Designing PDF Forms and Flex Form Guides – Adobe MAX 2007

Track ID: *EC202H* ***Level:*** *Intermediate* ***Hands-on:*** *90 minutes*

Author / Presenter

Stefan Cameron, Computer Scientist
LC Designer ES Team
Adobe Systems Incorporated
scameron@adobe.com
http://forms.stefcameron.com

Abstract

Get up-close and personal with Flex and XFA in this hands-on training session. You will learn how these two powerful technologies can be used together to offer your customers the best online and offline form-filling experiences. We will cover basic form design, scripting and data binding techniques while creating a dynamic PDF form which we will then easily re-purpose to a rich Flash-based wizard, deployable from LiveCycle Forms ES, using the Guide Builder tool.

Main Objectives

We will be adding the final but crucial details to a registration form for fictitious races to be held over the course of the MAX conference. We will be

- defining a schema data connection to ensure that submitted data adheres to a schema;
- adding script to enable the applicant to register multiple people (e.g. for a family);
- using basic dynamic form design techniques to make the objects on the form expand and retract depending on specified data; and
- quickly re-purposing a dynamic form as a Flash-based Form Guide wizard using the new Guide Builder tool and deploying it to a web browser using LiveCycle Forms.

Required Software

The following software is **required** for this tutorial:

- Adobe LiveCycle Designer ES

- Adobe LiveCycle Guide Builder ES
- Adobe Acrobat 8.1 Professional
- Adobe Flash Player 9.x
- Email program such as Outlook or Outlook Express (because the form submits its data via an email submit button)

The following software is **optional**. Without this software, you will be able to preview the Form Guide from Walk-through Four however you will not be able to deploy it to a web browser, see the PDF form being filled as the Form Guide is filled and you will not be able to submit data from the Form Guide:

- Adobe LiveCycle Forms ES

I. Defining an XML Schema Data Connection

The main purpose of this form is to collect data. For an electronic form in an electronic workflow, the form's data is typically submitted to a receiving entity such as an email address, a URL, a database or a web service. Without doing more than placing fields and an email submit button on a form, clicking the email submit button will create a new email message with an XML data file attached to it. The data file will contain an XML structure that describes the content entered into the fields in a hierarchical manner.

While the default XML data that's produced can be useful, business processes usually require that the data output from a form adhere to a particular schema. To ensure that a form's output XML data is structured according to the rules defined in a schema, you can define a **Schema Data Connection** and create **bindings** between its data items and fields and containers on your form using the **Data View palette**.

LC Designer ES helps you quickly create the proper fields and containers with types and properties that adhere to a data connection (whether it's to a schema, database or web service) via the Data View palette.

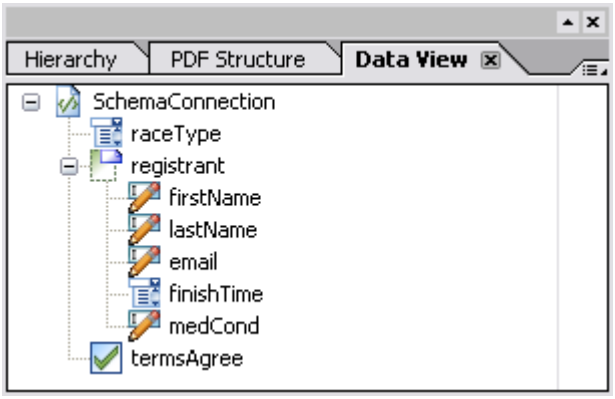

Figure 1: Data View palette with schema data connection

Once the data connection is defined, you may drag and drop data items onto the form. If you drag a single data item over an existing field, a little shortcut icon will appear on the cursor to indicate that

dropping the data item at that location would update the field to the data item's properties and binding. Otherwise, you may drop the data item onto the form and Designer will create a field with a type and constraints that best matches its definition as per the schema.

> **» Demo 1:** Define a data connection to a schema in a new form and quickly create fields bound to the data connection by dragging and dropping data items onto the form. Inspect the resulting fields to see how they relate to the schema's types and constraints.

A. Walk-through One

Objective: Define a data connection to a schema and use the drag and drop technique to link existing fields on a form to items in the data connection. Use the **Binding Properties** tool to update properties of those fields to match schema rules.

Inspect the schema

1. Open the "MaxRaceRegistration.xsd" schema file located in the "Data" folder using Notepad and inspect the various properties.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementForm
  <xs:element name="maxRaceReg">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="raceType"/>
        <xs:element ref="registrant" minOccurs="1" maxOccurs="5"/>
        <xs:element name="termsAgree" type="xs:boolean"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="raceType">
```

Figure 2: MaxRaceRegistration.xsd schema source

Note that the *maxRaceReg* element is the root which includes 3 data items:

a. *raceType* – an item whose value comes from an enumeration of defined values which describe the type of race the applicant is registering for.
b. *registrant* – a repeatable item containing information about a registrant of which there must be at least 1 occurrence but no more than 5.
c. *termsAgree* – a "true/false" item indicating whether the applicant has agreed to the "rules and regulations" of the race.

Open the form and define the data connection

2. Open the "ToTheMAX_Registration_**WT1**.pdf" form in LC Designer ES.
3. Open the Hierarchy palette (use the "Window > Hierarchy" menu command if it isn't visible). Notice that the root subform's name is "form1" (its default name).

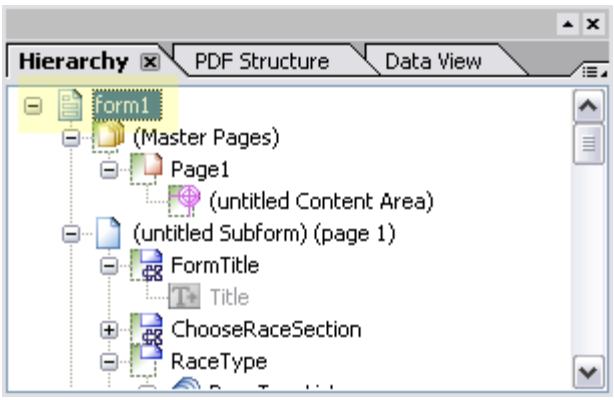

Figure 3: Default root subform name is "form1"

4. Open the Data View palette (use the "Window > Data View" menu command if it isn't visible). Notice that there are no data connections defined.
5. Using the palette fly-out menu, choose the "New Data Connection…" item.

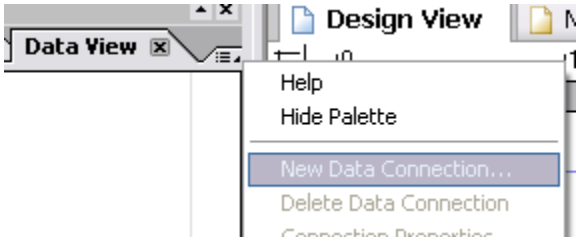

Figure 4: "New Data Connection" command in fly-out menu

6. Set the name to "SchemaConnection", choose the "XML Schema" option and click "Next".
7. Select the "MaxRaceRegistration.xsd" schema file located in the "Data" folder.
8. Select "maxRaceReg" as the "XML Data Root Element Name". This will **rename the form's root subform** (currently named "form1" by default) to "maxRaceReg" so that the submitted data has a root element named "maxRaceReg", **as the schema requires**.
9. Click "Finish". The new data connection will be displayed in the Data View palette. To save time, some of the bindings had already been defined in the form. You will see a binding icon next to each data item that is bound. The remaining items are "registrant" and "finishTime".

Apply bindings to "Registrant" subform and "FinishTime" fields

10. Open the Hierarchy palette and select the "Registrant" subform inside the "RegistrantInfo" subform.
11. Open the Object palette (use the "Window > Object" menu command if it isn't visible) and choose the Binding tab.
12. Use the popup button next to the "Default Binding" property to select the "SchemaConnection > registrant" data item. The resulting value in the property will be "registrant[*]" to indicate that this subform is bound to multiple occurrences of the "registrant" data item.
13. On the same tab, check the "Repeat Subform for Each Data Item" box and set the "Min Count" and "Max" properties to 1 and 5, respectively. This will limit the number of possible instances to a minimum of 1 and a maximum of 5, as per the schema.

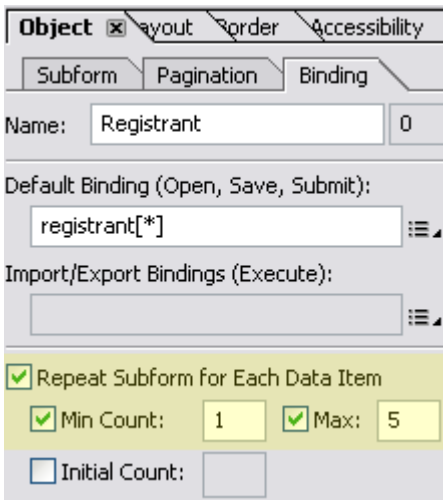

Figure 5: Making "Registrant" subform repeatable

14. Select the “Finish Time” field in the “Registrant Information” section. Notice that it doesn’t have any list items defined in the Object palette’s Field tab.
15. Return to the Data View palette and drag & drop the “finishTime” data item onto the “Finish Time” field. In the **Binding Properties** tool that opens, select “Update the following properties only” and make sure that “List Items” is the **only** box that is checked. Click “OK”.
16. Have another look at the Object palette’s Field tab. You should now see that the “Finish Time” field has been updated with list items as defined in the schema.

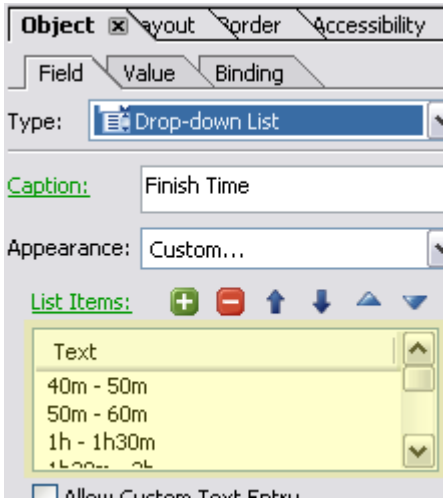

Figure 6: "FinishTime" drop down list now has items from schema

B. Further Reading

Using the Dynamic Binding feature (populate lists from data, set captions from data, etc.)
http://forms.stefcameron.com/2006/07/29/dynamic-properties/

Connecting a Form to a Database
http://forms.stefcameron.com/2006/09/18/connecting-a-form-to-a-database/

Selecting Specific Records in a Database

http://forms.stefcameron.com/2006/09/29/selecting-specific-database-records/

Displaying All Records in a Database
http://forms.stefcameron.com/2006/10/12/displaying-all-records-from-an-odbc-data-connection/

Connecting a Form to a Web Service
http://forms.stefcameron.com/2007/05/21/connecting-to-a-web-service/

II. Using the Instance Manager

It's often the case that a dynamic form will have one or more sections which may need to be repeated in order to accommodate more or less data depending on the person filling it out. In our race registration example, the applicant has the option of registering between one (the minimum) and five (the maximum) people for a race (as per the rules defined in the schema to which we bound the form in the previous step).

In an XFA form, sections are typically represented by subforms which are containers of other objects, even other ("nested" as some call them) subforms. Subforms play a very important role when it comes to data bindings because they represent the multiple levels (hierarchy) of data items in the data that gets submitted from a form (e.g. based on a schema as we saw in the previous section). They are also very useful for layout purposes because their content can either be **positioned** (fixed size with objects in specific places)

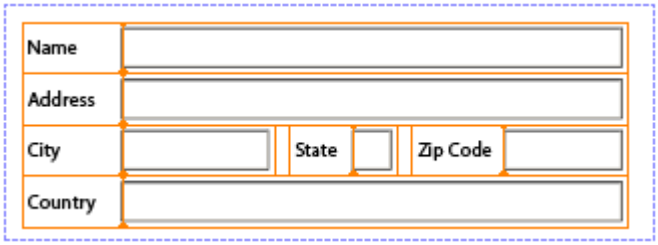

Figure 7: Subform with positioned content

or **flowed** (dynamic size with objects flowing one after another without any specific positioning).

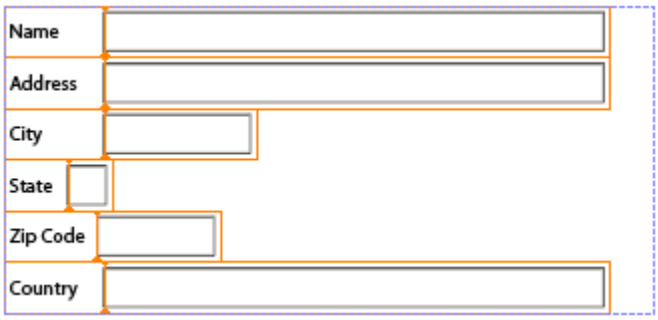

Figure 8: Subform with flowed content (result of making previous subform "flowed")

When a section must be repeated, the **Instance Manager** must be used to generate new instances and remove existing ones. By default, all subforms have an Instance Manager however, unless you make the subform **repeatable** using the Object palette's "Repeat for each data item" property,

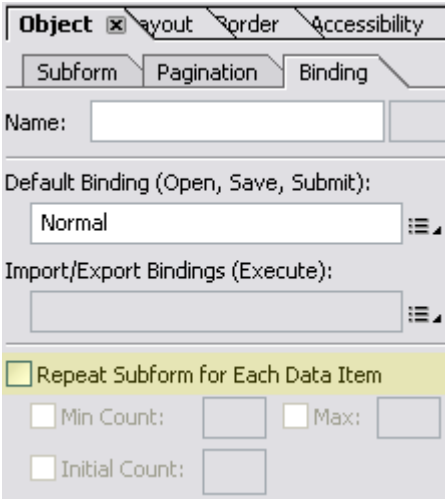

Figure 9: Object palette Binding tab's property to make subform repeatable

the Instance Manager will be unable to add/remove instances because the subform's minimum and maximum occurrences are both set to 1 by default.

LC Designer ES takes this a step further by restricting the subforms which can be made repeatable to those placed inside a **flowed** container (another subform with its Content property set to "Flowed") in order to encourage proper form layout. If this isn't the case, the Object palette will **disable** the "Repeat for each data item" property. We will cover the reasons behind this restriction in greater detail in the following section.

Note that in the previous section on Schema Data Connections, you made the "Registrant" subform, in the "Registrant Information" section, repeatable and you specified that it should have a minimum of 1 and a maximum of 5 instances at any given time.

The effect of making a subform repeatable is two-fold:

1. It has the effect of letting the subform **repeat as required** in order to display multiple instances of a repeating data item in **data that you import into the form** (either from an XML Data File, a database or a web service).
2. It gives the subform's Instance Manager the ability to **add/remove instances dynamically** when the **form is being filled** (e.g. in our race registration form, we want to give the applicant the ability to add and remove registrants).

A. Accessing the Instance Manager Object in Script

The Instance Manager object is something that's only available via script once the form is **saved as a dynamic form** and is **running in a filler application** such as Acrobat or a browser. The Instance Manager doesn't make sense in static forms since these cannot modify their structure dynamically.

Every subform has a property named "instanceManager" which you can access like this:

```
MySubform.instanceManager
```

Note that **at least one instance** of the (repeatable) subform **must exist** in order for your script to be able to access its instanceManager property. If, for example, you attempted to add an instance when no instances exist, you would get the following script error:

```
MySubform is not defined
```

That's because there is no instance of the "MySubform" object currently in existence in the Scripting Object Model.

> **» Demo 1:** Show a simple form with a repeatable subform and a button to add instances. Remove the minimum and use the "instanceManager" property to add instances in order to show the error.

But what if your form needs to permit the removal of **all** instances of a (repeatable) subform? How would you add a new instance once they've all been removed?

There is an **alternate way** to access a subform's Instance Manager object which you can use even if there are no instances currently in existence: Every subform gets a **sibling object** with the same name, prefixed with an underscore, which is a **reference** to its "instanceManager" property. Therefore, our "MySubform" object would get a sibling object in the scripting model named

```
_MySubform
```

Referring to our example of adding an instance when no instances exist, you would do this instead:

```
_MySubform.addInstance(0);
```

Since you can **always** access a subform's Instance Manager in this alternate way, I suggest you favor it over using the "instanceManager" property.

> **» Demo 2:** Return to the first demo form and update the syntax to use the "underscore prefix" notation to show that it's now possible to add instances without errors.

B. Adding, Removing and Counting Instances

When working with repeatable subforms, the three most important things you'll want to do is add, remove and count instances.

To **add instances**, you'll use the "addInstance(0)" method like this:

```
_RepeatingSubform.addInstance(0);
```

To **remove instances**, you'll use the "removeInstance(index)" method like this:

```
_RepeatingSubform.removeInstance(3); // remove the 4th (zero-based) instance
```

One **very important** thing to note about removing instances as of **Acrobat/Reader 8.1** is that if there's anything else you need to do (any other script statements) along with removing an instance, make sure that the instance is removed once all other statements have been executed. The new Direct Rendering engine in Acrobat/Reader 8.1 removes the instance, and all objects that it contains, **immediately** (as opposed to removing it on the next rendering pass as in previous versions) which means that any subsequent statements are not executed.

To **count** the number of **existing instances**, you'll use the "count" property like this:

```
_RepeatingSubform.count;
```

The "count" property is useful when you need to make common changes to all instances or when you need to determine if you can add more instances – if you've reached the maximum allowed – or if there are still instances that can be removed without going beyond the set minimum. Note that the filler application (Acrobat or a browser) will automatically prevent adding more instances than the set maximum (if any) and removing more instances than the set minimum (to zero) however you'll provide a much better user experience if you catch the invalid addition/removal before it happens and inform the user in some way (e.g. with a message stating the restriction).

C. Walk-through Two

Objective: The goal in this step is to enable the applicant to enter multiple registrants (from one to five) in the form. The applicant will be able to click the "Add" button to add registrants and each instance of the "Registrant" subform will have its own "X" button which can be used to remove that specific registrant from the set.

Preview the form to see that the "Add" button doesn't work

1. Open the "ToTheMAX_Registration_**WT2**.pdf" form in LC Designer ES.
2. Preview the form and press the "Add" button in the "Registrant Information" section. Note that **nothing happens** (new registrant instances aren't added to the form).

Add necessary script to "Add" button to add instances

3. Locate and select the "Add" button in the "Registrant Information" section (in the middle of the form).

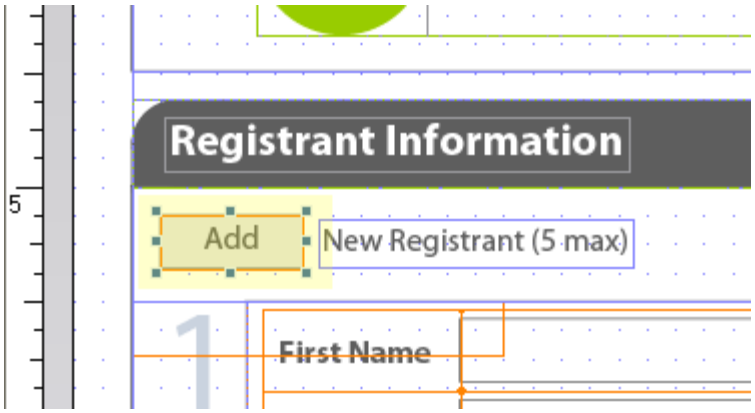

Figure 10: "Add" button in "Registrant Information" section

4. Open the Script Editor palette (use the "Window > Script Editor" command if it isn't visible; otherwise, make sure that it's in multiline view by making it tall).
5. Choose the "Click" event from the "Show" drop down list at the top left hand corner of the Script Editor palette.
6. There is already script in this event. It is designed to operate only if there are less than 5 instances of the "Registrant" subform and has been partially disabled. What's missing is the statement which will add a new instance of the "Registrant" repeatable subform.
7. Replace the "//// script here ////" comment with "var oNewReg = _Registrant.addInstance(0);" and un-comment the remainder of the script (by removing the "/*" and "*/" comment notation).

```
if (_Registrant.count < 5)
{
    //// script here ////

/*
    Utilities.EnableField(this, _Registrant.count < 5);
```

Figure 11: Comment to be replaced in "Add" button Click event

This statement will **add the new instance and capture a reference to it** so that the remainder of the script can adjust the "Add" button and the new instance's delete ("X") button depending on the new instance count.

Add necessary script to "Delete" button to remove instances

8. Locate and select the delete button. It can be found just below the "Count" field with the default value of "1" in the grayish blue color. Since it is hidden by default, you may need the Hierarchy palette to find and select it. Look for a button object with the name, "Delete".

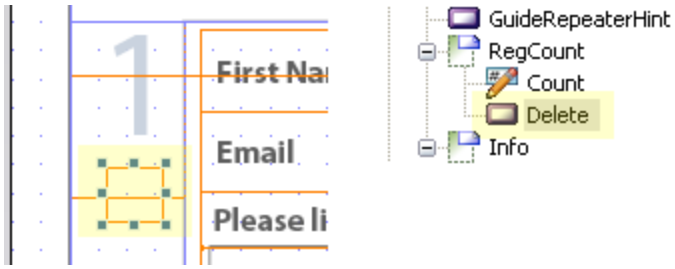

Figure 12: "Delete" button on canvas and in Hierarchy palette

9. If it isn't already, set the Script Editor palette to show the Click event script of the "Delete" button.
10. The script in this event is designed to remove the instance of the "Registrant" subform which contains the delete button being clicked, ensure that the "Add" button is enabled and, if a single instance remains, hide its delete button. What's missing is the statement which will remove the instance to which the delete button belongs.
11. Replace the "//// script here ////" entry with "_Registrant.removeInstance(Count.rawValue – 1);" This will **remove the instance to which the delete button belongs** because the value of the "Count" field, which displays the number of the instance, can be used to derive the instance's zero-based index which we need to give to the Instance Manager's "removeInstance" method.

```
    oOtherInstance.RegCount.
}

//// script here ////

// Make sure you don't put a
```

Figure 13: Comment to be replaced in "Delete" button Click event

Preview the form to see how it works; notice the layout problem!

12. Preview the form again and click on the "Add" button to add registrant instances. **The new instances will appear *behind* the "Legal Agreement" section** – this is **expected at this point** as we haven't corrected a couple of typical form design errors. We will cover that in the next section.
13. Enter information into the various fields and use the delete button ("X") to remove a specific instance. The instance removed should be the one to which the delete button belonged.

D. Further Reading

Instance Manager Object Reference
http://forms.stefcameron.com/2006/11/11/instance-manager-object-reference/

Ensuring calculated fields in added/removed instances are properly accounted for in Acrobat 7 or earlier
http://forms.stefcameron.com/2006/05/20/add-recalculate/
http://forms.stefcameron.com/2006/05/25/remove-remerge/

When would you use "addInstance(1)" rather than "addInstance(0)"?
http://forms.stefcameron.com/2006/07/08/demystifying-imaddinstance/

III. Basic Dynamic Form Design Techniques

So far, we've added a schema data connection to the form and enabled the form to generate new registrant instances (up to a maximum of 5 as per the schema's rules) however we saw, in the previous section, how the form's layout is not adapting very nicely when new instances are added. The main source of this problem is the fact that the form has been designed by putting all components into the default **page** subform. In this section, we'll see why we would want to get rid of this subform (for a dynamic form which must be flowable) and how easily we can get rid of it.

A. Repeatable sections (subforms) must be contained within a flowed container

As we saw in the previous section, there are two types of containers (subforms) in XFA: Positioned and Flowed. Objects contained within positioned containers have precise locations and may overlap each other while those contained in flowed containers follow one another (either top-to-bottom or left-to-right) and do not overlap.

To ensure that new instances of the "Registrant Information" section properly flow onto subsequent pages, the section must be contained within a **flowed** container (subform). This way, when new instances are added, the container can grow to accommodate them.

For proper pagination, once you have a section with a repeating subform, it usually follows that all subsequent sections must "move down" on the page (and possibly onto new pages) in order to accommodate the new height of the section as instances of its repeating subform are added/removed. This means that the section (along with all other sections) must itself be contained within a flowed container. Otherwise, the instances of the repeating subform will appear underneath the positioned objects which follow it (as we saw at the end of the previous exercise).

To illustrate this, here is a form with two sections. The first is one which contains a repeating address block and the second contains a simple list box. The outer blue dotted line is what is called the **page subform** in LC Designer ES and it is as big as the entire page. This page subform, with positioned content, contains the two sections.

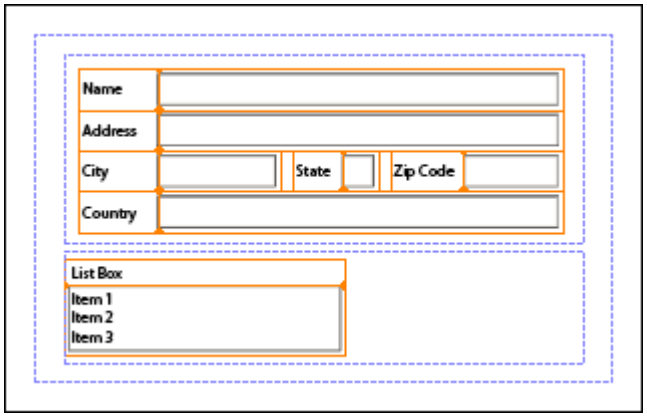

Figure 14: Form with two sections inside positioned page subform

When previewed in Acrobat/Reader, the result, as seen below, is that the instances of the address block appear underneath the subsequent section containing the list box and do not flow onto any new page(s).

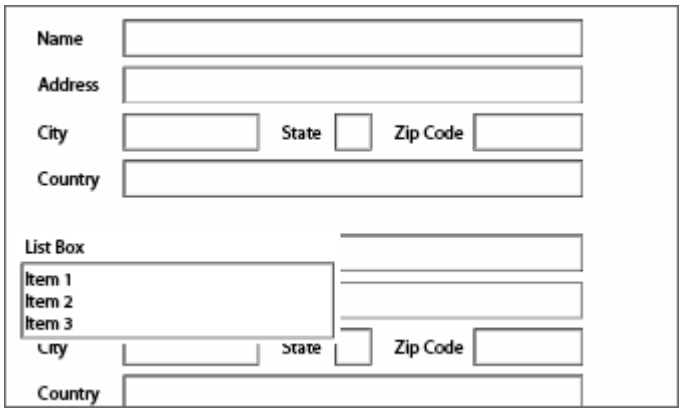

Figure 15: Form with two sections inside page subform as seen in PDF Preview

» **Demo 1:** Show the result of placing dynamic content within the default positioned page subform.

B. Positioned Objects Within Flowed Containers

When a container is flowed, the objects that it contains are either flowed from top to bottom or left to right and you don't have control over specific locations of each object. This causes a problem when you need to mix flowed and positioned objects together.

The solution is simple: Place all the objects that need to be positioned in a container with positioned content and place all containers, flowed and positioned, within a common flowed container.

In a dynamic XFA form, the common flowed container can be the root subform! This is the subform you'll find at the very top of the object tree in the Hierarchy palette (named "form1" by default) which contains the default page subform.

To get started quickly with a new blank form, simply reduce the size of the default page subform and use it as a header, then insert anything extra you'll need after it. By starting-off this way, you'll ensure that any subforms following repeating sections will automatically flow downward onto new pages, if needed.

Tip: To add new sections (subforms) to your form once you've resized the default page subform, use the "Insert > New Page" menu command, resize it and then set its **Place** property to "Following Previous" using the Pagination tab in the Object palette.

Let's go back to our illustration where we have two sections inside the "page" subform. If we remove the superfluous page subform container, we end-up with the following design (where the address block and list box sections are now contained directly inside the root subform).

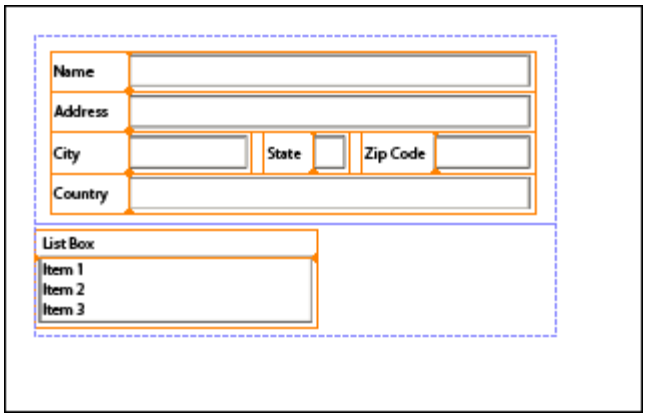

Figure 16: Form with two sections now inside root subform

Since both sections are now contained within a flowed container (the root subform), new instances of the address block subform not only cause the address block section to expand vertically but they also cause the list box section to flow downward such that they no longer appear behind it. Furthermore, since both instances of the address block subform cannot both fit on the same page, a new page is now generated (which is what we wanted in the first place).

Figure 17: Form with two sections inside root subform as seen in PDF Preview

> **» Demo 2:** Rebuild Demo 1 using this technique and show that the positioned objects (now in a positioned subform) below the repeating subform are pushed down when new instances are added.

C. Expandable Fields

Using flowed containers for objects is also very useful when a form has **Expandable Fields**. These objects can either grow in width and/or height to accommodate extra content. This feature is especially useful when your form may be printed as part of its workflow.

By default, field objects don't expand in width and/or height. It's a feature you activate by using the **Expand to fit** option of the **Width** and **Height** properties on the Layout palette. Once you turn-on that option, the field's current size becomes its **minimum** size. When more content is entered into the field than its minimum size will accommodate, the field's size is expanded to fit the content. This way, if the form is printed, you will not lose any of the content because it will not get cut-off.

The key thing to remember about Expandable Fields is that when they expand, they will overlap adjacent objects unless the concept of flowed containers is used to ensure that when the field's size is expanded, adjacent objects/sections are repositioned accordingly.

> **» Demo 3:** Expandable fields in positioned versus flowed containers.

D. Walk-through Three

Objective: Fix the form such that new instances of the Registrant Information section will push the Legal Agreement section downward and adding extra lines of text in the Medical Conditions field (in the Registrant Information section) will cause the field to expand in height to accommodate the extra lines. Finish by importing some data into the form to see the result of all this hard work!

Preview the form in current state – notice reflow problems

1. Open the "ToTheMAX_Registration_**WT3**.pdf" form in LC Designer ES.
2. Preview the form and add instances of the Registrant repeatable subform inside the Registrant Information section. Notice how the new instances show-up behind the Legal Agreement section rather than pushing the Legal Agreement section downward.

Remove the page subform; place all sections inside root subform

3. Back in Design View, open the Hierarchy palette and right-click on the "(untitled Subform) (page 1)" object (the default *positioned* page subform into which all objects were placed).

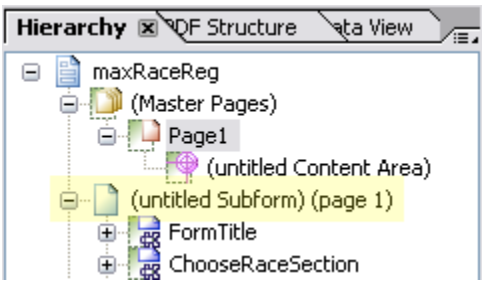

Figure 18: Finding the untitled page subform in the Hierarchy palette

4. Select the "Unwrap Subform" command. This will at once delete the untitled page subform and promote all objects contained within it up one level in the tree. All objects are now contained directly inside the root subform which is a flowed container.

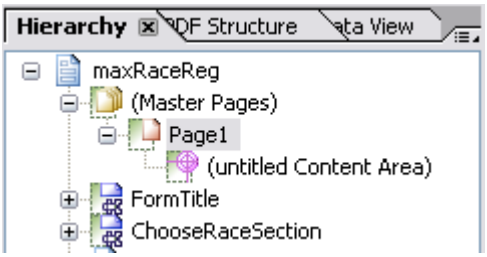

Figure 19: Untitled page subform has been "unwrapped"; all sections now inside root subform

Preview again – reflow issues fixed but "MedCond" field not expanding

5. Preview the form and add instances of the Registrant repeatable subform inside the Registrant Information section. Because the Registrant Information and Legal Agreement containers now reside in the root subform (which is a flowed container), when the Registrant Information section grows to accommodate new Registrant instances, the Legal Agreement section flows downward and onto a new page, if necessary.
6. While still in the preview, add a few lines of text to one of the medical conditions fields in one of the Registrant instances. Click- or tab-out of the field and notice how the field doesn't grow to show all of the text you typed-in (Acrobat indicates that there's hidden data by placing a black "+" sign at the bottom right corner of the field).

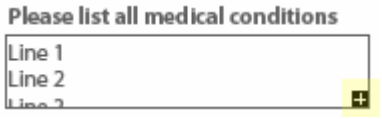

Figure 20: Hidden data in the "MedCond" field

Make "MedCond" field height-expandable

7. Go back to the Design View and select the "MedCond" field in the Registrant subform. It has the caption, "Please list all medical conditions".
8. Open the Layout palette (use "Window > Layout" if necessary) and check the "Expand to fit" box under the Height property.
9. Preview the form once again and add a few lines of text into the medical condition field of an instance of the Registrant subform. Notice, **once you click- or tab-out of the field**, how the field's height is expanded to fit all of the lines you entered and also how the field's height

expansion causes the Registrant subform, Registrant Information section and Legal Agreement section to resize and reflow accordingly.

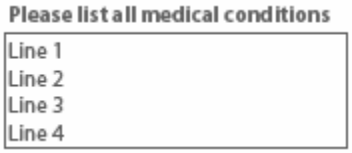

Figure 21: All data visible in now height-expandable "MedCond" field

Preview form with sample XML data

10. Go back to the Design View once again and choose the “Form Properties” command under the top-level “File” menu.
11. Choose the “Preview” tab and specify one of the following data files: “TestData_Full.xml”, “TestData_Half.xml” or “TestData_10k.xml”. The data files are located in the “Data” folder inside the folder containing the walk-through forms.
12. Click on the “OK” button and then preview the form: It should automatically be populated with the data from the preview data file you selected based on the field bindings you specified in the first walk-through exercise.

IV. Generating a Flash-based Form Guide

After following the first three walk-through exercises, we now have a completed form which can be used to register for a race. The typical workflow would see the applicant opening the form as a PDF file in their browser, filling it and submitting the information back to an email address for someone to process.

That’s all fine but let’s face it: When it comes to pizzazz, a form can be pretty dry and you want people registering for the races to be **engaged** and excited about being part of a fantastic event!

New to the LiveCycle Enterprise Suite is the integration of **Flex technology** which lets you quickly re-purpose a PDF form as a **Flash-based** Rich Internet Application called a **Form Guide** and it can be done right there in LC Designer ES: When you have an open document, you’ll find a new command under the top-level “Tools” menu called “Create or Edit Form Guide”. Selecting this command will start the **Guide Builder** plug-in, a Flex-based Form Guide editor.

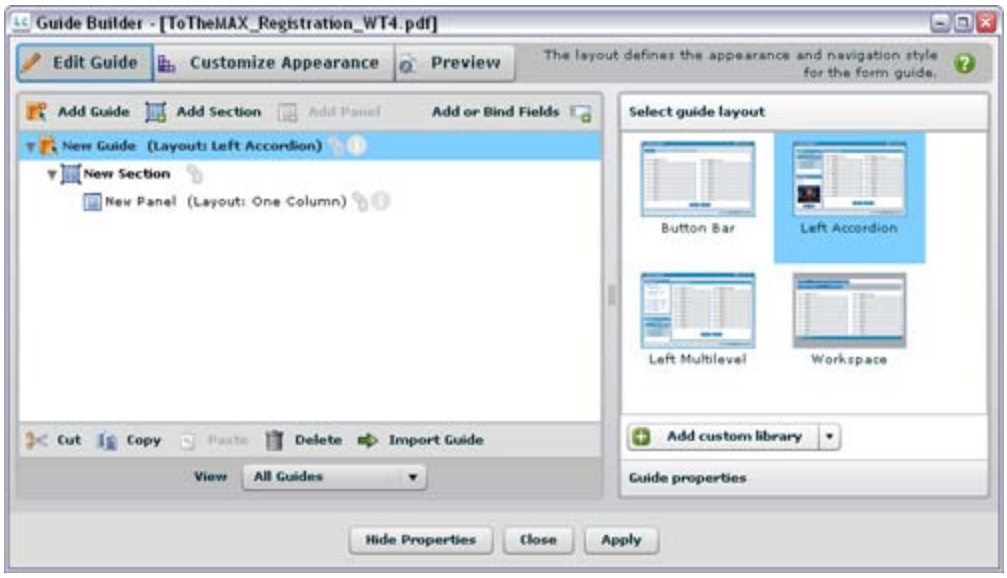

Figure 22: Guide Builder Plug-in

A form may have multiple Form Guides each organized in a series of sections containing panels of various types. Further customization may be done within Guide Builder as well as with Flex Builder (for the more advanced).

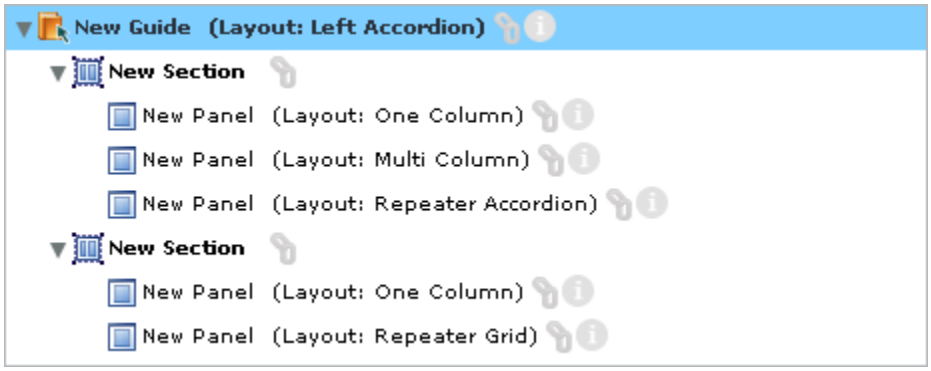

Figure 23: Form Guides are composed of sections and panels

In order to facilitate the design of the guide(s) associated with a form, changes to most form object properties are automatically reflected thereby reducing the amount of work required to keep guides updated as per changes to the form. It's also possible to create explicit links of certain form object properties to guide object properties. For instance, you can link a form object's caption to a guide object's caption such that changes to the caption on the form are automatically replicated on the linked guide objects (otherwise, you may specify captions on guide objects which are different from those set on the form objects which they represent and these are not automatically updated).

Figure 24: Form Guide section caption linked to form object caption

The Guide Builder plug-in is designed to make it very easy to create one or more form guides based on a given form. The idea is to use the richness of the Flash environment to create an alternate "view" or "rendering" of your form which is more interactive, more intuitive and more user-friendly than an electronic PDF form might be. For example, when filling-out an insurance claim form for a car accident, it can be very complicated to specify what kind of damage was sustained at particular locations on the vehicle. In this case, a Form Guide could be designed to present a more intuitive form-filling experience such as an illustration of a vehicle with "hotspots" on the doors, hood, trunk and bumpers where the user could choose from a list of various damage types (e.g. "scratches", "dent", "chipped paint", etc.). The guide would then take care of properly filling-out the associated form fields with the correct information, reducing input errors.

Since a Form Guide is an alternate "view" or "rendering" of a form, **LiveCycle Forms ES** is required to deploy a form as a Form Guide although the LC Designer ES Guide Builder plug-in provides a **preview** feature so that you can preview your work prior to deploying your form to the server. Form Guide definitions are stored in the form itself and LC Forms ES will serve the form out to the browser using the Form Guide definition if it is requested.

A. Form Guide Repeater Panels

Before we dive into the final walk-through and build a Form Guide, it is important to note how repeater panels work. Repeater panels are used to manage the instances of repeatable subforms within a Form Guide much in the same way the add and delete buttons are used on the registration form.

Figure 25: Form Guide repeater panel (accordion-style) for "Registrant" subform

When you specify that a panel is a "repeater accordion/grid/tab", Guide Builder will **assume** that the **parent subform of the first field which you allow to repeat** within the panel is the repeatable subform which contains the field and has the Instance Manager which controls the instances.

Due to the complex layout of the registration form we have been working on, a more complex subform hierarchy was necessary within the "Registrant" repeatable subform. The result is that the fields we want to repeat within the panel are inside the "Info" subform which is itself inside the "Registrant" repeatable subform. If we were to make the "FirstName" field repeatable, Guide Builder would setup the repeater panel to use the "Info" subform's Instance Manager which we know is incorrect because the "Info" subform isn't repeatable.

In order to work around this issue, it's necessary to have a field directly parented to the "Registrant" repeatable subform which we can use within the repeater panel to "trick" Guide Builder into choosing the correct repeatable subform for any fields it contains which are allowed to repeat. This is the reason behind the hidden "GuideRepeaterHint" button located just under the "Registrant" repeatable subform. The button is hidden so that it doesn't have any impact on the form's layout.

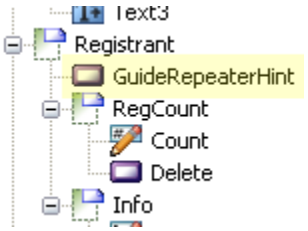

Figure 26: Hidden "GuideRepeaterHint" button in Hierarchy palette

When we configure the repeater panel later on in the walk-through, we'll make sure the "GuideRepeaterHint" field is given the permission to repeat **before** any of the fields inside the "Info" subform are made repeatable.

B. Deploying the Forms-IVS Test Server

Before we proceed to the last walk-through, I thought I would mention the basics about using the LiveCycle Forms ES Installation Verification Sample (Forms-IVS) test server that comes with LiveCycle Forms ES. Forms-IVS will allow us to quickly test the deployment of the registration form as a Form Guide in the browser with the alternate PDF view running in the background, being filled as information is entered into the Form Guide.

Once LC Forms ES has been installed as part of a LiveCycle ES installation, deploying Forms-IVS is quite simple (assuming here that you're using JBoss as your application server):

1. Locate the "adobe-forms-ivs-jboss.ear" file in the "…\LiveCycle8\deploy" folder and copy it into "…\LiveCycle8\jboss\server\all\deploy". Since JBoss allows hot deployments, that is all you need to do in order to deploy Forms-IVS.

2. You can confirm the successful deployment of Forms-IVS by verifying the JBoss server log (found in "…\LiveCycle8\jboss\server\all\log\server.log") afterwards. You should see entries similar to the following near the end of the file:

```
2007-07-19 12:29:25,533 INFO  [org.jboss.web.tomcat.tc5.TomcatDeployer] deploy,
ctxPath=/FormsIVS, warUrl=.../tmp/deploy/tmp30946adobe-forms-ivs-jboss.ear-
contents/FormsIVS-exp.war/

2007-07-19 12:29:26,238 INFO  [org.jboss.web.tomcat.tc5.TomcatDeployer] deploy,
ctxPath=/FormsIVS/Help, warUrl=.../tmp/deploy/tmp30946adobe-forms-ivs-
jboss.ear-contents/FormsIVS_help-exp.war/

2007-07-19 12:29:26,379 INFO  [org.jboss.deployment.EARDeployer] Started J2EE
application: file:/C:/Adobe/LiveCycle8/jboss/server/all/deploy/adobe-forms-ivs-
jboss.ear
```

3. Launch Forms-IVS simply by opening a browser window and navigating to http://<serverName>:8080/FormsIVS

C. Walk-through Four

Objective: Use the Guide Builder plug-in tool to quickly re-purpose the registration form as a Flash-based Form Guide and use Forms-IVS to deploy the form as a guide to a web browser.

Create a new Form Guide based on application form

1. Open the "ToTheMAX_Registration_**WT4**.pdf" form in LC Designer ES.
2. Choose the "Create or Edit Form Guide" command under the top-level "Tools" menu.
3. Click on "Add or Bind Fields" in the toolbar and link the "Title" text object from the "FormTitle" subform to the caption of the guide (with a default caption of "New Guide") by dragging and dropping the "Title" text object onto the link icon next to the "New Guide" object.

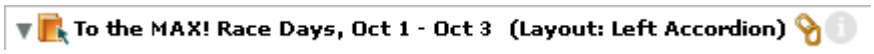

Figure 27: Guide caption linked to "Title" text object value

4. Select the default section (which has a default caption of "New Section") and press the F2 key to set its caption manually to "Registration Form".

Create the "Choose Your Race" panel

5. Link the "Text2" object from "ChooseRaceSection" subform to the panel's caption.
6. Drag each **individual** radio button into the first panel (you can multi-select them to do this in one operation). We don't use the radio button list "RaceTypeList" as a whole because we need to link objects to each radio button's caption in the guide since the radio buttons don't have captions in the form.
7. Link the "AdobeRace", "FinanceRace" and "SideStitchRace" text objects to the captions of the "Marathon", "HalfMarathon" and "Race10k" radio buttons, respectively.

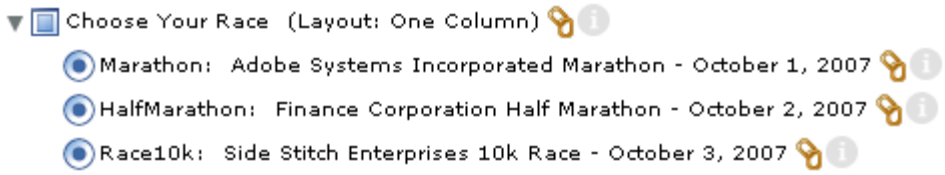

Figure 28: Panel and radio button captions linked

Create the "Registrant Information" repeater-accordion panel

8. Select the "Registration Form" section and click on the "Add Panel" button in the toolbar at the top to add a new panel for the Registrant Information.
9. Select the new panel and link the "Text2" text object from the "RegistrantSection" subform to its caption.
10. With the panel still selected, click on "Set Properties" in the toolbar and specify a "Repeater Accordion" layout. This will let the panel handle new instances of registrant information and will provide the "add and remove" functionality for instances like we have on the form.

Figure 29: "Set Properties" button in the Guide Builder toolbar

11. Click on "Add or Bind Fields" in the toolbar and drag the "GuideRepeaterHint" button, found under the "Registrant" repeatable subform, into the "Registrant Information" panel.
12. Select the "GuideRepeaterHint" field in the panel and click on "Set Properties" in the toolbar.
13. Put a check mark in the "Allow field to repeat" property. This will tell Guide Builder that the button should repeat and will ensure that Guide Builder identifies the "Registrant" subform, the button's immediate parent subform, as the repeatable subform which has the Instance Manager that will be used to add and/or remove instances. (You'll also notice new "Next Area" objects that get added to the panel. They are there to allow you to specify a header, footer and second column within the "Repeater Accordion" panel if you should need to do so.)

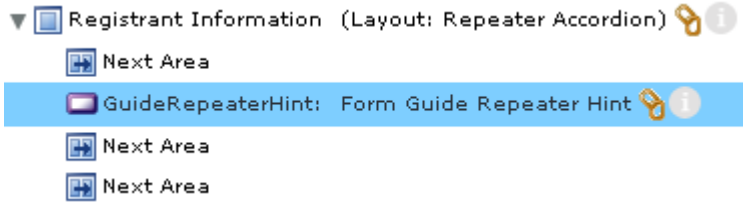

Figure 30: After inserting "GuideRepeaterHint" button into panel and making it repeatable

14. Click on "Add or Bind Fields" in the toolbar and drag all the fields from the "Info" subform into the "Registrant Information" panel next to the "GuideRepeaterHint" button. You'll find the "Info" subform inside the "Registrant" subform which is inside the "RegistrantInfo" section. Make sure that the fields end-up between the first and second "Next Area" guide objects, just like the "GuideRepeaterHint" button.

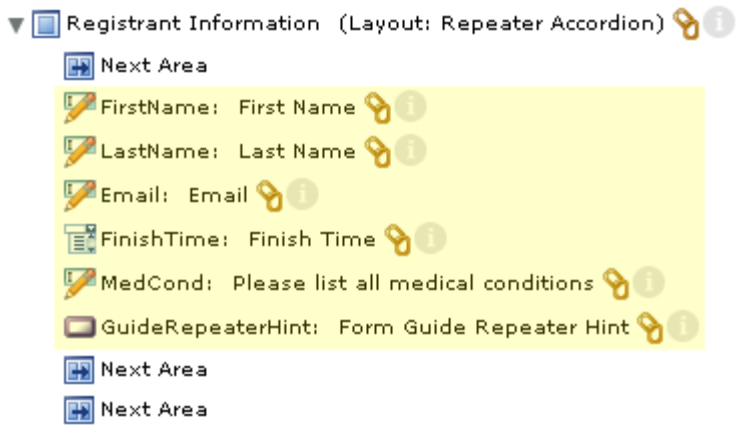

Figure 31: All repeatable fields between first and second "Next Area" guide objects

Create the "Agree and Submit" panel

15. Select the "Registration Form" section and add a new panel.
16. Use the F2 key to manually set the panel's caption to "Agree and Submit".
17. With the "Agree and Submit" panel selected, right-click and choose the "Add Guide Text" command.
18. Select the new "Guide Text" object, click on "Set Properties" in the toolbar and set its text as follows: "To complete your registration, please click on the 'Submit from PDF' button at the bottom of this guide and locate the 'Legal Agreement' section."
19. Select the guide object itself (at the top of the guide/section/panel hierarchy) and, with the Properties panel still visible on the right, select the "Guide properties" panel at the bottom and set the "Submit from" property to "PDF". This will cause a "Submit from PDF" button to be displayed on the last panel of the guide once all required fields have been filled. The user will then click on this button to load the pre-filled PDF form and, after accepting the legal terms, they will be able to submit the form using Adobe Acrobat or Reader.

Preview the completed Form Guide

20. Preview the guide to make sure that it's working correctly. You should be able to select a race type in the first panel, add or remove registrant information in the second panel and submit the data to a "mock" PDF form (when deploying the form as a Form Guide via LC Forms ES, this would be the actual PDF form). Try to skip to the "Agree and Submit" panel without filling a registrant field: You should get an error message stating that some required fields need to be filled prior to submitting the form. This is yet another way in which properties defined on form objects (such as required fields) are automatically carried-over to objects representing these fields in Form Guides.

Save Form Guide to form for Forms-IVS deployment

21. Return to the Guide Builder plug-in and click the "Apply" button to save the guide definition to the form.
22. Close Guide Builder, save and close the form.

Deploying the Form Guide with Forms-IVS

23. Load Forms-IVS in your browser (i.e. Internet Explorer): http://<serverName>:8080/FormsIVS
24. Click on the "Maintenance" button in the top-level toolbar.

Figure 32: Forms-IVS "Maintenance" button (highlighted)

25. From the "Manage form designs" section, upload the registration form containing the Form Guide.
26. When you saved the Form Guide into the registration form, Guide Builder generated a Flash Component File (with extension ".swc") in the same folder as the form. This file contains style information required to render the Form Guide properly and also needs to be uploaded using the "Manage form designs" section.
27. From the "Manage data files" section, upload the three sample XML data files found in the "Data" folder: "TestData_Full.xml", "TestData_Half.xml" and "TestData_10k.xml". These files will serve as option data files which you can use to initialize the registration form.
28. Click on the "Test forms" button in the top-level toolbar. You should then see a page with "Data files" and "Form designs" list boxes containing the files (less the ".swc" file) that you uploaded in the previous steps.

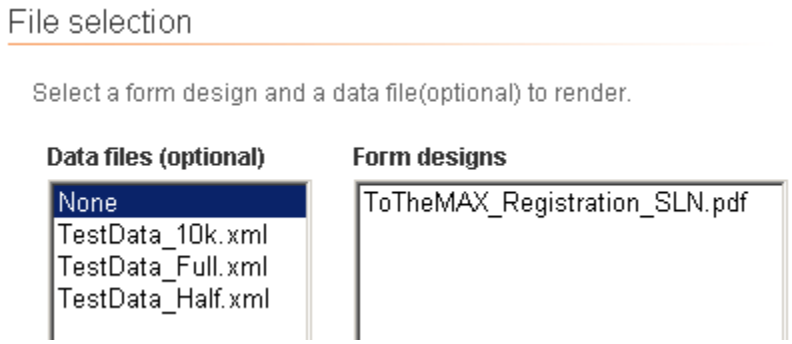

Figure 33: "Data files" and "Form designs" list boxes in "Test forms" view

29. Select the registration form in the "Form designs" list box, choose "Guide" as the "Output format" below and click on the "Use EJB" button to deploy it as a Form Guide.
    a. You may optionally choose a data file to be merged into the form as well.
30. Once the Form Guide appears in the browser, experiment with filling fields in the Form Guide and the PDF views by using the "Show/Hide PDF" button in the top-right corner toolbar. Data filled in the Form Guide is automatically replicated in the PDF and vice versa.

Solution File: The "ToTheMAX_Registration_**SLN**.pdf" form contains the Form Guide completed as per Walk-through Four. You'll also need the "ga.wrappers.LeftAccordion.swc" Flash Component File found in the same folder if you intend on deploying the solution file to LC Forms ES or Forms-IVS.

D. Further Reading

Check-out the "Using Guide Builder" topic in LC Designer ES' Help System.

Anthony Rumsey's "Form Nation" blog on developing "dynamic and rich applications using Adobe LiveCycle Forms and Flex."
http://blogs.adobe.com/formnation/

V. Additional Resources

MAX 2007: J.P. Terry's (SmartDoc Technologies) hands-on session on "Designing Dynamic Forms with LiveCycle Designer 8". This session will deal more in-depth with **form design techniques**.

MAX 2007: Anthony Rumsey's (Adobe) session on "Everything You Want to Know about **LiveCycle Form Guides**".

J.P. Terry's (SmartDoc Technologies) book on LC Designer : "Creating Dynamic Forms with Adobe LiveCycle Designer", available at the MAX Store and on Amazon.

Adobe LiveCycle Developer Center
http://www.adobe.com/devnet/livecycle/designing_forms.html

Adobe LiveCycle ES Online Documentation (containing links to LC Designer ES Live Docs Help, Scripting Basics and Scripting Reference under the **Develop tab**)
http://www.adobe.com/support/documentation/en/livecycle/

Transformation Reference (XML Form Object Model subset supported in Form Guides)
http://www.adobe.com/go/learn_lc_transformation

Anthony Rumsey's blog on Flex and LiveCycle Forms ES
http://blogs.adobe.com/formnation/

My blog on LiveCycle Designer ES
http://forms.stefcameron.com/