Best Practices for Model-Driven Development with Adobe® LiveCycle® ES2 Guides

Guiding people through the data capture experience.

Introduction

Background

Guides in Adobe LiveCycle ES2 are Flash Player compatible wizard-like panels that help guide people through a data capture experience. Guides can dynamically change and adapt based on input data to ensure that only relevant questions are asked and accurate data is captured. These rich and engaging experiences help **reduce transaction abandonment rates** that are prevalent with more complex data collection interfaces.

In LiveCycle ES2, Guides use the concept of model-driven development, which is the next level of abstraction in writing software applications. This abstraction offers improved productivity and makes applications that are higher in quality and easier to create, compared to lower-level development techniques. Model-driven development separates the data structures and constraint logic from the user interface and display logic. **Models can then be shared and reused by other applications that capture, display or process the same data.**

The previous incarnation of guides was known as "form guides," and used an XDP form as both the model and host of the form guide definition. Form guides were built directly in LiveCycle Designer, and were rendered by LiveCycle Forms into a SWF (pronounced "swiff") file for display in the Flash Player.

With the release of LiveCycle ES2, the Guide Builder tool has been moved to LiveCycle Workbench and a new application model has been introduced. The application model (.fml) is created in the Data Modeler tool, which is also a new part of the LiveCycle Workbench in ES2. The two main consumers of the application model within LiveCycle ES2 are guides and Data Services; however, Output and Process Management also have a role to play with model-driven development. Application models used with guides or Data Services can be used to present data in Output-rendered PDF documents and can be exposed as data types within Process Management.

With the release of LiveCycle ES2 Service Pack 1, the guide design perspective in Workbench will support both the application model (.fml) and the form model (.xdp). This provides an upgrade path for customers using form guides in LiveCycle ES and ES Update 1, as well as some additional choices in the application design.

Audience

This document is aimed at IT analysts and developers creating guided data capture applications with LiveCycle. The audience should be familiar with LiveCycle ES2 Workbench, the general approach to application development in ES2 and be familiar the Data Modeler, Form Design, and Guide Design perspectives in Workbench ES2. This document is not intended to provide basic documentation on these subjects. It is intended to assist the more advanced user in design decisions about creating applications faster and in a manner that is easier to maintain. Model development and guide development require no programming; readers do not need to understand Flex programming to benefit from this document. Some sections, however, will address customization capabilities that require Flex development.

Choosing the model type

The type of model you choose to use can impact the guide's use and capabilities within a LiveCycle application. Here are some things to consider when selecting your model type:

Application models

Application models (.fml) are developed in the LiveCycle Data Model tool provided in Workbench. The model defines the data shape, constraints and validation logic. Any constraints and validation logic can be developed in the data modeling tool without the need to write script. Additional logic such as rules about when to display a guide panel can be developed in the guide itself. The application model can be used as a data connection in LiveCycle Designer. The resulting XDPs can be rendered using guide data to a non-interactive PDF through Output ES2. The guide data cannot be exchanged in the browser with an interactive PDF form.

XDP models

As of the ES2 Service Pack 1 release, you can use an XDP form as your model. The XDP can be bound to an XML schema definition. If you have an existing XML schema, you can quickly build an XDP form in the LiveCycle Designer that is bound to that schema and submits data that is validated against that schema. JavaScript written in an XDP form is automatically translated to ActionScript in a guide, and is used to validate data and enforce constraints. **If any script runs on the server side, Forms ES2 will need to be licensed**. Additional logic, such as rules about when to display a guide panel, can be developed in the guide itself. Guides based on an XDP model can also exchange data inside the browser with an interactive PDF form to provide a dual data entry mechanism.

Using an application model

Planning your application model

Before you create your application model, spend some time sketching out the relationships among the various data structures that exist in your application. Think about how data structures may be repeated in the application (now and in the future), and create them as separate entities. These structures often appear as repeating panels within your guide. You can easily change the relationships from one-to-one to one-to-many later, but breaking up monolithic structures later can necessitate a lot of reworking. Creating entities that align with guide structures like nested panels allows you to bind them to the entire entity rather than individually binding portions of a larger entity. It helps to understand what goes into the model and what goes into the guide. The model will dictate the structure of the data, including repeating structures, and contains logic that ensures that the data is valid independently of its presentation format. The model may also contain services, such as Adobe Web Service, that retrieve data. The guide definition determines the physical organization and layout of the application, contains navigation logic and show/hide logic, and determines when and how services are invoked.

Consider which entity is the top-level entity in the data structure, and make that persistent by setting the persistence property in the entity's proper page. You'll need to add a property with the "ID Property" checkbox selected into the persistent entity. The ID Property helps identify unique instances of the entity. Marking an entity as persistent means that it will be saved somehow (possibly by committing to a database, by committing to the server as XML or by a completely different persistent method, as the fiber model is really agnostic to the persistence method). Once the entity is saved, you need a way of distinguishing among the many entities of this type that could be saved. So you need identity keys to distinguish them. For readability's sake, consider a CamelCase naming convention rather than Hungarian notation for entities and properties. It will help to identify the fields better within the guide builder and the property's data type will already be obvious in the guide builder through icons and property types. Entities must begin with upper case letters and properties must begin with lower case. You may be overriding the data types anyway in your application and CamelCase will save you time as it will automatically be converted to a usable caption (firstName becomes First Name).

Using web services in an application model

If your model uses web services to call functions or return data back to the guide, consider your **plan to deploy the crossdomain.xml file** and your use of relative addresses in the service call. To prevent cross-site scripting security vulnerabilities, Flash Player requires this file in order to make web service calls. The server destination for your web service must have a crossdomain.xml file in its root directory and contain the domain from which your guide will be trying to access it. If you do not have file system access to the destination server, consider creating a LiveCycle process to pass the request on to the destination server and put your crossdomain.xml file on the LiveCycle ES2 server. You can then make web service calls from your guide back through LiveCycle and a crossdomain.xml file that you control. The address to your LiveCycle Server should be a relative address. This makes moving the guide from development, test and production server environments possible without having to edit the application. The address will be relative to the server that serves up the HTML wrapper page in which the guide is delivered, and will be fully resolved to include the server address. Unfortunately relative addresses won't work in a guide's preview unless the Workbench and LiveCycle ES server are installed on the same machine as the Workbench that serves up the HTML wrapper.

Validations and constraints

Validations and constraints are the logic within a model that ensures that data is valid according to business rules. Validations ensure that the data value of a single property is valid (such as within an acceptable range of numbers). Constraints ensure that combinations of data between two or more properties are valid. For example, an address may be considered valid if any combination of the street address and the state or ZIP code contains values. Validations and constraints should be used when their rules need to be applied regardless of the presentation. Guide rules and input masks should be used to control the presentation aspects. It's possible to create fairly complex expressions with constraints using ternary expressions or building up multiple constraints. However, if the logic is very complex, is shared by other applications or needs server-side access in order to make decisions, a web service may be more appropriate. From guides you are able to pass field data as parameters to the web service and execute the service based on events within the guide.

Using application model styles and text strategies

In data model terms, a style is a collection of user-interface-related aspects that can be associated with a property element in the model. This is not to be confused with style sheets, which determine the guide's color, font face and size. Among other things, a style can define validation and constraint messages that are to be displayed when the data value of a property is not valid or the data in two or more properties is not consistent. You express the association between a property and a style by referencing the style in the style attribute of a property element or by declaring the style in-line in the property. Global styles are useful when you have properties with identical validation requirements such as multiple ZIP code fields in your model. In that case, creating a global style for ZIP code and referencing that style from the ZIP code fields in your model will consolidate both the validation logic and the validation message, making maintenance easier. In-line styles are appropriate for properties that are unique within the model. If you think the model may grow over time, using global styles to start may make future changes easier.

There are several places within a guide that may contain text. For more dynamic text values that change on a regular basis, such as drop-down list items, consider storing these values externally and retrieving them through a web service. The results can then be bound into the various properties of guide components. If you want to centralize the text values, but putting that text in the model or calling web services is not appropriate, consider using SelectionLists to store text and binding that to the captions and text within your guide.

Using an XDP form model

While any form design can be made into a guide, the process of developing the guide can be made easier by following these guidelines:

- Group form sections into subforms.
- Do not use nested subforms in repeating subforms. Guides assume all repeating form objects are immediate children of the repeating subform.
- Avoid using form fields on master pages.
- If possible, use field captions instead of static text. This will avoid you having to link static text to panel items in the guide definition if you want captions to appear in the form guide.

Form scripting

Form scripting is a common area where issues may occur when trying to develop a guide from an existing form design. The reason has to do with the level of scripting support provided by guides and behavior differences between the Flash runtime and Acrobat/Reader.

Guides support a **subset** of the full XFA scripting model. Any scripting that falls outside of the subset will fail to compile. Referring to the guide scripting support documentation is imperative in order to ensure that the scripts on your form design remain compliant. This information is found in the document Scripting Support for HTML Forms and Guides.

Only use JavaScript for client-side logic. If the script in your XDP is set to run on the client guides only supports JavaScript and **not FormCalc**. Scripts executed on the server can use either script language.

When developing script on a form design that will be executed in a guide, be sure to follow these guidelines:

- Scripts that assign a value must be expressed as:

  ```
  this.rawValue = price.rawValue * quantity.rawValue;
  ```

- The script price.rawValue * quantity.rawValue will compile because it is within the XFA subset, but it will not work in a guide, as XFA automatically assigns the result of the calculation to the value of the field.
- Nested functions in JavaScript are not supported.
- Maintain a library of common form guide functions in a script object that can be reused.
- The formReady event only fires once each when the PDF and guide load.
- The calculate event will fire when referenced field values change *and* when switching between the guide and PDF view.
- Only use xfa.host.messageBox. Message boxes will appear as default Flex Alert boxes in a guide. Custom CSS can be added to a guide's style to match the Alert box style to the rest of the application.
- Guides are unable to wait for user response (e.g. Yes/No) from a message box. Such behavior is possible but requires Flex custom components to be built (advanced usage).
- When checking for empty field values, look for **null and zero-length**.
- When scripting the value of Date fields use field.dataNode.value to ensure consistent behavior in both environments. **In form guides, accessing a date field's rawValue returns its formatted value**.
- Do not use xfa.event.change since it is not supported in form guides.
- When adding script to your form, it may be appropriate to limit execution to only guides (Flash) or provide alternate execution paths depending on the host environment. This can be achieved by querying the xfa.host.name property.

  ```
  if (xfa.host.name == "Flash")
          { //script will execute in guide }
  else
          { //script will execute in Acrobat/Reader }
  ```

Customizing your guide

Using the SDK

You may want to customize your guide through guide extensions, custom components or style sheets. Once created, these extensions can be used in other guides you create to get consistent layouts, navigation and look and feel. The LiveCycle ES2 SDK is installed with Workbench and is typically found under:

"C:\Program Files (x86)\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK".

The guide portion of the SDK is found under the "Misc\Guides" subdirectories. The guides SDK contains the source code and Flex Builder projects for modifying and creating your own guide extensions. If you use custom extensions, be sure to add the dcruntime_library.swc to the library path of the project and mark the Link Type as "External". This will remove the framework and any guide components from your extension SWC, significantly reducing the guide size. **If you don't do this, you are adding about 1400 KB to your guide**.

Customizing appearance

Guide Builder provides the ability to include custom style sheets that can be used in one or more guides. Styles can be created in cascading style sheet authoring tools such as Dream Weaver.

You will need to compile your CSS file to a SWF to use it in your guide. This allows any referenced images to be included as one file. You will need to use Flex Builder / Flash Builder 4 to compile the CSS to a SWF. You can then add your style SWF to your guide application or place it in a central application if the style will be used by multiple guides.

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

Recommended software

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

Workbench ES2 is installed (by default C:\Program Files\Adobe\Adobe LiveCycle Workbench ES2\LiveCycle_ES_SDK\misc\Guides). See the system requirements for Flex Builder and for Workbench ES2.

Extensions

Extensions to a guide are any external libraries that are referenced using Guide Builder. These libraries can contain new guide or panel layouts that you have created in Flash Builder or they can even be a third-party library that includes new UI controls.

The following guidelines can be followed when developing any type of extension to a guide:

- Set up your Flex project according to the instructions in the customizing guides documentation including the use of the **Flex 3.4.1 SDK.**
- After adding both guide SWC's to your project ,set their link type to External. This will keep your SWC file size to a minimum.
- Import all the guide source projects into Flex Builder.
- Try to use the source of one of the existing guide samples as a starting point by copying and pasting into a new MXML file.

A guide consists of both guide and panel layouts. A guide layout defines the visual layout and structure of a guide that remains constant throughout a form-filling session. A panel layout defines the visual layout and presentation of objects on a panel in the guide hierarchy.

While it is advisable to make use of the various layouts provided by default, in many cases customization will be required. Customization can be as simple as moving existing controls, or they could be far more complex, affecting layouts and behaviors.

Guide layouts

When creating a custom guide layout, you should consider the following guidelines:

- Override the createChildren() function to set up any event listeners.
- Listening for GAEvent.INTIALIZED is useful to ensure the guide and its model have been fully initialized.
- Retain as many style class names as possible. The style names used by the default layouts refer to styles that are defined by Guide Builder in Customize Appearance. Retaining these style names in your own layout means that its style can be easily customized as well.

Panel layouts

When creating a custom panel layout, you should consider the following guidelines:

- Override the documentDescriptor property (get/set) in order for the slot replacement algorithm to work correctly.
- Retain as many style class names as possible.
- Place PanelItem components inside a hidden container to access their data only. This technique is useful if you have a complex control that binds to multiple data items since in guides each panel item can only bind to one data node.

Navigation controls

Navigation controls are interactive objects that allow a user to navigate through a guide. Guides provide a number a navigation controls that are provided in the default wrappers.

Field controls

A field control is a UI component that is used to display the data contained in a form design object. Guide Builder offers some suggestions by default, but in some situations, using a different Flex object may make for a more engaging user experience. A custom field control can either extend an existing Flex UI component or be made completely from scratch if desired.

Resources
LiveCycle Guide Builder Blog
Stefan Cameron's Form Blog
The LiveCycle Product Blog
Adobe LiveCycle Café

When creating custom field controls, it is good practice to understand how the guide runtime responds to field changes and how it updates the data model. The following table lists what events the guide runtime listens to and how it maps data from the Flex control to the XFA data model. The first matching Flex class that is found will be used.

Flex class	Event listener	Flex property	Data model property
Repeater	valueCommit	selected	selectedItems
DateField	valueCommit	text	formattedValue
CheckBox	valueCommit	selected	booleanValue
Radiobutton	valueCommit	selected	booleanValue
Button	valueCommit	label	rawValue
ComboBase	valueCommit	selectedItem	selectedItem
ComboBase (editable)	valueCommit change	text	rawValue
ListBase	change	selectedItems	selectedItems
Default *	valueCommit	text	formattedValue
Default *	valueCommit	selected	booleanValue

* By default if a Flex control is not one of the types listed, the guide runtime will determine if the control has a text, selected or value property and act accordingly.

Other considerations

Previewing your guide

There are several options that you can configure on the guide properties panel during guide preview. The Preview data field on the guide properties tab is used only for design-time previewing. It is not used when rendering a guide in Workspace. This is helpful when testing the guide's functionality, as it avoids the need to key in test data every time the guide is previewed. One way to quickly create a data file that will match your model is to add a Model Viewer panel to your guide (see the section in the documentation on debugging guides). On the Model Viewer panel, the XML data tab will display the data formatted according to the model. This data can then be cut and pasted into a test data file. To pre-populate a guide with data at run-time, you'll need to add a pre-fill process on the guide's action profile.

Accessibility

In order to make an application created with Flex (and by extension guides) accessible, you need to enable accessibility for that application. This compiler setting is automatically set for you in ES2 guides. This will import the accessibility object for each component used in the application.

The read-order/tab-order is automatically set by guides and follows the flow of the objects in the panel. The Cobalt standard wrapper is the best choice for accessibility. Consider limiting the number of different panels that are used in the guide to prevent confusing a visually impaired user with different locations of help instructions. The standard field components provided by Guide Builder are accessible, but if your application requires custom components, you should ensure that it is one of the 28 accessible Flex components. Also reviewing the best practice documentation on Flex accessibility will help you understand how accessibility functions in Flex applications.

Feedback
We welcome your comments. Please send any feedback on this technical guide or suggestions for other topics to: LCES-Feedback@adobe.com

For more information and additional product details:
http://www.adobe.com/devnet/livecycle/

Adobe Systems Incorporated
345 Park Avenue
San Jose, CA 95110-2704
USA
www.adobe.com