

# Improving the Performance of Adobe® LiveCycle™ Designer Forms

These guidelines are intended to help you create and maintain Adobe® LiveCycle™ Designer forms for optimal performance. These guidelines also help you diagnose and correct performance-related issues.

Performance considerations depend on the type of form you are designing. This document describes considerations for two types of forms:

**Interactive forms** are opened by users in Adobe Acrobat® Professional or Acrobat Standard, or Adobe Reader®, and are completed online.

**Non-interactive forms** are usually printed by users and completed by hand. Non-interactive forms may also be merged with data by Adobe LiveCycle Forms. However, no data is provided online by the user.

**Note:** This document refers to LiveCycle Forms as the component that performs Adobe services related to form and document creation and manipulation. For users working in the SAP environment, LiveCycle Forms provides the same functionality as Adobe document services.

This document includes the following sections:

- “Document Rendering Considerations” describes design options that improve performance when LiveCycle Forms renders interactive and non-interactive forms. Rendering a form is the process of merging a form design, possibly with data, to display a form in Adobe PDF or HTML format in a browser. LiveCycle Forms, not a servlet or browser, renders the form. Note that, in the SAP environment, forms can be displayed only in PDF in a browser.
- “Print Considerations” describes design options that improve performance for printing non-interactive forms.
- “Interactive Form Considerations” describes design options that improve performance when submitting data from interactive forms to LiveCycle Forms or when refreshing data.

## Form Rendering Considerations

Design choices affect the performance of LiveCycle Forms when it renders interactive and non-interactive forms. This section describes considerations for structuring data, designing forms, and reducing file size.

### Structuring Data

Forms can be merged with data from data sources such as XML schemas and databases. The following tips are intended to help you design forms based on the

## TABLE OF CONTENTS

Form Rendering Considerations.....	2
Print Considerations.....	7
Interactive Form Considerations .....	8

structure of the data that will be merged. This approach optimizes performance when LiveCycle Forms merges data with forms.

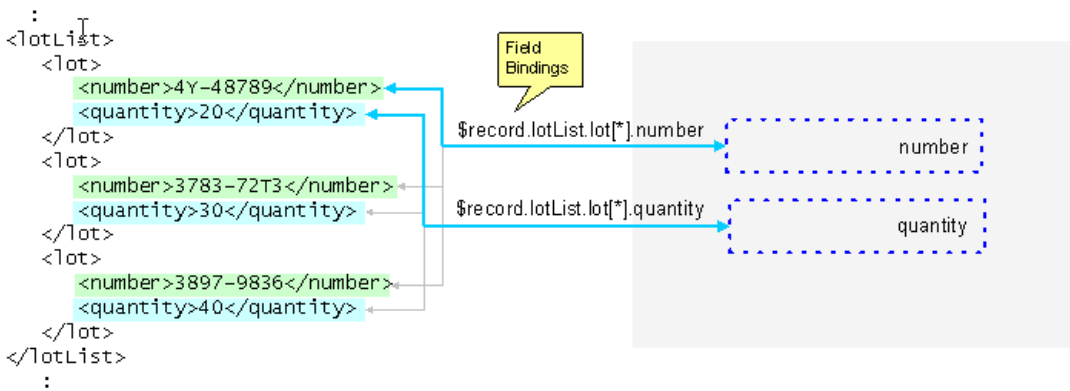
### Data Connections and Data Binding References

You can associate a data element with a form object, such as a field or a subform. This association is called *data binding*. You can choose objects that shrink or grow depending on the amount of data they display.

Many data binding definitions can produce the same results when a form is rendered. However, the more obvious the relationship between the form object and its data element, the more efficiently LiveCycle Forms can process the data binding.

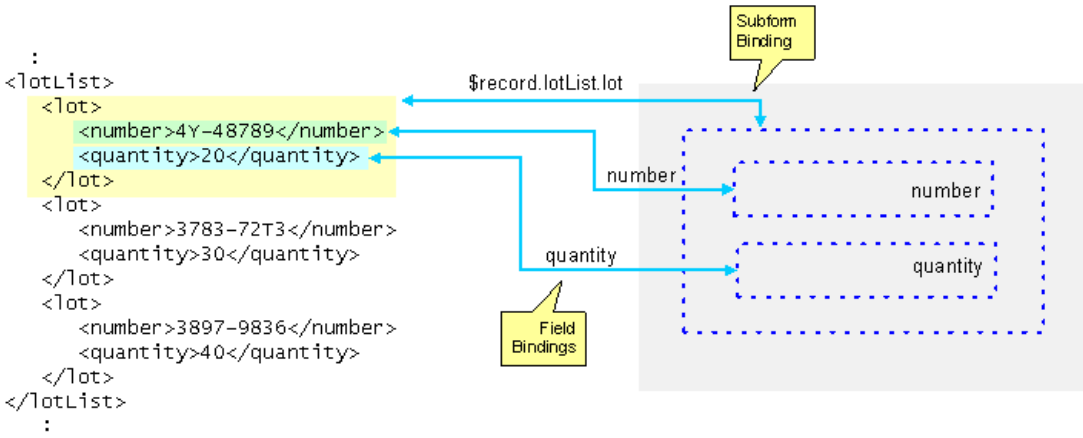
The following diagram shows a field binding example where two field objects in the form are bound to data elements. This binding is ambiguous because multiple instances of the field objects and their data exist. More processing is required to match the *n*th occurrence of each object to the *n*th occurrence of the corresponding data when rendering the form.

#### Ambiguous field binding



The following diagram shows the use of subform binding to group objects and thereby simplify the processing to render the form. The subform is bound to the repeating group in the data, eliminating the ambiguous binding.

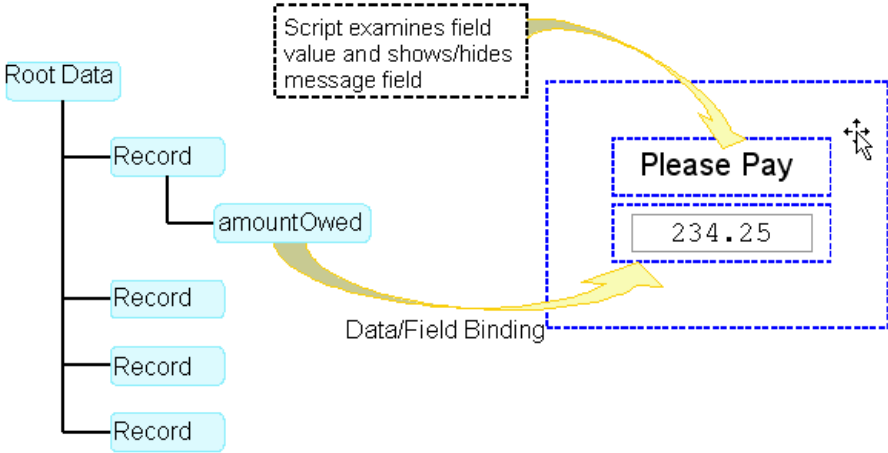
#### Binding with subforms



### Selective Field Display

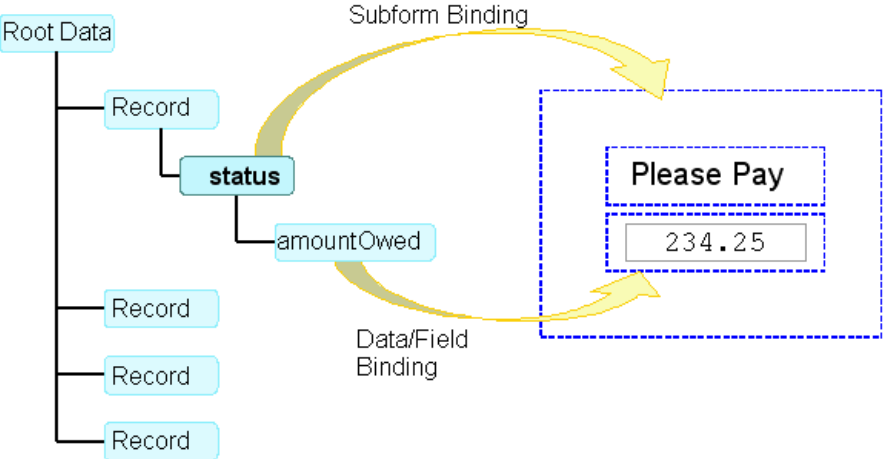
Selective field display is one way to demonstrate how data structure affects performance. Assume that we want to display a message only if the value of the amountOwed data element is greater than zero. One approach is to use a script to examine the value of amountOwed and alter the show/hide property of the message field, depending on the value of the data element.

#### Selective field display using a script



Alternatively, you could bind the data to a subform. Binding data to a subform is more efficient than binding data to a field, although both methods produce the same result. Here, we add the status element to the data structure and bind this element to a subform. The subform includes the message field and the amountOwed field. With the minimum count of the subform set to zero, the message appears only if the status element is populated.

#### Selective field display using subform binding

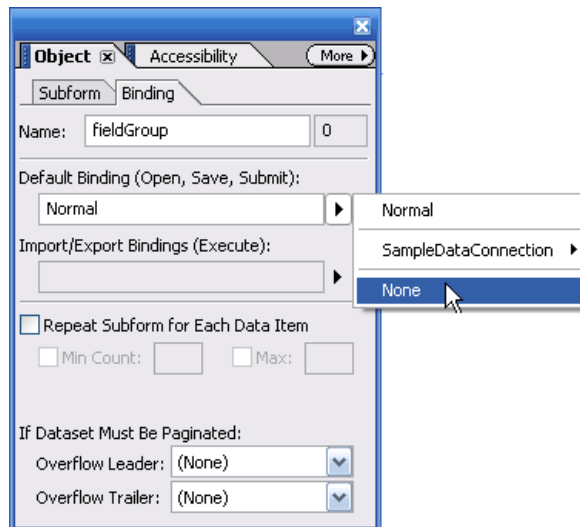


In the data context, the decision to add the status node to the data file should be done at generation time, only if the value is greater than 0. This technique moves the logic to determine whether a subform is required in the data context design.

## Subforms Not Bound to Data

You can use subforms to group and organize objects without binding the subform to a data element. To prevent LiveCycle Forms from searching for a data element for the subform when merging data, change the subform's data binding type from the default value of Normal to None.

### Turning off subform binding



## Designing Forms

The following design guidelines help optimize performance when LiveCycle Forms renders forms. Many of these guidelines also make your forms easier to maintain.

### Number of Objects

Keep the following tips in mind to minimize the number of objects on forms:

- When creating boxes, use a single rectangle object instead of joining four individual lines.
- When creating a border for a field or subform object, use the object's border attributes instead of creating a separate box.
- When creating backgrounds for objects, use a background fill instead of creating a separate shaded box object.
- For captions, use an object's caption property instead of using a separate static text object.
- Combine multiple static text objects into a single text object. This strategy is particularly useful after importing forms into Designer. For information about importing forms, see "Imported Forms".

### Complexity of Objects

Keep the following tips in mind to minimize the complexity of objects on forms:

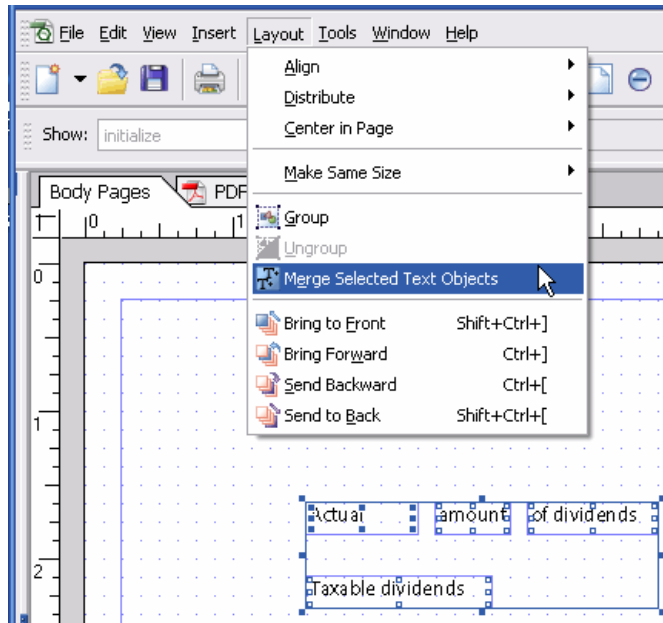
- Keep the number of fonts, styles, and sizes to a minimum to take advantage of cached font information. For other tips about using fonts, see [Text Field Objects](#) and [Fonts](#).
- Use fixed objects instead of dynamic objects, if possible. Fixed objects are processed more efficiently than dynamic objects.
- Avoid layering objects, especially more than three layers deep.

## Imported Forms

When importing a form into Designer, the imported form may include an unnecessary number of objects. For example, when importing a PDF form, one static text object may be created for each line or word in the original form. Similarly, additional objects may be created to produce layers. For example, a background may be imported as a shaded rectangle under an object, and borders may be imported as individual lines.

To reduce the number and complexity of objects on imported forms, see the tips provided in “Number of Objects” and “Complexity of Objects”. To further reduce the number of objects on the form, combine related text objects by clicking Layout > Merge Selected Text Objects.

### Merging text objects



In addition, choosing the Maintain Editability option or the Preserve Appearance option when importing the form produce different results in terms of the number and complexity of objects on the resulting form. You may find it useful to import the form twice—once using the Maintain Editability option and once using the Preserve Appearance option. You can produce the final form by combining the most appropriate elements from each resulting form.

## Text Field Objects

Allow plain text entries only for text field objects. The font, style, and size used at design time are applied to user input. The form can be rendered more efficiently by using the text properties used in the form design.

For rich text entries, users can customize the font style and size for text entered in the object. Although rich text entries may be useful in some situations, such as for narrative information, their additional attributes decrease performance.

## Radio Buttons

Several object types provide discrete option choices: check boxes, list boxes, and radio buttons. List boxes and check boxes are rendered more quickly than radio buttons.

## Master Page Objects

Limit the number of objects in master pages because these objects require additional processing time. If possible, place objects on body pages.

## Subforms

Designer distinguishes boilerplate or static objects, such as text, lines, and images, from objects that contain variable content, such as text fields and image fields. For example, interactive forms can include static text labels that provide information to the user and text field objects that capture information from the user.

A subform is a type of object that contains content. Subforms are often grouped based on the structure of the data that is bound to each subform. The ability to bind repeating, optional, and conditional data groups to subforms reduces the risk of design errors that may occur if you used scripts to achieve the same results. For example, you can create a subform that includes objects for repeating data groups. When rendering the form, LiveCycle Forms creates as many instances of the subform as necessary to represent all the data groups.

When using subforms, keep the following in mind for optimal performance:

- Repeating and nested subforms require additional processing to render the form. Avoid using them unless they offer better performance than alternative options. For example, a repeating subform may eliminate the need for a number of repeating objects.
- Allowing page breaks in subforms causes additional processing, even if LiveCycle Forms does not apply page breaks. For example, the location, size, or content of a subform may prevent a page break. To optimize performance, turn off page breaks in Flow Content type subforms. By default, page breaks are allowed.

### Preventing page breaks within subforms

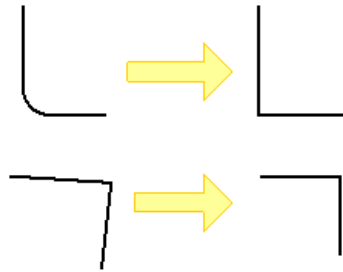


## Images

Images require additional processing time to render forms. If you choose to use images, keep these performance tips in mind:

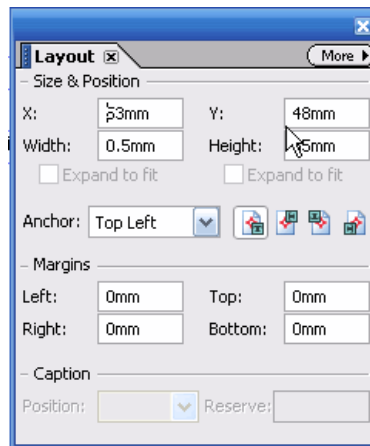
- Keep the file size as small as possible by using compact file formats and the smallest possible image dimensions. Size the image to the intended dimensions before adding it to the form rather than after adding it to the form.
- Use square corners and lines that are exactly horizontal and vertical.

## Avoid curves and sloped lines



Lines that appear to be exactly vertical or horizontal may have a slight slope. To check the slope of a horizontal line, use the Height option in the Layout palette. To check the slope of a vertical line, use the Width option in the Layout palette. In the following example, a vertical line has a small slope of approximately 0.5 millimeters. Changing the width to 0mm makes the line exactly vertical.

## Preventing sloped lines



## Accessibility

The tagging applied for accessibility when rendering forms increases the file size, which reduces performance. If you choose to render forms for accessibility, choose suitable design elements. For example, choose list boxes or check boxes instead of radio buttons. For more information about these types of design elements, see [Radio Buttons](#).

## Fonts

To minimize the file size of rendered forms and improve performance, keep the number of fonts to a minimum and avoid fonts that must be embedded. The following fonts are always available in Adobe Reader and are never embedded:

- TimesRoman, TimesBold, TimesItalic, TimesBoldItalic
- Helvetica, HelveticaBold, HelveticaOblique, HelveticaBoldOblique
- Courier, CourierBold, CourierOblique, CourierBoldOblique
- Symbol, ZapfDingbats

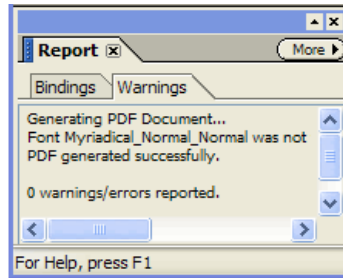
## Warnings and Error Messages

Some events, such as font substitutions, generate warning or error messages, which reduce performance. Try to resolve as many messages as possible, although some messages are generated even when forms are working properly.

You can check for warning and error messages during two phases:

- You can view messages generated by Designer while you design the form. Messages appear in the Warnings tab of the Report palette.

### Checking for warnings and messages



- If a form generates warning and error messages when it is rendered by WebDynpro, you can access the messages in Designer in the Report palette. You can also see these messages in the LiveCycle Forms log files.

It is recommended that you always review both types of generated messages. Even if Designer does not generate messages, the LiveCycle Forms environment may be significantly different than your desktop environment. For example, fonts available on your desktop may not be available to LiveCycle Forms.

The following list provides examples of common messages generated when LiveCycle Forms renders the form.

**Note:** Some of these examples refer to tables. Currently, tables are available only in the SAP environment.

- “Fonts ZaDb was not found. It was degraded to Myriad Pro.”  
The form uses a font that is not available to LiveCycle Forms and an available font was substituted. If the form appears correctly using the substitution font, you can eliminate this error by changing the form objects to use the substitution font. You can also resolve this error by installing the missing font for LiveCycle Forms.

- “Script failed (language is JavaScript; context is...)”  
The script cannot execute because of scripting errors. If the script does not produce results in the form, this error may not be detected when the form is tested. However, generating the error affects performance.

The following example shows a script error:

```
script = function testFractionDigits(maxDigits)
{
  index = this.rawValue.lastIndexOf(".");
  if (index >= 0)
    return (((this.rawValue.length1)index) <= maxDigits);
  return true;
}
```

In this example, `index` is an undefined variable. To resolve this error, add `var` in front of the first instance of `index`.

- “Fonts used in fields cannot be subset. Font ArialMT not subset and will be embedded.”  
A font cannot be subset. To resolve this error, use a standard font to eliminate font subsetting. This change also reduces the size of the file produced.

- “ImageField access attribute should explicitly be set to 'nonInteractive'. Field will be drawn as boilerplate.”  
An image field object is rendered as non-interactive in Acrobat 6.0 and Adobe Reader 6.0 forms. To resolve this message, convert the object to a static image object.
- “Invalid layout attribute on subform AccountDetails. Using default.” (SAP only)  
A subform has a row layout type that should be nested inside a table parent, but it is not enclosed in a table. To resolve this error, move this subform into a table or change the subform type to Position Content or Flow Content.

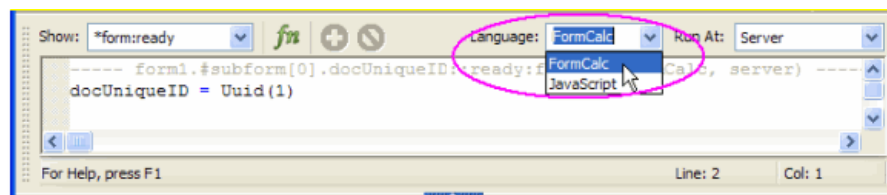
## Script Languages and Other Script Considerations

You can use JavaScript or FormCalc for calculations and validations in forms. FormCalc usually provides better performance than JavaScript for the following reasons:

- FormCalc executes simple calculations and validations more quickly.
- FormCalc interprets the XML Form Object Model syntax directly, which means it can evaluate Scripting Object Model (SOM) expressions more efficiently.

You select the script language using the Script Editor.

### Selecting the scripting language



In general, scripts increase the processing required to render forms. If possible, avoid using validation scripts. Also, before using scripts, determine if there is a better performance alternative. For example, character masks offer better performance than scripts, with the same results.

## XML Form Object Model Expressions

The Adobe XML Form Object Model uses an approach to building forms that distinguishes among form layout, the form design, and the form data. In a form design, SOM expressions associate objects with values, objects, and properties within the tree hierarchy of a Document Object Model (DOM). SOM expressions are usually written into form logic, in validations, calculations, or other business rules.

Keep these tips in mind to reduce the processing time required for resolving SOM expressions:

- Keep expressions as simple as possible.
- Avoid using “.” in expressions.
- After evaluating a SOM expression in a script, store the result in an object variable. This result can be reused if the SOM expression is used again, which is more efficient than re-evaluating the expression.

In the following example, `xfa.resolveNode()` is used once, and the resulting object reference is saved and reused to manipulate various related objects:

```
var oSubform = xfa.resolveNode("xfa.form.form1.Subform1");

if (oSubform.all.length < 3) {

    oSubform.SubformUpButton1.presence = "invisible";
    oSubform.SubformLabel11.presence = "invisible";
}
```

```
oSubform.SubformUpButton2.presence = "invisible";
oSubform.SubformLabel2.presence = "invisible";

}
```

This script can be processed more efficiently than one that uses a SOM expression for each object and uses the `xfa.resolveNode()` method on each one.

## Reducing File Size

The file size of forms can affect performance, particularly when they are transferred over slow communication channels or stored and retrieved from near-line storage.

The following tips may help minimize the file size of forms:

- Avoid using fonts that must be embedded. If you must use embedded fonts, avoid using them for field objects because embedded fonts cannot be subset.
- For interactive forms, embed images. By default, images are linked.
- Use Palette or Monochrome colors for images. Avoid using 24-bit color formats.
- Be aware that generating accessible forms affects performance.
- Design forms so that LiveCycle Forms can render them as non-interactive. Avoid using objects intended for interactive forms if a form does not require them.
- Use only the minimum number of objects in forms, therefore avoiding unnecessary objects.
- For interactive forms, use the XML Data (XML) format for exchanging data with LiveCycle Forms. For more information about formats for exchanging data, see [Submit XML Data Only](#).

## Print Considerations

This section describes design options that improve performance for printed forms.

### Non-Interactive or Interactive

In general, the smaller file size of non-interactive forms means they can be rendered faster than interactive forms. For forms intended for printing, avoid using objects or properties such as data entry validations and edit masks that cause LiveCycle Forms to produce interactive forms.

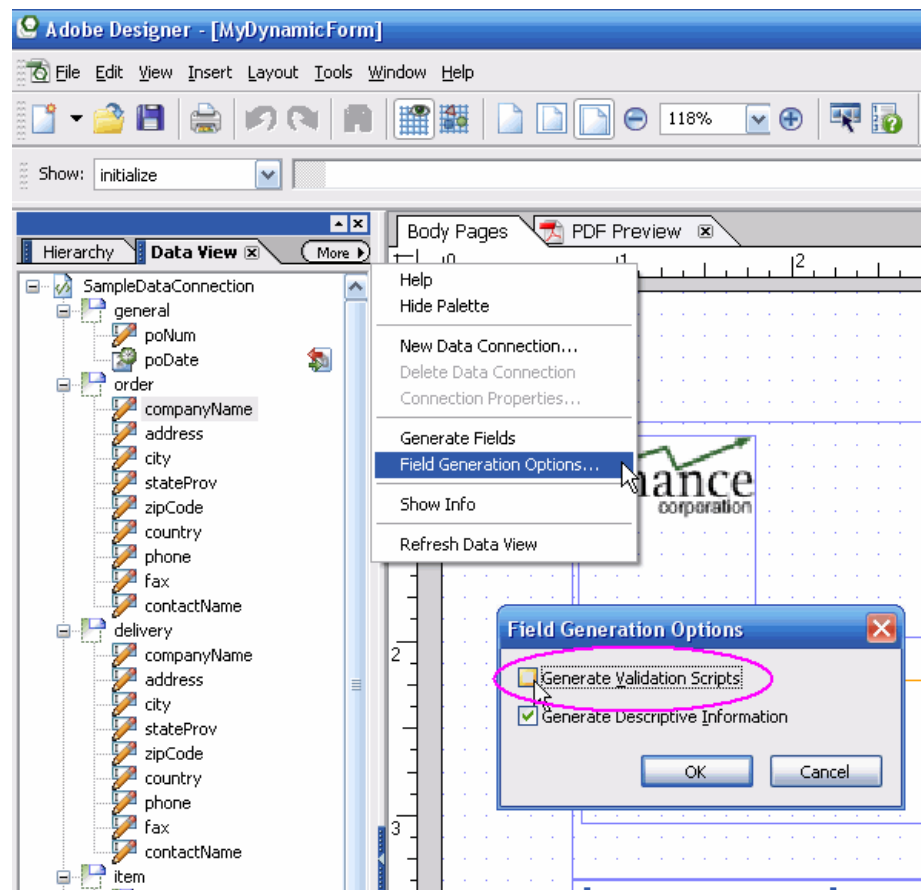
#### Avoid Validation Constructs

For forms that are bound to XML schemas, Designer can generate validation scripts for particular objects. These validation scripts ensure that data entered by the user complies with the schema specification. For example, a validation script may check that a value provided for a numeric object is an integer between 4 and 9.

For interactive forms, validation scripts are useful for checking data when the user enters it. For non-interactive forms, these scripts provide no benefit because they do not accept user input. Turning off the generation of these scripts improves performance by eliminating the processing to execute them.

To turn off validation script generation, click Data View > Field Generation Options.

## Turning off script validation



### Batch Documents

When rendering multiple instances of a single form, such as bills for different customers, gather the data for all the forms in a single data file and render the individual forms from this file. This approach is more efficient than processing forms individually.

However, when batching documents, group similar forms before sending them to be generated. For example, if Packing Lists and Invoices are run at the same time, group and run all the Invoices first, followed by all the Packing Lists.

## Interactive Form Considerations

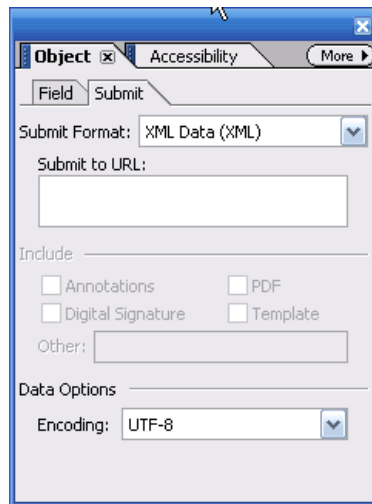
This section describes design options that improve performance when data is submitted from interactive forms to LiveCycle Forms.

### Submit XML Data Only

When choosing the format of the data to submit to LiveCycle Forms, use the XML Data (XML) format. Other formats, such as PDF, require more bandwidth and processing time.

To choose the format of the data to submit, use the Object palette. On the Submit tab, click Submit Format and select XML Data (XML).

## Choosing the format of submitted data

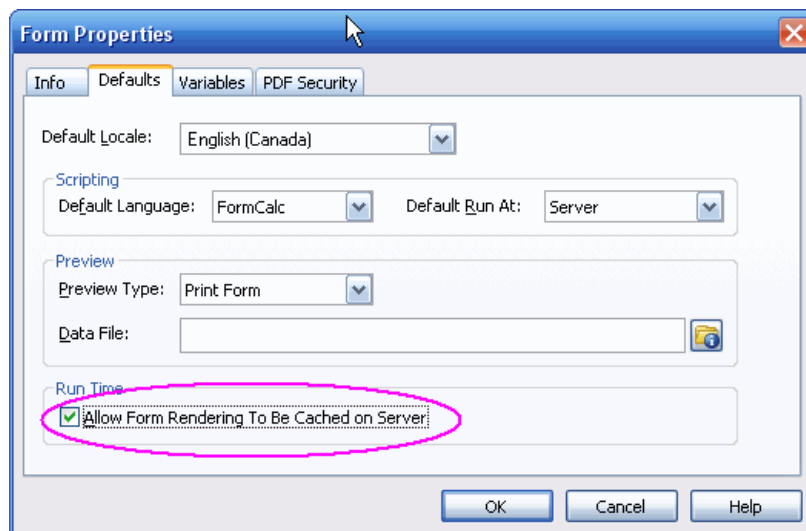


## Cache Forms

LiveCycle Forms renders forms in numerous formats through transformations. For performance reasons, LiveCycle Forms caches transformations when rendering them. For LiveCycle Forms to cache non-interactive forms properly, you must enable form caching when you create the form. However, for Acrobat 6.0 and Adobe Reader 6.0, enable form caching only if the form does not include dynamic subforms.

To enable form caching, click File > Form Properties > Defaults > and select Allow Form Rendering To Be Cached on Server.

## Enabling form caching



Adobe helps people create, manage, and deliver the highest quality digital content in the world.

**Better by Adobe.™**

**Adobe Systems Incorporated**

345 Park Avenue, San Jose, CA 95110-2704 USA

[www.adobe.com](http://www.adobe.com)

Adobe, the Adobe logo, Acrobat, Adobe LiveCycle, Reader, and “Better by Adobe” are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Macintosh is a trademark of Apple Computer, Inc., registered in the United States and other countries. Linux is a registered trademark of Linus Torvalds. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Java and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark of The Open Group.

© 2005 Adobe Systems Incorporated. All rights reserved.

Printed in the USA.

9500xxxx 1/05

