ga.uiComponents.StaticList</td>
</tr>
<tr>
<td>Static text</td>
<td>ga.uiComponents_StaticText</td>
</tr>
<tr>
<td>Static text multiline</td>
<td>ga.uiComponents_StaticTextMultiline</td>
</tr>
<tr>
<td>Submit Guide button</td>
<td>ga.uiComponents.SubmitGuideButton</td>
</tr>
<tr>
<td>Text area</td>
<td>mx.controls.TextArea</td>
</tr>
<tr>
<td>Text input</td>
<td>ga.uiComponents.TextInputPicture</td>
</tr>
<tr>
<td>Text input comb</td>
<td>ga.uiComponents.TextInputComb</td>
</tr>
<tr>
<td>Text input mask</td>
<td>ga.uiComponents.TextInputMask</td>
</tr>
<tr>
<td>Text input symbol</td>
<td>ga.uiComponents.TextInputSymbol</td>
</tr>
<tr>
<td>Vertical slider</td>
<td>mx.controls.VSlider</td>
</tr>
</tbody>
</table>

Creating a control

You can create new controls in either MXML or ActionScript. Save field controls in the controls folder and save Guide layout controls in a different folder.

Important: Try not to edit the field control files that are included with LiveCycle ES2.5. Instead, make uniquely named copies of the content to help you get started creating field control extensions.

Create a field control extension in MXML

1. Start Flash Builder.
2. Select File > New > MXML Component.
3. Type a unique filename.
4. In the Based On list, select a component that is most similar in behavior to the component you want to create. This method lets you take advantage of existing behaviors through inheritance.
5. (Optional) Set values for Width and Height.
6. Click Finish.
Create a field control extension in ActionScript

1. Start Flash Builder.

2. Select File > New > ActionScript Class.

3. Type a unique class name.

4. In the Superclass field, specify the ActionScript class that is most similar in behavior to the component you want to create. This method lets you take advantage of existing behaviors through inheritance.

5. Click Finish.

After you create the shell of the new control, you add the desired functionality using the Source view in Flash Builder. For more information about the Guides ActionScript API that you can take advantage of while adding functionality, see ActionScript Language Reference.

Note: When you are developing controls for use on a Guide, some situations require you to import ActionScript objects. These ActionScript objects are generated automatically from the data model associated with the Guide. The data model contains ActionScript related to the behavior of the objects stored in the data model. This ActionScript can help you develop your custom control. You can access the ActionScript generated for data model objects in the C:\Documents and Settings\<username>\Application Data\Adobe\LiveCycle\ES2\Guides\generated\<LiveCycle Application>\<version>\Data Models folder. Import any required ActionScript files into your Flex library project.

What’s next?
Learn to create custom controls by studying several examples:

“Example: FirstPanelButton” on page 25
“Example: LastPanelButton” on page 27
“Example: CustomHSlider” on page 29

More Help topics
“The folder structure for the Flex library project” on page 4

Creating Guide layout controls

Guide layout controls are extensions that affect how a user navigates through the sections and panels of a Guide. The Next and Previous buttons, controlled by the NextPanelButton and PreviousPanelButton ActionScript classes respectively, are examples of Guide layout controls.

Do not save Guide layout controls in the controls folder.

The following examples illustrate how to create Guide layout controls to help users browse Guides that contain many sections and panels.

Example: FirstPanelButton

Description
This example creates a button object that, when clicked by a user at runtime, displays the first panel in the Guide. Adding this control to a Guide layout creates the following experience:

- When the Guide renders, a button labeled First is dimmed. The user cannot interact with the button immediately.
• The user navigates to another panel in the Guide using the Next button or through the navigation supplied by the Guide layout. After this navigation, the First button is displayed normally and is available to the user. Clicking the button returns the user to the first panel in the Guide.

Source

```actionscript
package com.adobe.guides.controls {
    import mx.controls.Button;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import ga.model.GAEvent;
    import ga.model.PanelManager;

    public class FirstPanelButton extends Button {
        // The PanelManager class controls the organization and behavior
        // of panel instances at runtime. The class contains
        // convenience properties and methods that are useful for
        // manipulating panels within a Guide.
        private var _pm:PanelManager;

        protected override function createChildren():void {
            super.createChildren();
            this.label = "First";
            _pm = PanelManager.instance;

            // Adds event listeners for the three events that can cause a
            // change in the state of this button, and reevaluates if the button
            // should display as enabled.
            _pm.addEventListener(GAEvent.PAGE_SELECTION_CHANGE, pageChange);
            _pm.addEventListener(GAEvent.PAGE_REMOVE, pageChange);
            _pm.addEventListener(GAEvent.PAGE_ADD, pageChange);

            // Sets the default behavior when the Guide renders. In this
            // case, on the initial panel, the button should be dimmed.
            super.enabled = false;
        }
    }
}
```
// When the button is clicked by a user, the current panel displayed
// is set to be the first panel in the Guide.
override protected function clickHandler(event:MouseEvent):void
{
    if (super.enabled)
    {
        _pm.currentPage = _pm.firstPage;
    }
}

// Conditionally sets the button's enabled property dependent on
// whether the current panel is the first panel in the Guide.
private function pageChange(event:Event):void
{
    super.enabled = _pm.previousPage!=null;
}

For information about adding this control to a Guide layout, see “Adding custom Guide layout controls to panel
layouts and Guide layouts” on page 29.

When creating custom navigation controls, be aware of the methods and properties of the PanelManager class. (See ActionScript Language Reference.)

Example: LastPanelButton

Description
This example creates a button object that, when clicked by a user at runtime, displays the last panel in the Guide.
Adding this control to a Guide layout creates the following experience:

- When the Guide renders, a button labeled Last is displayed. Clicking the button sends the user to the last panel in
the Guide.
- If the user is on the last panel of the Guide, this button is dimmed and not available to the user.
Source

package com.adobe.guides.controls
{
    import mx.controls.Button;
    import flash.events.Event;
    import flash.events.MouseEvent;
    import ga.model.GAEvent;
    import ga.model.PanelManager;

    public class LastPanelButton extends Button
    {
        // The PanelManager class controls the organization and behavior
        // of panel instances at runtime. The class contains
        // convenience properties and methods that are useful for
        // manipulating panels within a Guide.
        private var _pm:PanelManager;

        protected override function createChildren():void
        {
            super.createChildren();
            this.label = "Last";
            _pm = PanelManager.instance;

            // Adds event listeners for the three events that can cause a
            // change in the state of this button, and reevaluates whether
            // the button should display as enabled.
            _pm.addEventListener(GAEvent.PAGE_SELECTION_CHANGE, pageChange);
            _pm.addEventListener(GAEvent.PAGE_REMOVE, pageChange);
            _pm.addEventListener(GAEvent.PAGE_ADD, pageChange);

            // Sets the default behavior when the Guide renders. In this
            // case, on the initial panel, the button should be disabled.
            super.enabled = true;
        }

        // When the button is clicked by a user, the current panel displayed
        // is set to be the last panel in the Guide.
        override protected function clickHandler(event:MouseEvent):void
        {
            if (super.enabled)
            {
                _pm.currentPage = _pm.lastPage;
            }
        }

        // Conditionally sets the button's enabled property dependent on
        // whether the current panel is the last panel in the Guide.
        private function pageChange(event:Event):void
        {
            super.enabled = _pm.nextPage!=null;
        }
    }
}

For information about adding this control to a Guide layout, see “Adding custom Guide layout controls to panel layouts and Guide layouts” on page 29.
Adding custom Guide layout controls to panel layouts and Guide layouts

You can implement Guide layout controls that you want to display as part of a Guide layout or panel layout. To implement the controls, add them to the MXML definition associated with a Guide layout or panel layout.

The following MXML source illustrates how to modify the Guide layout created in the "Creating a simple Guide layout" on page 10 section. It adds the First and Last buttons created in the "Example: FirstPanelButton" on page 25 and “Example: LastPanelButton” on page 27 sections to the layout.

```xml
<?xml version="1.0" encoding="utf-8"?>
<gc:Wrapper width="100%" height="100%"
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:gc="ga.controls.*"
    xmlns:cc="com.adobe.guides.controls.*" >
    <mx:VBox width="100%" height="100%">
        <gc:PanelContent width="100%" height="100%" />
        <mx:HBox>
            <cc:FirstPanelButton />
            <gc:PreviousPanelButton label="Back" />
            <gc:NextPanelButton label="Forward" />
            <cc:LastPanelButton />
            <gc:SubmitButton label="Submit Data" />
        </mx:HBox>
    </mx:VBox>
</gc:Wrapper>
```

In this example, a new namespace com.adobe.guides.controls.* is added to simplify the references to the new controls. In addition, the bold text represents the references to the custom navigation controls.

Creating custom field controls

Field controls are custom components that replace standard Guide items to provide an enhanced user experience.

The following information illustrates how to create field controls that provide a more intuitive data entry experience for users.

Example: CustomHSlider

Description

This example creates an HSlider control that you can use for mapping Guide items. This custom field control assumes that the original Guide item is used to specify integer values from 0 through 100 that indicate a percentage. Mapping a Guide item to this control creates the following experience:

- When the Guide renders, an HSlider control appears in place of the original Guide item. The HSlider has the same caption value as the original Guide item. The user can drag the slider to set a value from 0 through 100.
- When switching to the PDF view of the form, the value the user sets using the slider appears in the corresponding field. If the user changes the value on the PDF and then returns to the Guide, the slider value reflects the updated value.
Source
package com.adobe.guides.controls
{
    import mx.controls.HSlider;

    public class CustomHSlider extends HSlider
    {

        protected override function createChildren():void
        {
            super.createChildren();

            // Sets the minimum, maximum, and initial values for the range.
            this.minimum = 0;
            this.maximum = 100;
            this.value = 0;

            // The increment range for the slider, and the increments for the
            // increment label.
            this.snapInterval = 1;
            this.tickInterval = 10;

            // The label for the range represented by the slider.
            this.labels=['0%', '100%'];

            // Sets interaction properties allowing the slider to update in
            // real-time in response to user interaction.
            this.allowTrackClick = true;
            this.liveDragging = true;
        }
    }
}

For information about adding this control to a Guide layout, see “Adding custom Guide layout controls to panel layouts and Guide layouts” on page 29.

Referencing your Flex library project in Workbench
After you build your Flex library project in Flash Builder, reference the compiled SWC file in the Guide Design perspective of Workbench. This reference makes your control extension available.

Reference your Flex library project in Workbench
1 Start Workbench.
2 Add your Flex library project to a LiveCycle ES2.5 application.
3 Switch to the Guide Design perspective, and open a Guide into which you want to incorporate the custom Guide layout.
4 In the Guide Tree view, select the root node.
5 In the Guide Properties view, in the Guide extensions option, click .
6 Go to the SWC file for your Flex library project, and then click OK.

Your Guide extensions are now available in the Display as option in the Guide Properties view. Click Preview to render the Guide.
**Important:** If you edit the Flex library project after you add the project to a LiveCycle ES2.5 application, remove the reference to the project. The reference is located in the **Guide extensions** option (**Guide Properties** view). After you remove the reference, to make the changes available, add the reference again.

**Mapping Guide objects to custom controls**

To use your custom field control on a Guide, associate an object in the Guide tree with your custom field control. Perform this mapping for each Guide object you want to associate with the new custom field control. You do not need to add the custom field control to a Guide layout or panel layout.

**Map form design objects to custom field controls**

1. Start Workbench and either open an existing Guide, or create a Guide.
2. Select the Guide object that you want to map in the Guide Tree view.
3. In the **Guide Properties** view, click **Display as**, and select the name of your custom control. For example, using the custom control created in the “Example: CustomHSlider” on page 29 section, select **Custom H Slider**.
4. Save the Guide.
   - Preview your Guide to view the new custom control.

**What’s next?**

Create your own custom controls, beginning with any of the examples, or by starting from a new MXML component or ActionScript class. For more information about the Guides ActionScript API, see [ActionScript Language Reference](#).

To start learning about creating custom panel layouts, see “Creating Panel Layouts” on page 16 or, for custom Guide layouts, see “Creating Guide Layouts” on page 9.
Chapter 8: Custom Components Reference


The custom components that you can add to layouts are listed alphabetically in the Flash Builder Components view but conceptually fall into the following categories:

- Button components
- Help components
- Label components
- Navigation components
- Output components

**Button components**

Button components provide the most common Guide actions.

**AddPanelButton:** Adds a new panel to a list of repeating panels. This button is available only when the following statements are true:

- The current panel can repeat.
- Adding a new panel does not conflict with the maximum occurrence value of the associated subform object on the form.

**CopyPanelButton:** Creates a copy of the currently selected panel and adds it to the list of repeating panels. This button is available only when the following statements are true:

- The current panel can repeat.
- Adding a copy of the current panel does not conflict with the maximum occurrence value of the associated subform object on the form.

**NextPanelButton:** Displays the next panel in the Guide tree. If the current panel is the last panel in the tree, this button is not available.

**PreviousPanelButton:** Displays the previous panel in the Guide tree. If the current panel is the first panel in the tree, this button is not available.

**RemovePanelButton:** Removes the current panel from the list of repeating panels. This button is available only when the following statements are true:

- The current panel can repeat.
- Removing the current panel does not conflict with the minimum occurrence value of the associated subform object on the form.
SubmitButton: Displays a submit button, but only when the current panel is the last panel in the Guide tree. When clicked, this button performs one of the following actions. The action depends on the values selected in the Submit From list in the Guide Properties view in Workbench:

- **Guide** (Default) Data submission depends on where the Guide is running. If the Guide is running in a browser, the submit button appears in the navigation button area, where the Next and Previous buttons appear. Typically it appears on the last visible panel in the Guide. When the Guide runs in Workspace, no submit button appears on the Guide, and the data is submitted from a button on the Workspace chrome.

- **Interactive PDF** Submits the data from an interactive PDF document. This option is available when the Guide is based on an XDP form that includes an interactive PDF form. When you select this option, a Submit From PDF button appears in the navigation button area. Clicking this button switches the view to the interactive PDF document, which must include a Submit button. The Submit From PDF button appears in all environments where the Guide can run, such as Workspace or a browser. No other submit button appears in the navigation button area of the Guide or in the Workspace chrome.

- **User Button** Submits the data from a button that you add to a panel in the Guide. This submit button appears in all environments where the Guide can run. No other submit button appears in the navigation button area of the Guide or in the Workspace chrome.

*Note: In Guides based on XDP or PDF forms, Email Submit Buttons are not supported.*

- **Host:** Specifies that data is submitted from the application that is hosting the Guide, such as Workspace. The submit button appears on the Workspace chrome, under the Guide. There is no submit button in the navigation button area of the Guide.

**Help components**

Help components let you display help to end users in text, image, and video format:

- **HelpBox:** Displays Guide help.
- **HelpCenter:** Displays a centralized region within a Guide layout to display Guide help and panel help.
- **HelpPanel:** Displays panel help.
- **HelpVideo:** Displays the help video control.

**Label components**

Label components provide objects that display section and panel titles:

- **PanelTitle:** A label that displays the name of the currently selected panel.
- **SectionTitle:** A label that displays the name of the currently selected section.

**Navigation components**

Navigation components provide the system for navigating the sections and panels of a Guide:

- **AccordionNav:** An accordion menu composed of sections that each contain a list of panels. The default Guide layout named *Left Accordion*, is an example of the AccordionNav component.
MxTreeNav: A tree structure that lists multiple section and panel levels. The default Guide layout named Cobalt Tree is an example of the MxTreeNav component.

Note: The MxTreeNav component is the only navigation component that displays nested section and panel levels.

ProgressSectionBarNav: A horizontal list of buttons that represents each section in the Guide tree. The default Guide layout named Cobalt Bar is an example of the ProgressSectionBarNav component.

Note: This navigator is useful when each Guide section includes only one panel.

StepNav: An accordion menu that lists section names where each section contains a list of panels. The default Guide layout named Cobalt Standard is an example of the usage of the StepNav component.

TabTabNav: A navigation system that consists of two corresponding levels of tab menus. The top-level tabs list the sections in the Guide tree, and the bottom-level tabs list the panels for the currently selected top-level tab. The default Guide layout named Workspace is an example of the TabTabNav component.

Output components

Output components provide objects that display content or functionality to users at runtime.

PanelContent: Displays the content of Guide panels.

ProgressBar: Indicates the percentage of mandatory fields into which a user entered data. This control is not available if the Guide does not contain mandatory fields.

ToolBar: Displays the Guide toolbar, which includes the Save PDF, Print PDF, Email PDF, and Show/Hide PDF controls. See Application Development Using LiveCycle Workbench ES2 for information on including a PDF with a Guide.
Chapter 9: Guide CSS Reference

Use this Guide CSS styles information to create a common CSS file. Share the common CSS file among multiple Guides to maintain common styling.

Guide CSS classes

.application
The .application class specifies overall style properties for a Guide application.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;backgroundAlpha&quot; on page 45</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;backgroundColor&quot; on page 45</td>
<td>#336699</td>
</tr>
<tr>
<td>&quot;backgroundGradientColors&quot; on page 46</td>
<td>#336699, #3366CC</td>
</tr>
<tr>
<td>&quot;basewrapper&quot; on page 46</td>
<td>ga.wrappers.LeftAccordion</td>
</tr>
<tr>
<td>&quot;paddingBottom&quot; on page 53</td>
<td>10</td>
</tr>
<tr>
<td>&quot;paddingLeft&quot; on page 53</td>
<td>10</td>
</tr>
<tr>
<td>&quot;paddingRight&quot; on page 54</td>
<td>10</td>
</tr>
<tr>
<td>&quot;paddingTop&quot; on page 54</td>
<td>5</td>
</tr>
<tr>
<td>&quot;version&quot; on page 58</td>
<td>1.0.0</td>
</tr>
</tbody>
</table>

.buttons
The .buttons class defines the style properties for button objects.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;alpha&quot; on page 45</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;backgroundAlpha&quot; on page 45</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;color&quot; on page 48</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td>&quot;fillAlphas&quot; on page 49</td>
<td>1.0,1.0</td>
</tr>
<tr>
<td>&quot;fillColors&quot; on page 49</td>
<td>#336699, #006699</td>
</tr>
<tr>
<td>&quot;themeColor&quot; on page 57</td>
<td>#336699</td>
</tr>
</tbody>
</table>

.fieldhelp
The .fieldhelp class specifies properties for field level help on a Guide.
### Customizing Guides Using Flash Builder

**Guide CSS Reference**

#### .guide

The `.guide` class specifies style properties for the top-level layer of the Guide application as defined in the Guide layout definition.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>backgroundColor</code></td>
<td><code>#666666</code></td>
</tr>
<tr>
<td><code>borderColor</code></td>
<td><code>#666666</code></td>
</tr>
<tr>
<td><code>color</code></td>
<td><code>#FFFFFF</code></td>
</tr>
<tr>
<td><code>dropShadowEnabled</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>gradientColors</code></td>
<td><code>#333333, #666666</code></td>
</tr>
<tr>
<td><code>fontSize</code></td>
<td><code>12</code></td>
</tr>
<tr>
<td><code>fontStyle</code></td>
<td><code>normal</code></td>
</tr>
<tr>
<td><code>fontWeight</code></td>
<td><code>normal</code></td>
</tr>
<tr>
<td><code>textDecoration</code></td>
<td><code>normal</code></td>
</tr>
</tbody>
</table>

#### .guidehelp

The `.guidehelp` class specifies properties for Guide-level help on a Guide.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>alpha</code></td>
<td><code>1.0</code></td>
</tr>
<tr>
<td><code>backgroundAlpha</code></td>
<td><code>1.0</code></td>
</tr>
<tr>
<td><code>backgroundColor</code></td>
<td><code>#FFFFFF</code></td>
</tr>
<tr>
<td><code>borderColor</code></td>
<td><code>#336699</code></td>
</tr>
</tbody>
</table>

**Last updated 10/29/2010**
The `.layoutobjects` class defines the margins around an object within a panel layout.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;paddingBottom&quot; on page 53</td>
<td>20</td>
</tr>
<tr>
<td>&quot;paddingLeft&quot; on page 53</td>
<td>25</td>
</tr>
<tr>
<td>&quot;paddingRight&quot; on page 54</td>
<td>25</td>
</tr>
<tr>
<td>&quot;paddingTop&quot; on page 54</td>
<td>25</td>
</tr>
</tbody>
</table>

The `.layoutrepeaterobjects` class defines the margins around an object within a repeating panel layout.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;paddingBottom&quot; on page 53</td>
<td>20</td>
</tr>
<tr>
<td>&quot;paddingLeft&quot; on page 53</td>
<td>25</td>
</tr>
<tr>
<td>&quot;paddingRight&quot; on page 54</td>
<td>25</td>
</tr>
<tr>
<td>&quot;paddingTop&quot; on page 54</td>
<td>25</td>
</tr>
</tbody>
</table>

The `.logo` class controls the space reserved for a logo graphic.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;paddingBottom&quot; on page 53</td>
<td>0</td>
</tr>
<tr>
<td>&quot;paddingRight&quot; on page 54</td>
<td>0</td>
</tr>
</tbody>
</table>

The `.navigationbase` class specifies that area that surrounds the navigation buttons.
<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;horizontalAlign&quot; on page 52</td>
<td>center</td>
</tr>
<tr>
<td>&quot;paddingBottom&quot; on page 53</td>
<td>10</td>
</tr>
<tr>
<td>&quot;paddingLeft&quot; on page 53</td>
<td>50</td>
</tr>
<tr>
<td>&quot;paddingRight&quot; on page 54</td>
<td>50</td>
</tr>
<tr>
<td>&quot;paddingTop&quot; on page 54</td>
<td>10</td>
</tr>
</tbody>
</table>

**.navigationlevel1**

The `.navigationlevel1` class specifies properties for the first level navigation heading.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;backgroundColor&quot; on page 45</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td>&quot;borderStyle&quot; on page 47</td>
<td>none</td>
</tr>
<tr>
<td>&quot;color&quot; on page 48</td>
<td>#0b333c</td>
</tr>
<tr>
<td>&quot;fillColors&quot; on page 49</td>
<td>#336699</td>
</tr>
<tr>
<td>&quot;fontSize&quot; on page 50</td>
<td>14</td>
</tr>
<tr>
<td>&quot;fontWeight&quot; on page 51</td>
<td>bold</td>
</tr>
<tr>
<td>&quot;rollOverColor&quot; on page 55</td>
<td>#006699</td>
</tr>
<tr>
<td>&quot;selectionColor&quot; on page 56</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td>&quot;textDecoration&quot; on page 56</td>
<td>normal</td>
</tr>
<tr>
<td>&quot;textRollOverColor&quot; on page 57</td>
<td>#333333</td>
</tr>
<tr>
<td>&quot;textSelectedColor&quot; on page 57</td>
<td>#000000</td>
</tr>
<tr>
<td>&quot;themeColor&quot; on page 57</td>
<td>#006699</td>
</tr>
</tbody>
</table>

**.navigationlevel2**

The `.navigationlevel2` class specifies properties for the second-level navigation heading.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;backgroundColor&quot; on page 45</td>
<td>#006699</td>
</tr>
<tr>
<td>&quot;color&quot; on page 48</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td>&quot;fillColors&quot; on page 49</td>
<td>#3399CC, #3399CC</td>
</tr>
<tr>
<td>&quot;fontSize&quot; on page 50</td>
<td>12</td>
</tr>
<tr>
<td>&quot;fontWeight&quot; on page 51</td>
<td>normal</td>
</tr>
<tr>
<td>&quot;rollOverColor&quot; on page 55</td>
<td>#006699</td>
</tr>
<tr>
<td>&quot;selectionColor&quot; on page 56</td>
<td>#006699</td>
</tr>
</tbody>
</table>
The `.navigationlevel3` class specifies properties for the third level navigation heading.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>backgroundColor</code> on page 45</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>color</code> on page 48</td>
<td>#0b333c</td>
</tr>
<tr>
<td><code>fillColors</code> on page 49</td>
<td>#FFFFFF, #FFFFFF</td>
</tr>
<tr>
<td><code>fontSize</code> on page 50</td>
<td>14</td>
</tr>
<tr>
<td><code>fontWeight</code> on page 51</td>
<td>bold</td>
</tr>
<tr>
<td><code>rollOverColor</code> on page 55</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>selectionColor</code> on page 56</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>textDecoration</code> on page 56</td>
<td>normal</td>
</tr>
<tr>
<td><code>textRollOverColor</code> on page 57</td>
<td>#888888</td>
</tr>
<tr>
<td><code>textSelectedColor</code> on page 57</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>themeColor</code> on page 57</td>
<td>#FFFFFF</td>
</tr>
</tbody>
</table>

The `.navigationlevel4` class specifies properties for the fourth level navigation heading.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>backgroundColor</code> on page 45</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>color</code> on page 48</td>
<td>#0b333c</td>
</tr>
<tr>
<td><code>fillColors</code> on page 49</td>
<td>#FFFFFF, #FFFFFF</td>
</tr>
<tr>
<td><code>fontSize</code> on page 50</td>
<td>14</td>
</tr>
<tr>
<td><code>fontWeight</code> on page 51</td>
<td>bold</td>
</tr>
<tr>
<td><code>rollOverColor</code> on page 55</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>selectionColor</code> on page 56</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>textDecoration</code> on page 56</td>
<td>normal</td>
</tr>
<tr>
<td><code>textRollOverColor</code> on page 57</td>
<td>#888888</td>
</tr>
<tr>
<td><code>textSelectedColor</code> on page 57</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>themeColor</code> on page 57</td>
<td>#FFFFFF</td>
</tr>
</tbody>
</table>
**.navigationlevel5**
The `.navigationlevel5` class specifies properties for the fifth level navigation heading.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>backgroundColor</code> on page 45</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>color</code> on page 48</td>
<td>#0b333c</td>
</tr>
<tr>
<td><code>fillColors</code> on page 49</td>
<td>#FFFFFF, #FFFFFF</td>
</tr>
<tr>
<td><code>fontSize</code> on page 50</td>
<td>9</td>
</tr>
<tr>
<td><code>fontWeight</code> on page 51</td>
<td>normal</td>
</tr>
<tr>
<td><code>rollOverColor</code> on page 55</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>selectionColor</code> on page 56</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>textDecoration</code> on page 56</td>
<td>normal</td>
</tr>
<tr>
<td><code>themeColor</code> on page 57</td>
<td>#FFFFFF</td>
</tr>
</tbody>
</table>

**.navigationOver**
The `.navigationOver` class specifies the border color that is used when the Guide filler moves the pointer over a navigation control.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fillColors</code> on page 49</td>
<td>#3366CC, #3366CC</td>
</tr>
</tbody>
</table>

**.navigationSelected**
The `.navigationSelected` class specifies the background color that is used when the Guide filler selects a navigation control.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fillColors</code> on page 49</td>
<td>#3366CC, #3366CC</td>
</tr>
</tbody>
</table>

**.panelcaption**
The `.panelcaption` class defines the text attributes of field captions in a panel.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>color</code> on page 48</td>
<td>#333333</td>
</tr>
<tr>
<td><code>fontSize</code> on page 50</td>
<td>12</td>
</tr>
<tr>
<td><code>fontStyle</code> on page 51</td>
<td>normal</td>
</tr>
<tr>
<td><code>fontWeight</code> on page 51</td>
<td>normal</td>
</tr>
<tr>
<td><code>textDecoration</code> on page 56</td>
<td>normal</td>
</tr>
</tbody>
</table>
.paneldata
The .paneldata class defines the properties for the area of a panel layout that displays Guide objects and data.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;backgroundAlpha&quot; on page 45</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;backgroundColor&quot; on page 45</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td>&quot;borderAlpha&quot; on page 47</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;borderColor&quot; on page 47</td>
<td>#336699</td>
</tr>
<tr>
<td>&quot;color&quot; on page 48</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td>&quot;cornerRadius&quot; on page 49</td>
<td>10</td>
</tr>
<tr>
<td>&quot;headerAlphas&quot; on page 52</td>
<td>1.0, 1.0</td>
</tr>
<tr>
<td>&quot;headerColors&quot; on page 52</td>
<td>#336699, #336699</td>
</tr>
<tr>
<td>&quot;roundedBottomCorners&quot; on page 55</td>
<td>true</td>
</tr>
</tbody>
</table>

.panelhelp
The .panelhelp class specifies a background color for the help that appears in the data entry panel.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;alpha&quot; on page 45</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;backgroundAlpha&quot; on page 45</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;backgroundColor&quot; on page 45</td>
<td>#999999</td>
</tr>
<tr>
<td>&quot;borderColor&quot; on page 47</td>
<td>#336699</td>
</tr>
<tr>
<td>&quot;borderStyle&quot; on page 47</td>
<td>solid</td>
</tr>
<tr>
<td>&quot;color&quot; on page 48</td>
<td>#000000</td>
</tr>
<tr>
<td>&quot;cornerRadius&quot; on page 49</td>
<td>5</td>
</tr>
<tr>
<td>&quot;headerAlphas&quot; on page 52</td>
<td>1.0, 1.0</td>
</tr>
<tr>
<td>&quot;headerColors&quot; on page 52</td>
<td>#336699, #336699</td>
</tr>
</tbody>
</table>

.panelitem
The .panelitem class specifies style properties for Guide fields.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;backgroundAlpha&quot; on page 45</td>
<td>1.0</td>
</tr>
<tr>
<td>&quot;backgroundColor&quot; on page 45</td>
<td>#CCCCCC</td>
</tr>
<tr>
<td>&quot;borderStyle&quot; on page 47</td>
<td>inset</td>
</tr>
<tr>
<td>&quot;color&quot; on page 48</td>
<td>#000000</td>
</tr>
<tr>
<td>&quot;dropShadowEnabled&quot; on page 49</td>
<td>false</td>
</tr>
<tr>
<td>&quot;fontSize&quot; on page 50</td>
<td>12</td>
</tr>
</tbody>
</table>
**.panelnav**

The `.panelnav` class defines the colors for the controls in the navigation panel, such as accordions or tabs.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>backgroundAlpha</code> on page 45</td>
<td>1.0</td>
</tr>
<tr>
<td><code>backgroundColor</code> on page 45</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>barColor</code> on page 46</td>
<td>#3366cc</td>
</tr>
<tr>
<td><code>borderAlpha</code> on page 47</td>
<td>1.0</td>
</tr>
<tr>
<td><code>borderColor</code> on page 47</td>
<td>#336699</td>
</tr>
<tr>
<td><code>color</code> on page 48</td>
<td>#FFFFFF</td>
</tr>
<tr>
<td><code>cornerRadius</code> on page 49</td>
<td>10</td>
</tr>
<tr>
<td><code>headerAlphas</code> on page 52</td>
<td>1.0, 1.0</td>
</tr>
<tr>
<td><code>headerColors</code> on page 52</td>
<td>#336699, #336699</td>
</tr>
<tr>
<td><code>roundedBottomCorners</code> on page 55</td>
<td>true</td>
</tr>
</tbody>
</table>

**.paneltext**

The `.paneltext` class defines text attributes of the panel text.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>color</code> on page 48</td>
<td>#000000</td>
</tr>
<tr>
<td><code>fontSize</code> on page 50</td>
<td>12</td>
</tr>
<tr>
<td><code>fontStyle</code> on page 51</td>
<td>normal</td>
</tr>
<tr>
<td><code>fontWeight</code> on page 51</td>
<td>normal</td>
</tr>
<tr>
<td><code>textDecoration</code> on page 56</td>
<td>normal</td>
</tr>
</tbody>
</table>

**.progressbar**

The `.progressbar` class defines the background colors of the progress bar.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>backgroundColor</code> on page 45</td>
<td>#336699</td>
</tr>
<tr>
<td><code>barColor</code> on page 46</td>
<td>#336699</td>
</tr>
</tbody>
</table>
.repeater
The .repeater class defines the background colors of the repeater components.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;backgroundColor&quot; on page 45</td>
<td>#336699</td>
</tr>
<tr>
<td>&quot;themeColor&quot; on page 57</td>
<td>#336699</td>
</tr>
</tbody>
</table>

.videocontrol
The .videocontrol class controls the color of the background and of text.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;backgroundColor&quot; on page 45</td>
<td>#AFAFAF</td>
</tr>
<tr>
<td>&quot;color&quot; on page 48</td>
<td>#000000</td>
</tr>
</tbody>
</table>

ComboBox
The ComboBox class specifies the default styles for all drop-down lists in a Guide.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;color&quot; on page 48</td>
<td>#000000</td>
</tr>
<tr>
<td>&quot;dropShadowEnabled&quot; on page 49</td>
<td>true</td>
</tr>
<tr>
<td>&quot;fontFamily&quot; on page 50</td>
<td>Myriad Pro</td>
</tr>
<tr>
<td>&quot;fontSize&quot; on page 50</td>
<td>11</td>
</tr>
</tbody>
</table>

DateField
The DateField class specifies the default formatting for all date fields in a Guide.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;color&quot; on page 48</td>
<td>#000000</td>
</tr>
<tr>
<td>&quot;dropShadowEnabled&quot; on page 49</td>
<td>true</td>
</tr>
<tr>
<td>&quot;fontFamily&quot; on page 50</td>
<td>Myriad Pro</td>
</tr>
<tr>
<td>&quot;fontSize&quot; on page 50</td>
<td>12</td>
</tr>
</tbody>
</table>

GAIcon
The GAIcon class specifies the format for the logo in a Guide.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;iconColors&quot; on page 53</td>
<td>#336699, #336699</td>
</tr>
<tr>
<td>&quot;iconTextColor&quot; on page 53</td>
<td>#FFFFFF</td>
</tr>
</tbody>
</table>
Label
The Label class specifies the format for static text in a Guide.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;color&quot;</td>
<td>#000000</td>
</tr>
</tbody>
</table>

RadioButton
The RadioButton class specifies formatting for all radio buttons used in a Guide.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;dropShadowEnabled&quot;</td>
<td>true</td>
</tr>
<tr>
<td>&quot;fontFamily&quot;</td>
<td>Myriad Pro</td>
</tr>
<tr>
<td>&quot;fontSize&quot;</td>
<td>12</td>
</tr>
</tbody>
</table>

TextInput
The TextInput class specifies the format for all text fields used in a Guide.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;color&quot;</td>
<td>#000000</td>
</tr>
<tr>
<td>&quot;dropShadowEnabled&quot;</td>
<td>true</td>
</tr>
<tr>
<td>&quot;fontFamily&quot;</td>
<td>Myriad Pro</td>
</tr>
<tr>
<td>&quot;fontSize&quot;</td>
<td>12</td>
</tr>
</tbody>
</table>

TextInputMask
The TextInputMask class specifies the style properties of the mask input comb field. It enables guided data entry by providing visual cues to the user filling the Guide. For example, a postal code field would display as a field with six cells.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;backgroundColor&quot;</td>
<td>#333333</td>
</tr>
<tr>
<td>&quot;cellColor&quot;</td>
<td>#CCCCCC</td>
</tr>
</tbody>
</table>

TextInputSymbol
The TextInputSymbol class specifies the format for currency symbols used in a Guide.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;prefixBackgroundColor&quot;</td>
<td>#333333</td>
</tr>
<tr>
<td>&quot;prefixColor&quot;</td>
<td>#CCCCCC</td>
</tr>
<tr>
<td>&quot;suffixBackgroundColor&quot;</td>
<td>#333333</td>
</tr>
<tr>
<td>&quot;suffixColor&quot;</td>
<td>#CCCCCC</td>
</tr>
</tbody>
</table>
Guide CSS properties

alpha
The alpha property specifies the transparency of the color of text on a button.

Syntax
alpha="[0.0..1.0]"

Applies to
• “.buttons” on page 35
• “.guidehelp” on page 36
• “.navigationlevel2” on page 38
• “.panelhelp” on page 41

backgroundAlpha
The backgroundAlpha property specifies the transparency of the background color.

Syntax
backgroundAlpha="[0.0..1.0]"

Applies to
• “.application” on page 35
• “.buttons” on page 35
• “.guide” on page 36
• “.guidehelp” on page 36
• “.navigationlevel1” on page 38
• “.paneldata” on page 41
• “.panelhelp” on page 41
• “.panelitem” on page 41
• “.panelnав” on page 42
• “TextInputMask” on page 44

backgroundColor
The backgroundColor property specifies the Guide background color.

Syntax
backgroundColor="color-name | color-rgb | color-hex | transparent"

Applies to
• “.application” on page 35
• “.fieldhelp” on page 35
backgroundGradientColors
The `backgroundGradientColors` property specifies the two color extremes of the background color gradient.

Syntax
```
backgroundGradientColors="[color-name | color-rgb | color-hex | transparent],
                           [color-name | color-rgb | color-hex | transparent]"
```

Applies to
```
".application" on page 35
```

barColor
The `barColor` property specifies the color of the progress bar.

Syntax
```
barColor="[color-name | color-rgb | color-hex | transparent],
              [color-name | color-rgb | color-hex | transparent]"
```

Applies to
```
".guide" on page 36
".panelnav" on page 42
".progressbar" on page 42
```

basewrapper
The `basewrapper` property specifies the name of the Guide layout.

Last updated 10/29/2010
Syntax
basewrapper="[Guide layout name]"

Applies to
“.application” on page 35

borderAlpha
The borderAlpha property specifies the standard Flex style for a panel.

Syntax
borderAlpha="0.0..1.0"

Applies to
“.guide” on page 36
“.guidehelp” on page 36
“.paneldata” on page 41
“.panelnav” on page 42

borderColor
The borderColor property specifies the color of the border around field help.

Syntax
borderColor="color-name | color-rgb | color-hex | transparent"

Applies to
“.fieldhelp” on page 35
“.guide” on page 36
“.guidehelp” on page 36
“.paneldata” on page 41
“.panelhelp” on page 41
“.panelnav” on page 42

borderStyle
The borderStyle property specifies the required default border value used by the supplied layouts.

Syntax
borderStyle=""

Applies to
“.guide” on page 36
“.navigationlevel1” on page 38
cellColor

The cellColor property specifies the style attribute for the TextInputMask.

Syntax

cellColor="[color-name | color-rgb | color-hex | transparent],
   [color-name | color-rgb | color-hex | transparent]"

Applies to

"TextInputMask" on page 44

color

The color property specifies the color to use for text in Guide help.

Syntax

color="[color-name | color-rgb | color-hex | transparent]"

Applies to

".buttons" on page 35
".fieldhelp" on page 35
".guidehelp" on page 36
".navigationlevel1" on page 38
".navigationlevel2" on page 38
".navigationlevel3" on page 39
".navigationlevel4" on page 39
".navigationlevel5" on page 40
".panelcaption" on page 40
".paneldata" on page 41
".panelhelp" on page 41
".panelitem" on page 41
".panellnav" on page 42
".paneltext" on page 42
".videocontrol" on page 43
"ComboBox" on page 43
"DateField" on page 43
"Label" on page 44
**cornerRadius**

The `cornerRadius` property specifies the roundness of the panel border.

**Syntax**

```css
cornerRadius="[0..80]"
```

**Applies to**

- `.guide` on page 36
- `.guidehelp` on page 36
- `.paneldata` on page 41
- `.panelhelp` on page 41
- `.panelnav` on page 42

**dropShadowEnabled**

The `dropShadowEnabled` property specifies whether to add a drop shadow effect to field help.

**Syntax**

```css
dropShadowEnabled="true | false"
```

**Applies to**

- `.fieldhelp` on page 35
- `.panelitem` on page 41
- `.ComboBox` on page 43
- `.DateField` on page 43
- `.RadioButton` on page 44
- `.TextInput` on page 44

**fillAlphas**

The `fillAlphas` property specifies the transparency of the gradient colors defined by the `fillColors` property.

**Syntax**

```css
fillAlphas="[0..1.0],[0..1.0]"
```

**Applies to**

- `.buttons` on page 35

**fillColors**

The `fillColors` property specifies the two color extremes of the color gradient to use for button color.
Syntax
fillColors="[color-name | color-rgb | color-hex | transparent],
          [color-name | color-rgb | color-hex | transparent]"

Applies to
“.buttons” on page 35
“.navigationlevel1” on page 38
“.navigationlevel2” on page 38
“.navigationlevel3” on page 39
“.navigationlevel4” on page 39
“.navigationlevel5” on page 40
“.navigationOver” on page 40
“.navigationSelected” on page 40

**fontFamily**
The `fontFamily` property specifies the font typeface.

Syntax
`fontFamily="[font name]"`

Applies to
“ComboBox” on page 43
“DateField” on page 43
“RadioButton” on page 44
“TextInput” on page 44

**fontSize**
The `fontSize` property specifies the font size.

Syntax
`fontSize="integer"`

Applies to
“.fieldhelp” on page 35
“.navigationlevel1” on page 38
“.navigationlevel2” on page 38
“.navigationlevel3” on page 39
“.navigationlevel4” on page 39
“.navigationlevel5” on page 40
“.panelcaption” on page 40
**fontStyle**
The `fontStyle` property specifies whether to use italicized font.

**Syntax**
```css
fontStyle="normal | italic"
```

**Applies to**
```
".fieldhelp" on page 35
".panelcaption" on page 40
".panelitem" on page 41
".paneltext" on page 42
```

**fontWeight**
The `fontWeight` property specifies whether to use font bolding.

**Syntax**
```css
fontWeight="normal | bold"
```

**Applies to**
```
".fieldhelp" on page 35
".navigationlevel1" on page 38
".navigationlevel2" on page 38
".navigationlevel3" on page 39
".navigationlevel4" on page 39
".navigationlevel5" on page 40
".panelcaption" on page 40
".panelitem" on page 41
".paneltext" on page 42
```

**gradientColors**
The `gradientColors` property specifies two colors for the Guide container. Guides use the two colors to create a gradient fill.

Last updated 10/29/2010
Syntax
gradientColors="[color-name | color-rgb | color-hex | transparent],
           [color-name | color-rgb | color-hex | transparent]"

Applies to
“.fieldhelp” on page 35

headerAlphas
The headerAlphas property controls the transparency of the title area.

Syntax
headerAlphas=""

Applies to
“.guidehelp” on page 36
“.paneldata” on page 41
“.panelhelp” on page 41
“.panelnav” on page 42

headerColors
The headerColors property specifies the color of the Guide help title area.

Syntax
headerColors="[color-name | color-rgb | color-hex | transparent],
               [color-name | color-rgb | color-hex | transparent]"

Applies to
“.guidehelp” on page 36
“.paneldata” on page 41
“.panelhelp” on page 41
“.panelnav” on page 42

headerHeight
The headerHeight property specifies title bar size for panel areas (auto defined).

Syntax
headerHeight="[integer]"

Applies to
“.guide” on page 36

horizontalAlign
The horizontalAlign property specifies the standard Flex style for most containers.
Syntax
horizontalAlign=""

Applies to
".navigationbase" on page 37

**iconColors**
The iconColors property specifies a pair of fill colors for the icon component. The icon component consists of a sphere with a gradient fill and the letter “i” in the middle. The icon component is used for the information icon.

Syntax
iconColors="[color-name | color-rgb | color-hex | transparent],
            [color-name | color-rgb | color-hex | transparent]"

Applies to
“GAIIcon” on page 43

**iconTextColor**
The iconTextColor property specifies the color of the text inside the gradient sphere of the icon component.

Syntax
iconTextColor="[color-name | color-rgb | color-hex | transparent]"

Applies to
“GAIIcon” on page 43

**paddingBottom**
The paddingBottom property specifies the border around the bottom edge of the Guide.

Syntax
paddingBottom="[0..80]"

Applies to
“.application” on page 35
“.guide” on page 36
“.layoutobjects” on page 37
“.layoutrepeaterobjects” on page 37
“.logo” on page 37
“.navigationbase” on page 37

**paddingLeft**
The paddingLeft property specifies the border around the left edge of the Guide.
Syntax
paddingLeft="[0..80]"

Applies to
“.application” on page 35
“.guide” on page 36
“.layoutobjects” on page 37
“.layoutrepeaterobjects” on page 37
“.navigationbase” on page 37

paddingRight
The paddingRight property specifies the border around the right edge of the Guide.

Syntax
paddingRight="[0..80]"

Applies to
“.application” on page 35
“.guide” on page 36
“.layoutobjects” on page 37
“.layoutrepeaterobjects” on page 37
“.logo” on page 37
“.navigationbase” on page 37

paddingTop
The paddingTop property specifies the border around the top edge of the Guide.

Syntax
paddingTop="[0..80]"

Applies to
“.application” on page 35
“.guide” on page 36
“.layoutobjects” on page 37
“.layoutrepeaterobjects” on page 37
“.navigationbase” on page 37

prefixBackgroundColor
The prefixBackgroundColor property specifies the background color of the prefix area of the TextInputSymbol component.
Syntax
prefixBackgroundColor="[color-name | color-rgb | color-hex | transparent]"

Applies to
“TextInputSymbol” on page 44

prefixColor
The prefixColor property specifies the color of the text inside the prefix area of the TextInputSymbol component.

Syntax
prefixColor="[color-name | color-rgb | color-hex | transparent]"

Applies to
“TextInputSymbol” on page 44

rollOverColor
The rollOverColor property specifies the border color that is used when the Guide filler moves the pointer over a navigation control.

Syntax
rollOverColor="[color-name | color-rgb | color-hex | transparent]"

Applies to
“.navigationlevel1” on page 38
“.navigationlevel2” on page 38
“.navigationlevel3” on page 39
“.navigationlevel4” on page 39
“.navigationlevel5” on page 40

roundedBottomCorners
The roundedBottomCorners property specifies whether the panel border is rounded.

Syntax
roundedBottomCorners="true | false"

Applies to
“.guidehelp” on page 36
“.paneldata” on page 41
“.panelnav” on page 42
**selectionColor**

The `selectionColor` property specifies the background color that is used when the form filler selects a navigation control.

**Syntax**

```
selectionColor="[color-name | color-rgb | color-hex | transparent]"
```

**Applies to**

```
.navigationlevel1" on page 38
.navigationlevel2" on page 38
.navigationlevel3" on page 39
.navigationlevel4" on page 39
.navigationlevel5" on page 40
```

**suffixBackgroundColor**

The `suffixBackgroundColor` property specifies the background color of the suffix area of the TextInputSymbol component.

**Syntax**

```
suffixBackgroundColor="[color-name | color-rgb | color-hex | transparent]"
```

**Applies to**

```
"TextInputSymbol" on page 44
```

**suffixColor**

The `suffixColor` property specifies the color of the text inside the suffix area of the TextInputSymbol component.

**Syntax**

```
suffixColor="[color-name | color-rgb | color-hex | transparent]"
```

**Applies to**

```
"TextInputSymbol" on page 44
```

**textDecoration**

The `textDecoration` property specifies whether to use font underlining.

**Syntax**

```
textDecoration="normal | underline"
```

**Applies to**

```
.fieldhelp" on page 35
"guidehelp" on page 36
.navigationlevel1" on page 38
```

---

Last updated 10/29/2010
textRollOverColor

The `textRollOverColor` property specifies the text color that is used when a form filler moves the pointer moves over a navigation control.

Syntax

```
textRollOverColor="[color-name | color-rgb | color-hex | transparent]"
```

Applies to

```
.navigationlevel1" on page 38
.navi...level5" on page 40
```

textSelectedColor

The `textSelectedColor` property specifies the text color of an element of a component when the element is selected.

Syntax

```
textSelectedColor="[color-name | color-rgb | color-hex | transparent]"
```

Applies to

```
.navigationlevel1" on page 38
.navi...level5" on page 40
```

themeColor

The `themeColor` property specifies the theme color to use for buttons on a Guide.

Syntax

```
themeColor="[color-name | color-rgb | color-hex | transparent],
            [color-name | color-rgb | color-hex | transparent]"
```
version

The `version` property specifies the version of the CSS file.

*Note: This property is for information purposes only.*

**Syntax**

```plaintext
version="[version number]"
```

**Applies to**

“.application” on page 35
Index

A
AccordionNav component 33
ActionScript
control, creating 23
AddPanelButton component 32

B
building Flex library projects 5
button components 32

C
controls
creating 23
CopyPanelButton component 32
creating
controls 23
Flex library projects 3

D
documentation resources 1

F
Flex Builder 3
Flex library projects
building 5
creating 3
folder structure 4
folder structure, Flex library project 4
form guide layouts
about 9
form guides
creating Flex library projects for 3

G
Guide Builder, making SWC files available
to 5

H
HelpBox component 33
HelpPanel component 33
HelpVideo component 33

L
label components 33

M
MXML control, creating 23
MxTreeNav component 33

N
navigation components 33
NextPanelButton component 32

O
output components 34

P
PanelContent output component 34
PanelTitle label component 33
PDFBox output component 34
PreviousPanelButton component 32
ProgressBar output component 34
ProgressSectionBarNav component 33

R
RemovePanelButton component 32
resources, documentation 1

S
SectionTitle label component 33
SubmitButton component 32
SWC files
adding to a Flex library project 3
importing into Guide Builder 5

T
TabTabNav component 33
ToolBar output component 34