Customizing Guides Using Flex® Builder™

Adobe® LiveCycle® ES2

Version 9

April 2010

© 2010 Adobe Systems Incorporated. All rights reserved.

Adobe® LiveCycle® ES2 (9.0) Customizing Guides Using Flex® Builder™ for Microsoft® Windows®

Edition 3.1, April 2010

This guide is licensed for use under the terms of the Creative Commons Attribution Non-Commercial 3.0 License. This License allows users to copy, distribute, and transmit the guide for noncommercial purposes only so long as (1) proper attribution to Adobe is given as the owner of the document; and (2) any reuse or distribution of the guide contains a notice that use of the guide is governed by these terms. The best way to provide notice is to include the following link. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/.

Adobe, the Adobe logo, Adobe Reader, Acrobat, Flex, Flex Builder, and LiveCycle are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Contents

Who should read this document? 5
Additional information 5

1. About Customizing Guides Using Flex Builder

2. Creating Flex Library Projects for Custom Guides
Setting up your development environment 7
Creating a new Flex library project 7
The folder structure for the Flex library project 8
Importing sample Flex library projects 8
Building your custom Flex library project 9
What's next? 9

3. Creating Custom Style Sheets
Creating a style sheet SWF file using Flex Builder 10
Creating a style sheet SWF file using a command-line compiler 10
Referencing custom style sheets in a Guide 11
What's Next? 11

4. Creating Guide Layouts
Overview of Guide layouts 12
Getting started creating Guide layouts 13
Button Bar Guide layout 15
Binding To A Data Model 16
What's next? 16

5. Creating Panel Layouts
Overview of panel layouts 17
Getting started creating panel layouts 19
What's next? 22

6. Creating Controls
Overview of Guide controls 23
Creating a Guide control 23
Creating navigation controls 24
Creating custom field controls 27
What's next? 29

7. Custom Extension Lists
Button components 30
Help components 31
Label components 31
Navigation components 31
Output components 31

8. Guide CSS Styles

Guide CSS classes 33
Guide CSS properties 43

About This Document

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

Who should read this document?

This document is intended for Flex developers who are interested in learning how to create custom Guide extensions to extend the Guide components shipped with Workbench ES2, or Guide extensions to meet specific needs.

A knowledge of Guides, Workbench ES2, and Flex Builder is assumed.

Additional information

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

For information about	See
The starting point for learning about Guides. This document includes a walkthrough of creating a Guide, and an example of how to render and deploy a Guide to Adobe Workspace ES2 by using processes created in Workbench ES2. Creating and editing Guides using the Guide Design perspective within Workbench ES2. Rendering and deploying Guides using processes created in Workbench ES2. Debugging Guides.	Creating Guides Using LiveCycle Workbench ES2, also available within Workbench ES2
The ActionScript™ classes and properties included with LiveCycle ES2.	LiveCycle ES2 ActionScript Language
LiveCycle ES2 terminology	LiveCycle ES2 Glossary
Other services and products that integrate with LiveCycle ES2	www.adobe.com
Patch updates, technical notes, and additional information on this product version	LiveCycle Technical Support

1. About Customizing Guides Using Flex Builder

Using Flex Builder, you can go beyond the Guide customizing options available in the Guide Design perspective in Workbench ES2 to create Guide extensions designed to suit your specific needs. For example, a Flex developer can create a custom 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 Guide controls, follows the same general steps:

1. Create a Flex Library project in Flex Builder (see "Creating Flex Library Projects for Custom Guides" on page 7).
2. Create Guide extensions as part of the Flex Library project. See one of the following:
   - "Creating Custom Style Sheets" on page 10
   - "Creating Guide Layouts" on page 12
   - "Creating Panel Layouts" on page 17
   - "Creating Controls" on page 23
3. Compile the Flex library project to an SWC file (see "Building your custom Flex library project" on page 9).
4. Import the SWC file or SWF into the Guide Designer perspective in Workbench ES2 (see one of "Referencing custom style sheets in a Guide" on page 11, "Referencing your Flex library project in Workbench ES2" on page 15, "Referencing your Flex library project in Workbench ES2" on page 22, or "Referencing your Flex library project in Workbench ES2" on page 28).
5. Apply the new Guide extensions to a Guide.

2. Creating Flex Library Projects for Custom Guides

This chapter outlines how to configure your Flex development environment and how to create Flex library projects for Guide extensions.

Setting up your development environment

Before you can create custom Guide layouts, panel layouts, and controls, first 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 ES2 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

- Flex Builder 3 Standard, Flex Builder 3 Professional, or Flex Builder 3 Plug-in for Eclipse
- Workbench ES2

See the system requirements for Flex Builder and for Workbench ES2.

Note: *The term Flex Builder in this document refers to all three versions of Flex Builder.*

Creating a new Flex library project

The first step in creating Guide extensions is to create a Flex library project in Flex 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 ES2. 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 8.

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

To create a Flex library project for custom Guides:

1. Start Flex 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\libraries folder where Workbench ES2 is installed. By default, the path is C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK\misc\Guides\libraries. 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. When you compile the project to an SWC file and import the SWC file into the Guide Design perspective of Workbench ES2, Guide extensions are loaded from the project folders and made available through the Guide Design interface.

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*.

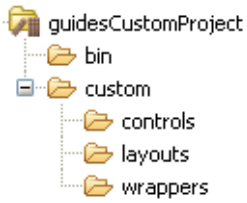
  Note: *The folder cannot be named ga.*
- At least one nested subfolder that has one of the following names:

  controls (Guide controls, such as panel navigation)

  layouts (panel layouts)

  wrappers (Guide layouts)

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



A basic folder structure that includes subfolders for all Guide extensions.

Importing sample Flex library projects

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

Caution: *Rather than editing the sample files directly, or modifying those files that are included with LiveCycle ES2, it is a best practice to make uniquely named copies of the content to help you get started creating Guide extensions.*

To import the sample Guide extensions Flex Builder project:

1 Start Flex Builder.

2 Click **File** > **Import** > **Flex Project**.
3 Select **Project Folder** and then click **Browse**. Select the LiveCycle_ES_SDK\misc\Guides\extensions folder located where you installed LiveCycle ES2 (by default, C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK\misc\Guides\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, you can compile it and add the compiled extension library to your Guide using the Guide Design perspective in Workbench ES2.

Building your custom Flex library project

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

To compile your Guide extension Flex library project:

1 Start Flex 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 application using Workbench ES2.

For more information on adding resources to a LiveCycle ES2 application, see Application Development Using LiveCycle Workbench ES2.

Renaming project components

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

To 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 Flex Builder to build automatically, manually rebuild your Flex library project.

What's next?

After you import the sample Guide projects, you can start creating your own Guide layouts. (See "Creating Guide Layouts" on page 12.)

3. Creating Custom Style Sheets

Guides supports the ability to use style sheets (CSS), compiled as SWFs 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 Styles" on page 33.

2 Do one of the following:

- Add a CSS file to Flex project, and then right-click the CSS file and ensure that the **Compile CSS to SWF** option is selected (see "Creating a style sheet SWF file using Flex Builder" on page 10).

- or -

- Compile the CSS to a SWF file using the MXML compiler included with the Flex SDK available with Workbench ES2 (see "Creating a style sheet SWF file using a command-line compiler" on page 10).

3 Add the generated style sheet SWF file to a LiveCycle ES2 application in Workbench ES2 and reference the style sheet SWF file in a Guide (see "Referencing custom style sheets in a Guide" on page 11).

Creating a style sheet SWF file using Flex Builder

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

To create a style sheet SWF file using Flex Builder:

1 Start Flex Builder.

2 Select **File** > **New** > **Flex 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 **Finish**.

5 Add a CSS containing Guide styles, as well as any assets referenced (such as images), to your Flex project.

6 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 application in Workbench ES2. For more information on adding resources to a LiveCycle ES2 application, see "Referencing custom style sheets in a Guide" on page 11.

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 ES2.

To create a style sheet SWF file using a command-line compiler:

1 On the machine where Workbench ES2 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 ES2 install location. By default, the path is C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2.

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 application in Workbench ES2. For more information on adding resources to a LiveCycle ES2 application, see "Referencing custom style sheets in a Guide" on page 11.

Referencing custom style sheets in a Guide

After you build your Flex library project in Flex Builder, reference the compiled SWC file in the Guide Design perspective of Workbench ES2 to make your panel layout extension available.

To import your custom style sheet in Workbench ES2:

1. Start Workbench ES2.
2. Add your custom style sheet to a LiveCycle ES2 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, in the **Guide extensions** option, click .
6. 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.

Caution: *If you alter a custom style sheet (SWF file) after you add it to a LiveCycle ES2 application,remove the reference to the SWF file in the* ***Guide extensions*** *option (****Guide Properties*** *view). Once 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 12, "Creating Panel Layouts" on page 17, and "Creating Controls" on page 23

4. Creating Guide Layouts

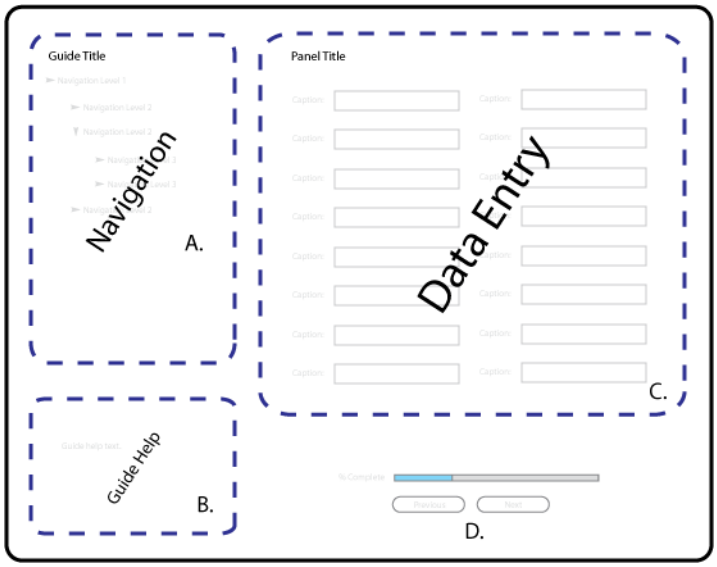
A *Guide layout* defines the visual layout and structure of a Guide that remains constant throughout a data capture session. The Guide Design perspective of Workbench ES2 includes default Guide layouts. The layouts are designed to help you quickly create Guides that are structured in visually appealing and meaningful ways. However, using Flex 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 LiveCycle ES2 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.



A. *Navigators display the Guide navigation control.* ***B.*** *Guide help text entered by a form author in Guide Builder.* ***C.*** *The content for each panel that is defined in the Guide Tree view. Each data entry panel defines its own individual layout.* ***D.*** *Previous and Next buttons for navigating the Guide, and a progress bar to indicate the percentage of mandatory fields into which the user entered data.*

Getting started creating Guide layouts

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

- "Creating a simple Guide layout" on page 13

  Walks through creating a basic Guide layout using MXML.
- "Referencing your Flex library project in Workbench ES2" on page 15

  Add your Flex library project to your LiveCycle ES2 application in Workbench ES2 to make the custom panel layout available on your Guide.

Creating a simple Guide layout

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

Caution: *Rather than editing the Guide layout files that are included with LiveCycle ES2, it is a best practice to make uniquely named copies of the content to help you get started creating Guide layout extensions.*

To create a simple Guide layout:

1. Start Flex Builder.
2. Create a Flex library project and configure it using the procedures in "Creating Flex Library Projects for Custom Guides" on page 7. Ensure that you create the required folder structure in your Flex library project. For custom Guide layouts to display in the list of Guide layouts in the Guide Design perspective of Workbench ES2, create Guide layouts in the `wrappers` folder in your Flex library project.
3. Right-click the `wrappers` folder and select **New** > **MXML Component**.
4. Type a unique filename. By default, the Guide Design perspective of Workbench ES2 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 ES2.
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 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, both to reduce the amount of MXML code, and to increase code readability. Updating the MXML source for the blank panel layout, the panel layout source looks like the following:

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

</gc:Wrapper>
```

Adding content to your Guide layout

After you create the shell of the new guide layout, you add Flex Builder components using the Flex 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.

In this example, the blank panel layout created in the topic "Creating a simple Guide layout" on page 13 is extended to include an area for displaying panel content, as well as some navigation buttons to move between panels and submit the Guide data.

Add the following to the blank panel layout:

- `<mx:VBox width="100%" height="100%">`
  A standard Flex component for organizing objects into a vertical list.
- `<gc:PanelContent width="100%" height="100%" />`
  The region of the Guide layout for displaying the data entry panel and the associated layout.
- `<mx:HBox>`
  A standard Flex component for organizing objects into a horizontal list.
- `<gc:PreviousPanelButton label="Back" />`
  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:NextPanelButton label="Forward" />`
  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. The `label` attribute controls the button caption text.
- `<gc:SubmitButton label="Submit Data" />`
  A button object that submits the Guide data. This object is inactive if the current panel is the first panel in the Guide. The `label` attribute controls the button caption text.

  Note: *Specifying a value for the `label` attribute overwrites the default SubmitButton labels that are added based on the submission option selected in the Guide Design perspective of Workbench ES2.*

  The SubmitButton component behaves differently depending on the submission options the Guide author sets for the Guide in the Guide Design perspective of Workbench ES2. The following table outlines the behavior of the SubmitButton component for each submission option:

Submit option	SubmitButton behavior	Default SubmitButton label
Guide	Users must submit the data by clicking the submit button on the Guide.	Submit
Hosted Application	The SubmitButton component does not appear on the Guide. The hosted application, such as Workspace ES2, must extract the data from the Guide and perform the data submission.	N/A

Your MXML source looks like the following snippet.

```
<?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:VBox width="100%" height="100%">
        <gc: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>
```

Your Flex library project should build with no warnings or errors. You can now move on to "Referencing your Flex library project in Workbench ES2" on page 15.

Referencing your Flex library project in Workbench ES2

After you build your Flex library project in Flex Builder, reference the compiled SWC file in the Guide Design perspective of Workbench ES2 to make your panel layout extension available.

To import your Flex library component (SWC) in Workbench ES2:

1. Start Workbench ES2.
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.
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 **Guide layouts** option in the **Guide Properties** view. Click **Preview** to render the Guide.

Caution: *If you make changes to Flex library project after you add the project to a LiveCycle ES2 application, you must remove the reference to the project in the* ***Guide extensions*** *option (****Guide Properties*** *view). Once you remove the reference, re-add it for the changes to be available.*

What's next?

Try creating one of your own custom Guide layouts, either by starting with a new blank layout, or by using the MXML source for one of the Guide layouts included with Workbench ES2 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 15.

Button Bar Guide layout

To better understand how to structure Guide layouts, this document uses the Button Bar Guide layout included in the LiveCycle_ES_SDK\misc\Guides\defaults\wrappers\ga\wrappers folder where Workbench ES2 is installed as an example. In addition to the structure, the Button Bar Guide layout contains the following custom ActionScript variables that 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, which 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 to 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 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%">
        <gc: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 `{data provider}`.

What's next?

Create your own custom Guide layouts, beginning with either the simple Guide layout created in the section "Creating a simple Guide layout" on page 13, or by copying and modifying one of the default Guide panel layouts. For more information about the Guides ActionScript API, see LiveCycle ES2 ActionScript Language Reference.

To start learning about creating custom panel layouts, see "Creating Panel Layouts" on page 17, or for custom Guide controls, see "Creating Controls" on page 23.

5. 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 ES2 includes default panel layouts designed to help you quickly create Guides with panels that are structured in visually appealing and meaningful ways. Using Flex Builder, you can create new panel layouts to structure panel content in new ways or to 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 ES2. The following are the most common Guide placeholder components that appear on panel layouts:

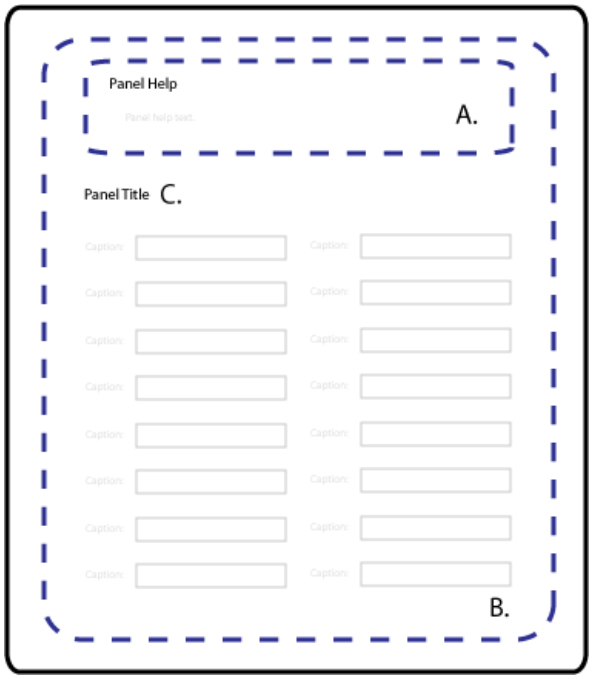
PanelItem: A placeholder object that displays an item, or a repeating sequence of items, from the Guide tree; either a field,text object, or button.

PanelText: Corresponds to the Guide Text object in the Guide Design perspective in Workbench ES2. Each `PanelText` object is contained within a `PanelItem`.

PanelTitle: A placeholder object that displays the name of the current panel. The panel title is specified in the Guide Design perspective in Workbench ES2.

HelpPanel: Displays panel help text to a form filler. The panel help text is specified in the Guide Design perspective in Workbench ES2.

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 theGuide Design perspective in Workbench ES2. Depending on the Guide layout, the panel help can appear as part of the panel, or 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 Builder components.* ***C.*** *Panel title text (PanelTitle) for the panel that a form author enters using the Guide Design perspective in Workbench ES2.*

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

For example, examine the MXML source for the One Column panel layout:

```
<?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.*"
    xmlns:gc="ga.controls.*">

    <mx:Metadata>
        [IconFile("assets/GuideComponentDisabled.png")]
    </mx:Metadata>

    <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>
```

Later in this chapter, you will walk through creating MXML source in more detail. However, it is important to notice that 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 Flex 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.
- `<gc:HelpPanel>` and `<ga:PanelItem>`

  Placeholder components for the panel help and Guide objects respectively. Each of these components is discussed in more detail later in this chapter, but understanding `PanelItem` is critical for creating custom panel layouts.

Understanding the PanelItem class

Each panel in the Guide Tree view in the Guide Design perspective in Workbench ES2, where you add Guide objects and utility objects, 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.

You can view an MXML example of using the `PanelItem` class in the section "Extending the blank panel layout" on page 21, but 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 SDK, which are available in the LiveCycle_ES_SDK\misc\Guides\defaults\layouts\ga\layouts folder where Workbench ES2 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, and are a good introduction to the basic principles of working with the Guide ActionScript packages and classes.

This section provides the following information to get you started creating panel layouts:

- "Creating a blank panel layout" on page 20

  Walks through creating a blank panel layout using MXML.

- "Extending the blank panel layout" on page 21

  Walks through creating a One Column panel layout beginning with the blank panel layout example.
- "Referencing your Flex library project in Workbench ES2" on page 22

  Add your Flex library project to your Guide in the Guide Design perspective in Workbench ES2 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.

Caution: *Rather than editing the panel layout files that are included with LiveCycle ES2, it is a best practice to make uniquely named copies of the content to help you get started creating panel layout extensions.*

To create a blank panel layout:

1. Start Flex Builder.
2. Create a Flex library project and configure it using the procedures in "Creating Flex Library Projects for Custom Guides" on page 7. Ensure that you create the required folder structure in your Flex Library Project. For custom panel layouts to display in the list of panel layouts in the Guide Design perspective in Workbench ES2, you must create all panel layouts in the `layouts` folder of your Flex library project.
3. Right-click the `layouts` folder and select **New** > **MXML Component**.
4. Type a unique filename. When displaying the name of your custom panel layout, the Guide Design perspective in Workbench ES2 adds a space immediately before each uppercase character and number in the name of your component for readability. For example, an MXML component named MyPanel.mxml appears as My Panel in the Guide Design perspective in Workbench ES2.
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 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, both to reduce the amount of MXML code, and to increase code readability. Updating the MXML source for the blank panel layout, the panel layout source looks like the following:

```
<?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>
```

Your panel layout project should compile 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 form design objects and Flex components.

Extending the blank panel layout

After you create the shell of the new panel layout, you add Flex Builder components using the Flex 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, such as legal text or instructions, that a user must read before filling the Guide.

In this example, the blank panel layout created in the section "Creating a blank panel layout" on page 20 is extended to mimic the functionality of the One Column panel layout that is included with the Guide Design perspective in Workbench ES2.

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, if you want all the items you add to the Guide tree for a particular panel to appear in the output, this attribute should be set to a value of `-1`.

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

```
    <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>
```

Your MXML source should look like the following snippet.

Extended blank panel layout

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

Your Flex library project should build with no warnings or errors. You can now move on to "Referencing your Flex library project in Workbench ES2" on page 22.

Referencing your Flex library project in Workbench ES2

After you build your Flex library project in Flex Builder, reference the compiled SWC file in the Guide Design perspective of Workbench ES2 to make your panel layout extension available.

To reference your Flex library project in Guide Builder:

1. Start Workbench ES2.
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.
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.

Caution: *If you make changes to Flex library project after you add the project to a LiveCycle ES2 application, you must remove the reference to the project in the* ***Guide extensions*** *option (****Guide Properties*** *view). Once you remove the reference, re-add it for the changes to be available.*

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 ActionScript Language Reference.

To start learning about creating custom Guide layouts, see "Creating Guide Layouts" on page 12, or for custom Guide controls, see "Creating Controls" on page 23.

6. Creating Controls

Controls are Guide extensions that perform specific actions. For example, the Next and Previous buttons that display on a Guide are navigation controls that let users move forwards and backwards through a Guide. 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.

Overview of Guide controls

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

Navigation controls: The Guide Design perspective in Workbench ES2 provides a tremendous amount of functionality. However, in some situations you may want to create specific functionality based on your solution requirements. For example, you may want to create navigation controls that allow your users to quickly move to the first or last panel in a large Guide.

Field controls: In some situations, using a different Flex object may make for a user experience that is more engaging. For example, if the original Numeric Field object is used to store a percentage value, mapping the Numeric Field to an HSlider 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 LiveCycle ES2 ActionScript Language Reference.)

Creating a Guide control

You can create new Guide controls in either MXML or ActionScript.

Caution: *Rather than editing the Guide control files that are included with LiveCycle ES2, it is a best practice to make uniquely named copies of the content to help you get started creating Guide control extensions.*

To create a Guide control extension in MXML:

1. Start Flex 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 lets you take advantage of existing behaviors through inheritance.
5. (Optional) Set values for **Width** and **Height**.
6. Click **Finish**.

To create a Guide control extension in ActionScript:

1. Start Flex 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 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 Flex Builder. For more information about the Guides ActionScript API that you can take advantage of while adding functionality, see LiveCycle ES2 ActionScript Language Reference.

Note: *When developing controls for use on a Guide, you may need to import the ActionScript objects that 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, and you may need this ActionScript to help in developing your custom control. You can access the ActionScript generated for data model objects in the C:\Documents and Settings\<username>\Application Data\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 24
- "Example: LastPanelButton" on page 25
- "Example: CustomHSlider" on page 27

Creating navigation controls

Navigation 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 navigation controls.

In this section, we explore how to create navigation 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 should create the following experience:

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

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 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 navigation controls to panel layouts and Guide layouts" on page 27.

When creating custom navigation controls, be aware of the methods and properties of the `PanelManager` *class. (See LiveCycle ES2 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 should create the following experience:

- When the Guide renders, a button labeled Last should display to the user. 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 should appear dimmed and be disabled 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 navigation controls to panel layouts and Guide layouts" on page 27.

Adding custom navigation controls to panel layouts and Guide layouts

To implement controls that you want to display as part of a Guide layout or panel layout, add the control to the MXML definition of the Guide layout or panel layout.

As an example, the following MXML source illustrates how to modify the Guide layout created in the "Creating a simple Guide layout" on page 13 section with the First and Last buttons created in the "Example: FirstPanelButton" on page 24 and "Example: LastPanelButton" on page 25 sections.

```
<?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 objects to provide an enhanced user experience. Field controls exist only on the Guide, and do not replace the original form design object on the PDF version of the form.

In this section, we explore how to create field controls to 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 objects. This custom field control assumes that the original Guide object is used to specify integer values between 0 and 100 that indicate a percentage. Mapping a Guide object to this control creates the following experience:

- When the Guide renders, an HSlider control appears in place of the original Guide object. The HSlider should have the same caption value as the original Guide object, and 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 should appear in the corresponding field. If the user changes the value on the PDF and then returns to the Guide, the slider value should reflect 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 navigation controls to panel layouts and Guide layouts" on page 27.

Referencing your Flex library project in Workbench ES2

After you build your Flex library project in Flex Builder, reference the compiled SWC file in the Guide Design perspective of Workbench ES2 to make your control extension available.

To reference your Flex library project in Guide Builder:

1. Start Workbench ES2.
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.
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.

Caution: *If you make changes to Flex library project after you add the project to a LiveCycle ES2 application, you must remove the reference to the project in the* ***Guide extensions*** *option (****Guide Properties*** *view). Once you remove the reference, re-add it for the changes to be available.*

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.

To map form design objects to custom field controls:

1 Start Workbench ES2 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 27 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 in this chapter, or by starting from a new MXML component or ActionScript class. For more information about the Guides ActionScript API, see LiveCycle ES2 ActionScript Language Reference.

To start learning about creating custom panel layouts, see "Creating Panel Layouts" on page 17 or, for custom Guide layouts, see "Creating Guide Layouts" on page 12.

7. Custom Extension Lists

When you create a Flex library project for a Guide extension, include the Guide runtime SWC file. Including the SWC file adds a collection of Guide extensions in a Guide folder in your Flex Builder Components view. The Guide extensions include everything necessary to create custom Guide layouts and panel layouts.

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

- "Button components" on page 30
- "Help components" on page 31
- "Label components" on page 31
- "Navigation components" on page 31
- "Output components" on page 31

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, depending on the values selected in the Submit From list in Guide Builder:

Guide: Clicking the submit button submits the data from the Guide.

Hosted Application: Specifies that the hosted application, such as Workspace ES2, controls the data submission. In this case, the Guide does not have a submit button. The hosted application extracts the data from the Guide and performs the data submission.

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 an end 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.

8. Guide CSS Styles

This section lists the Guide CSS styles recognized by Guides. You can use this style information to create a common CSS file that you can share among multiple Guides to maintain common styling.

Guide CSS classes

.application

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

Property	Default value
"backgroundAlpha" on page 43	1.0
"backgroundColor" on page 44	#336699
"backgroundGradientColors" on page 44	#336699, #3366CC
"basewrapper" on page 45	ga.wrappers.LeftAccordion
"paddingBottom" on page 51	10
"paddingLeft" on page 52	10
"paddingRight" on page 52	10
"paddingTop" on page 52	5
"version" on page 56	1.0.0

.buttons

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

Property	Default value
"alpha" on page 43	1.0
"backgroundAlpha" on page 43	1.0
"color" on page 46	#FFFFFF
"fillAlphas" on page 47	1.0,1.0
"fillColors" on page 48	#336699, #006699
"themeColor" on page 55	#336699

.fieldhelp

The `.fieldhelp` class specifies properties for field level help on a Guide.

Property	Default value
"backgroundColor" on page 44	#666666
"borderColor" on page 45	#666666
"color" on page 46	#FFFFFF
"dropShadowEnabled" on page 47	true
"gradientColors" on page 50	#333333, #666666
"fontSize" on page 48	12
"fontStyle" on page 49	normal
"fontWeight" on page 49	normal
"textDecoration" on page 54	normal

.guide

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

Property	Default value
"backgroundAlpha" on page 43	1.0
"backgroundColor" on page 44	#0099CC
"barColor" on page 45	#3366cc
"borderAlpha" on page 45	0.7
"borderColor" on page 45	#0099CC
"borderStyle" on page 46	solid
"cornerRadius" on page 47	15
"headerHeight" on page 50	56
"paddingBottom" on page 51	15
"paddingLeft" on page 52	15
"paddingRight" on page 52	15
"paddingTop" on page 52	15

.guidehelp

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

Property	Default value
"alpha" on page 43	1.0
"backgroundAlpha" on page 43	1.0

Property	Default value
"backgroundColor" on page 44	#FFFFFF
"borderColor" on page 45	#336699
"color" on page 46	#FFFFFF
"cornerRadius" on page 47	10
"headerAlphas" on page 50	1.0,1.0
"headerColors" on page 50	#336699, #336699
"roundedBottomCorners" on page 53	true
"textDecoration" on page 54	normal

.layoutobjects

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

Property	Default value
"paddingBottom" on page 51	
"paddingLeft" on page 52	
"paddingRight" on page 52	
"paddingTop" on page 52	

.layoutrepeaterobjects

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

Property	Default value
"paddingBottom" on page 51	20
"paddingLeft" on page 52	25
"paddingRight" on page 52	25
"paddingTop" on page 52	25

.logo

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

Property	Default value
"paddingBottom" on page 51	0
"paddingRight" on page 52	0

.navigationbase

The `.navigationbase` class specifies that area that surrounds the navigation buttons.

Property	Default value
"horizontalAlign" on page 51	center
"paddingBottom" on page 51	10
"paddingLeft" on page 52	50
"paddingRight" on page 52	50
"paddingTop" on page 52	10

.navigationlevel1

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

Property	Default value
"backgroundAlpha" on page 43	0
"backgroundColor" on page 44	#FFFFFF
"borderStyle" on page 46	none
"color" on page 46	#0b333c
"fillColors" on page 48	#336699
"fontSize" on page 48	14
"fontWeight" on page 49	bold
"rollOverColor" on page 53	#006699
"selectionColor" on page 53	#FFFFFF
"textDecoration" on page 54	normal
"textRollOverColor" on page 55	#333333
"textSelectedColor" on page 55	#000000
"themeColor" on page 55	#006699

.navigationlevel2

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

Property	Default value
"alpha" on page 43	1.0
"backgroundColor" on page 44	#006699
"color" on page 46	#FFFFFF
"fillColors" on page 48	#3399CC, #3399CC
"fontSize" on page 48	12

Property	Default value
"fontWeight" on page 49	normal
"rollOverColor" on page 53	#006699
"selectionColor" on page 53	#006699
"textDecoration" on page 54	normal
"textRollOverColor" on page 55	#333333
"textSelectedColor" on page 55	#000000
"themeColor" on page 55	#006699

.navigationlevel3

The `.navigationlevel3` class specifies properties for the third level navigation heading.

Property	Default value
"backgroundColor" on page 44	#FFFFFF
"color" on page 46	#0b333c
"fillColors" on page 48	#FFFFFF, #FFFFFF
"fontSize" on page 48	14
"fontWeight" on page 49	bold
"rollOverColor" on page 53	#FFFFFF
"selectionColor" on page 53	#FFFFFF
"textDecoration" on page 54	normal
"textRollOverColor" on page 55	#888888
"textSelectedColor" on page 55	#FFFFFF
"themeColor" on page 55	#FFFFFF

.navigationlevel4

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

Property	Default value
"backgroundColor" on page 44	#FFFFFF
"color" on page 46	#0b333c
"fillColors" on page 48	#FFFFFF, #FFFFFF
"fontSize" on page 48	14
"fontWeight" on page 49	bold
"rollOverColor" on page 53	#FFFFFF
"selectionColor" on page 53	#FFFFFF

Property	Default value
"textDecoration" on page 54	normal
"textRollOverColor" on page 55	#888888
"textSelectedColor" on page 55	#FFFFFF
"themeColor" on page 55	#FFFFFF

.navigationlevel5

The `.navigationlevel5` class specifies properties for the fifth level navigation heading.

Property	Default value
"backgroundColor" on page 44	#FFFFFF
"color" on page 46	#0b333c
"fillColors" on page 48	#FFFFFF, #FFFFFF
"fontSize" on page 48	9
"fontWeight" on page 49	normal
"rollOverColor" on page 53	#FFFFFF
"selectionColor" on page 53	#FFFFFF
"textDecoration" on page 54	normal
"themeColor" on page 55	#FFFFFF

.navigationOver

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

Property	Default value
"fillColors" on page 48	#3366CC, #3366CC

.navigationSelected

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

Property	Default value
"fillColors" on page 48	#3366CC, #3366CC

.panelcaption

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

Property	Default value
"color" on page 46	#333333
"fontSize" on page 48	12

Property	Default value
"fontStyle" on page 49	normal
"fontWeight" on page 49	normal
"textDecoration" on page 54	normal

.paneldata

The `.paneldata` class defines the properties for the area of a panel layout that displays form objects and data.

Property	Default value
"backgroundAlpha" on page 43	1.0
"backgroundColor" on page 44	#FFFFFF
"borderAlpha" on page 45	1.0
"borderColor" on page 45	#336699
"color" on page 46	#FFFFFF
"cornerRadius" on page 47	10
"headerAlphas" on page 50	1.0,1.0
"headerColors" on page 50	#336699, #336699
"roundedBottomCorners" on page 53	true

.panelhelp

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

Property	Default value
"alpha" on page 43	1.0
"backgroundAlpha" on page 43	1.0
"backgroundColor" on page 44	#999999
"borderColor" on page 45	#336699
"borderStyle" on page 46	solid
"color" on page 46	#000000
"cornerRadius" on page 47	5
"headerAlphas" on page 50	1.0, 1.0
"headerColors" on page 50	#336699, #336699

.panelitem

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

Property	Default value
"backgroundAlpha" on page 43	1.0
"backgroundColor" on page 44	#CCCCCC
"borderStyle" on page 46	inset
"color" on page 46	#333333
"dropShadowEnabled" on page 47	false
"fontSize" on page 48	12
"fontStyle" on page 49	normal
"fontWeight" on page 49	normal
"textDecoration" on page 54	normal
"themeColor" on page 55	#336699

.panelnav

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

Property	Default value
"backgroundAlpha" on page 43	1.0
"backgroundColor" on page 44	#FFFFFF
"barColor" on page 45	#3366cc
"borderAlpha" on page 45	1.0
"borderColor" on page 45	#336699
"color" on page 46	#FFFFFF
"cornerRadius" on page 47	10
"headerAlphas" on page 50	1.0, 1.0
"headerColors" on page 50	#336699, #336699
"roundedBottomCorners" on page 53	true

.paneltext

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

Property	Default value
"color" on page 46	#000000
"fontSize" on page 48	12
"fontStyle" on page 49	normal

Property	Default value
"fontWeight" on page 49	normal
"textDecoration" on page 54	normal

.progressbar

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

Property	Default value
"backgroundColor" on page 44	#336699
"barColor" on page 45	#336699

.repeater

The `.repeater` class defines the background colors of the repeater components.

Property	Default value
"backgroundColor" on page 44	#336699
"themeColor" on page 55	#336699

.videocontrol

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

Property	Default value
"backgroundColor" on page 44	#AFAFAF
"color" on page 46	#000000

ComboBox

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

Property	Default value
"color" on page 46	#000000
"dropShadowEnabled" on page 47	true
"fontFamily" on page 48	Myriad Pro
"fontSize" on page 48	11

DateField

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

Property	Default value
"color" on page 46	#000000
"dropShadowEnabled" on page 47	true
"fontFamily" on page 48	Myriad Pro
"fontSize" on page 48	12

GAIcon

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

Property	Default value
"iconColors" on page 51	#336699, #336699
"iconTextColor" on page 51	#FFFFFF

Label

The `Label` class specifies the format for static text in a Guide.

Property	Default value
"color" on page 46	#000000

RadioButton

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

Property	Default value
"dropShadowEnabled" on page 47	true
"fontFamily" on page 48	Myriad Pro
"fontSize" on page 48	12

TextInput

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

Property	Default value
"color" on page 46	#000000
"dropShadowEnabled" on page 47	true
"fontFamily" on page 48	Myriad Pro
"fontSize" on page 48	12

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.

Property	Default value
"backgroundColor" on page 44	#333333
"cellColor" on page 46	#CCCCCC

TextInputSymbol

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

Property	Default value
"prefixBackgroundColor" on page 53	#333333
"prefixColor" on page 53	#CCCCCC
"suffixBackgroundColor" on page 54	#333333
"suffixColor" on page 54	#CCCCCC

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 33
- ".guidehelp" on page 34
- ".navigationlevel2" on page 36
- ".panelhelp" on page 39

backgroundAlpha

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

Syntax

```
backgroundAlpha="[0.0..1.0]"
```

Applies to

- ".application" on page 33
- ".buttons" on page 33
- ".guide" on page 34

- ".guidehelp" on page 34
- ".navigationlevel1" on page 36
- ".paneldata" on page 39
- ".panelhelp" on page 39
- ".panelitem" on page 40
- ".panelnav" on page 40
- "TextInputMask" on page 43

backgroundColor

The `backgroundColor` property specifies the Guide background color.

Syntax

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

Applies to

- ".application" on page 33
- ".fieldhelp" on page 34
- ".guide" on page 34
- ".guidehelp" on page 34
- ".navigationlevel1" on page 36
- ".navigationlevel2" on page 36
- ".navigationlevel3" on page 37
- ".navigationlevel4" on page 37
- ".navigationlevel5" on page 38
- ".paneldata" on page 39
- ".panelhelp" on page 39
- ".panelitem" on page 40
- ".panelnav" on page 40
- ".progressbar" on page 41
- ".repeater" on page 41
- ".videocontrol" on page 41
- "TextInputMask" on page 43

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 33

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 34

".panelnav" on page 40

".progressbar" on page 41

basewrapper

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

Syntax

```
basewrapper="[Guide layout name]"
```

Applies to

".application" on page 33

borderAlpha

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

Syntax

```
borderAlpha="0.0..1.0"
```

Applies to

".guide" on page 34

".guidehelp" on page 34

".paneldata" on page 39

".panelnav" on page 40

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 34

".guide" on page 34

".guidehelp" on page 34

".paneldata" on page 39

".panelhelp" on page 39

".panelnav" on page 40

borderStyle

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

Syntax

```
borderStyle=""
```

Applies to

".guide" on page 34

".navigationlevel1" on page 36

".panelhelp" on page 39

".panelitem" on page 40

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 43

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 33

".fieldhelp" on page 34

".guidehelp" on page 34

".navigationlevel1" on page 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

".navigationlevel5" on page 38

".panelcaption" on page 38

".paneldata" on page 39

".panelhelp" on page 39

".panelitem" on page 40

".panelnav" on page 40

".paneltext" on page 40

".videocontrol" on page 41

"ComboBox" on page 41

"DateField" on page 42

"Label" on page 42

"TextInput" on page 42

cornerRadius

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

Syntax

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

Applies to

- ".guide" on page 34
- ".guidehelp" on page 34
- ".paneldata" on page 39
- ".panelhelp" on page 39
- ".panelnav" on page 40

dropShadowEnabled

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

Syntax

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

Applies to

".fieldhelp" on page 34

".panelitem" on page 40

"ComboBox" on page 41

"DateField" on page 42

"RadioButton" on page 42

"TextInput" on page 42

fillAlphas

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

Syntax

```
fillAlphas="[0.0..1.0],[0.0..1.0]"
```

Applies to

".buttons" on page 33

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 33

".navigationlevel1" on page 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

".navigationlevel5" on page 38

".navigationOver" on page 38

".navigationSelected" on page 38

fontFamily

The `fontFamily` property specifies the font typeface.

Syntax

```
fontFamily="[font name]"
```

Applies to

"ComboBox" on page 41

"DateField" on page 42

"RadioButton" on page 42

"TextInput" on page 42

fontSize

The `fontSize` property specifies the font size.

Syntax

```
fontSize="integer"
```

Applies to

".fieldhelp" on page 34

".navigationlevel1" on page 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

".navigationlevel5" on page 38

".panelcaption" on page 38

".panelitem" on page 40

".paneltext" on page 40

"ComboBox" on page 41

"DateField" on page 42

"RadioButton" on page 42

"TextInput" on page 42

fontStyle

The `fontStyle` property specifies whether to use italicized font.

Syntax

```
fontStyle="normal | italic"
```

Applies to

".fieldhelp" on page 34

".panelcaption" on page 38

".panelitem" on page 40

".paneltext" on page 40

fontWeight

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

Syntax

```
fontWeight="normal | bold"
```

Applies to

".fieldhelp" on page 34

".navigationlevel1" on page 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

".navigationlevel5" on page 38

".panelcaption" on page 38

".panelitem" on page 40

".paneltext" on page 40

gradientColors

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

Syntax

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

Applies to

".fieldhelp" on page 34

headerAlphas

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

Syntax

```
headerAlphas=""
```

Applies to

".guidehelp" on page 34

".paneldata" on page 39

".panelhelp" on page 39

".panelnav" on page 40

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 34

".paneldata" on page 39

".panelhelp" on page 39

".panelnav" on page 40

headerHeight

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

Syntax

```
headerHeight="[integer]"
```

Applies to
".guide" on page 34

horizontalAlign

The `horizontalAlign` property specifies the standard Flex style for most containers.

Syntax

```
horizontalAlign=""
```

Applies to
".navigationbase" on page 36

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
"GAIcon" on page 42

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
"GAIcon" on page 42

paddingBottom

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

Syntax

```
paddingBottom="[0..80]"
```

Applies to
".application" on page 33

".guide" on page 34

".layoutobjects" on page 35

".layoutrepeaterobjects" on page 35

".logo" on page 35

".navigationbase" on page 36

paddingLeft

The `paddingLeft` property specifies the border around the left edge of the Guide.

Syntax

```
paddingLeft="[0..80]"
```

Applies to

".application" on page 33

".guide" on page 34

".layoutobjects" on page 35

".layoutrepeaterobjects" on page 35

".navigationbase" on page 36

paddingRight

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

Syntax

```
paddingRight="[0..80]"
```

Applies to

".application" on page 33

".guide" on page 34

".layoutobjects" on page 35

".layoutrepeaterobjects" on page 35

".logo" on page 35

".navigationbase" on page 36

paddingTop

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

Syntax

```
paddingTop="[0..80]"
```

Applies to

".application" on page 33

".guide" on page 34

".layoutobjects" on page 35

".layoutrepeaterobjects" on page 35

".navigationbase" on page 36

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 43

prefixColor

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

Syntax

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

Applies to

"TextInputSymbol" on page 43

rollOverColor

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

Syntax

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

Applies to

".navigationlevel1" on page 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

".navigationlevel5" on page 38

roundedBottomCorners

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

Syntax

`roundedBottomCorners="true | false"`

Applies to

".guidehelp" on page 34

".paneldata" on page 39

".panelnav" on page 40

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 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

".navigationlevel5" on page 38

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 43

suffixColor

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

Syntax

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

Applies to

"TextInputSymbol" on page 43

textDecoration

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

Syntax

`textDecoration="normal | underline"`

Applies to

".fieldhelp" on page 34

".guidehelp" on page 34

".navigationlevel1" on page 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

".navigationlevel5" on page 38

".panelcaption" on page 38

".panelitem" on page 40

".paneltext" on page 40

textRollOverColor

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

Syntax

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

Applies to

".navigationlevel1" on page 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

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 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

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]"
```

Applies to

".buttons" on page 33

".navigationlevel1" on page 36

".navigationlevel2" on page 36

".navigationlevel3" on page 37

".navigationlevel4" on page 37

".navigationlevel5" on page 38

".panelitem" on page 40

".repeater" on page 41

version

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

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

Syntax

`version="[version number]"`

Applies to

".application" on page 33