Converting Acrobat® JavaScript for Use in LiveCycle™ Designer Forms

Adobe® Acrobat®
Adobe LiveCycle™ Designer
Version 7.0

Adobe® Acrobat® Adobe LiveCycle™ 7.0 Designer Converting Acrobat JavaScript for Use in LiveCycle Designer Forms for Microsoft® Windows®, UNIX®, and Linux®
Edition 1.0, June 2005

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

Contents

Preface

Purpose

This document explains how to convert JavaScript contained in Adobe® Acrobat® Professional and Acrobat Standard forms for use in Adobe LiveCycle™ Designer forms.

Acrobat forms and LiveCycle Designer forms have different scripting object models, although most of the Acrobat forms model is supported in LiveCycle Designer forms. This document describes the differences and explains how to handle the conversion to help make the conversion as easy and effective as possible.

Information about the general process of converting from Acrobat to LiveCycle Designer forms is available in the help section of LiveCycle Designer under the topic *Importing Forms from Other Applications*.

Audience

This document is intended for form developers who need to convert Acrobat forms containing JavaScript, into LiveCycle Designer forms.

This document may also be useful for form developers who are creating new XML forms and need to add Acrobat JavaScript using LiveCycle Designer.

Contents

In addition to this Preface, this document contains the following chapters:

- Chapter 1: "Converting Acrobat Forms Containing JavaScript" — Introduces conversion issues, and explains how the scripting object models differ. It also discusses how to approach the scripting conversion, and how to deal with performance issues.
- Chapter 2: "Using Acrobat JavaScript in LiveCycle Designer Forms" — Describes how Acrobat JavaScript can be used in LiveCycle Designer forms, and how to use it to complement LiveCycle Designer functionality.
- Appendix A: "Acrobat JavaScript Conversion Table " — Lists the most commonly used Acrobat JavaScript objects and their LiveCycle Designer form equivalent. Specifies the Acrobat JavaScript expressions that will or will not work in a LiveCycle Designer form, and which can optionally be converted.
- Appendix B: "Getting Started with LiveCycle Designer Forms" — Discusses important issues related to events and script levels in LiveCycle Designer forms, as well as how to handle database connections.
- Appendix C: "LiveCycle Designer UI Form Logic Example" — Gives an example of the type of logic which can be added using the LiveCycle Designer UI, without the use of scripting. The only example given is for a Text field object, and does not attempt to cover all types of objects.

Terminology

The following terms are used in this document.

Acrobat form	A form created in Acrobat, as opposed to one created in LiveCycle Designer. Acrobat forms use the Acrobat form object model; they are saved as a PDF file; and they export/import FDF, XFDF, or XML data.
FDF	Form Data Format. A file format used by Acrobat forms as a data interchange format.
FormCalc	An LiveCycle forms calculation language based on the XML form object model. It is easier to learn and use than JavaScript. FormCalc works both server-side and client-side for PDF forms. For LiveCycle Designer forms that will be published as HTML forms, it can only be used server-side because the language is not supported by Internet browsers.
form environment	The application space in which the form is running (even if it is HTML in a browser, the environment being the browser). Generally, Acrobat JavaScript (used in a LiveCycle Designer form) allows you to manipulate the form environment; LiveCycle Designer JavaScript allows you to manipulate the form contents and appearance.
LiveCycle Designer form	An XML form created by using LiveCycle Designer. LiveCycle Designer forms use the XML form object model; they can be saved as either PDF or XDP files.
LiveCycle Forms	The LiveCycle server product that deploys dynamic electronic forms over the web. (Formerly known as LiveCycle Forms Server.)
LiveCycle Designer JavaScript	A scripting language supported by LiveCycle Designer which supports the native XML form object model.
LiveCycle Designer user interface (UI) form logic	Form features and business logic that you can add by using the LiveCycle Designer UI. The result is expressed as XML code as an integral part of the form definition.
PDF	Portable document format. Used as a file format to package LiveCycle Designer forms so they can be viewed using Adobe Reader®.
script object	The LiveCycle Designer script object is an object used to store JavaScript functions that can be called from multiple locations in a form. Equivalent to Acrobat JavaScript document-level JavaScript.
SOM	Scripting object model. LiveCycle Designer forms support the LiveCycle Designer SOM and parts of the Acrobat SOM.
XDP	XML data package. A file format option you can create in LiveCycle Designer that is used to submit the form design, data, annotations, or other relevant data to Adobe LiveCycle Forms to render the form at run time. Must be used if the form will initiate server-side processing.
XML Scripting Object Model	The scripting object model used for the XML forms supported by LiveCycle server products and by LiveCycle Designer. Both LiveCycle Designer JavaScript and FormCalc scripting allow manipulation of the XML scripting object model.

Other useful documentation

The documents listed in this section are referenced in this technical note.

The following documents are available from the Adobe Solutions Network Web site.

Acrobat JavaScript

Acrobat JavaScript Scripting Guide

Acrobat JavaScript Reference

LiveCycle Designer Forms / JavaScript

Adobe XML Form Object Model 2.2 Reference

LiveCycle Designer web page for developers

http://partners.adobe.com/public/developer/livecycle/topic_designer.html

LiveCycle Designer and Adobe LiveCycle Form Server

See product Help modules for both product.

Conventions used in this document

Font	Used for	Examples
monospaced	Paths and filenames	`C:\templates\mytmpl.fm`
	Code examples set off from plain text	These are variable declarations: `AVMenu commandMenu,helpMenu;`
blue	Live links to web pages	The Adobe website URL is: http://www.adobe.com
	Live links to sections within this document	See "Converting Acrobat Forms Containing JavaScript" on page 7
italic	LiveCycle Designer Help topics	*FormCalc User Reference*
	Document titles that are not live links	*Acrobat and PDF Library API Overview*
	New terms	*User space* specifies coordinates for...

1 Converting Acrobat Forms Containing JavaScript

This chapter describes the process of converting Acrobat forms to LiveCycle Designer forms, and explains the differences between the Acrobat and LiveCycle Designer form object models. It also discusses the available options for implementing equivalent form logic in LiveCycle Designer forms.

LiveCycle Designer

LiveCycle Designer lets you build intelligent forms either by using the LiveCycle Designer UI to add form logic or by adding JavaScript or FormCalc scripting to the form. The resulting form logic can greatly enhance usability and workflow efficiency, for example, by validating data when it is entered and by allowing users to work offline.

LiveCycle Designer allows you to create all aspects of the layout, design, and form features, in one application. By using LiveCycle Designer, you can avoid the need to go back to a separate layout application and then having to recreate the form objects every time an update is made.

After you have created a form in LiveCycle Designer, it can be saved as a PDF file that contains the XML form data. You can also save it as an XDP (XML data package) file, which can be used with LiveCycle Forms; or the form can be served as either a PDF or HTML form.

You can also use LiveCycle Designer to import an Acrobat form and convert it into a LiveCycle Designer form. This document describes how to plan and execute the conversion process to help you get the desired results.

Acrobat and LiveCycle Designer scripting object models

LiveCycle Designer forms support both the Acrobat and the LiveCycle Designer scripting object models (SOM). Although most Acrobat JavaScript will work in a LiveCycle Designer form, it is important to understand why some expressions will not work, and what options you have when converting (see "Understanding the Acrobat and LiveCycle Designer form object models" on page 14).

A simple summary of the two scripting languages would be:

- **Acrobat JavaScript** can be used in a LiveCycle Designer form to handle many tasks related to the form environment; for example you can use it to add attachments, bookmarks, and annotations; search or spell check the form; create reports; or access and manipulate metadata. Acrobat JavaScript uses the Acrobat forms object model; in a LiveCycle Designer form, it cannot be used to do things like set field values, add new fields, or delete pages.

  Note: To add Acrobat JavaScript to a LiveCycle Designer form, you must use LiveCycle Designer; you cannot use Acrobat. When you view an XML form in Acrobat, all JavaScript tools will be grayed-out.

- **LiveCycle Designer JavaScript** can be used to control all aspects of form structure and data. It uses the LiveCycle native XML forms object model.

Because of the differences between the two scripting models, there is not a lot of overlap, and hence the two object models can be used in a complementary manner.

JavaScript conversion categories

One of the first steps in the conversion process is to determine how much of the Acrobat JavaScript can be used in the LiveCycle Designer form, and how much must be converted to one of the LiveCycle Designer options.

Acrobat JavaScript can be classified into one of three categories with respect to whether it will work in a LiveCycle Designer form, as shown in the following table.

	Acrobat JavaScript	**Action**
1.	Will work in LiveCycle Designer forms, and there is no LiveCycle Designer JavaScript equivalent.	Acrobat JavaScript must be enabled (see "Testing Acrobat JavaScript in LiveCycle Designer" on page 18); `doc` object code must be converted as explained in "Using Acrobat JavaScript in LiveCycle Designer Forms" on page 14, and be thoroughly tested.
2.	Will work in LiveCycle Designer forms, but conversion to LiveCycle Designer JavaScript is possible.	If the Acrobat JavaScript *can* be converted to LiveCycle Designer JavaScript, it *should* be converted (see reasons below). For Acrobat JavaScript code that will be retained, see "Using Acrobat JavaScript in LiveCycle Designer Forms" on page 14. For Acrobat JavaScript that you need to convert, you must decide which conversion option to use, as shown in "JavaScript conversion options" on page 8."
3.	Will not work in LiveCycle Designer forms, so it must be converted.	Decide which LiveCycle Designer UI form logic option to use, as shown in "JavaScript conversion options" on page 8."

The goal should be to convert all Acrobat JavaScript to one of the LiveCycle Designer form logic options shown in the table in "JavaScript conversion options" on page 8, and use Acrobat JavaScript only for the tasks that only it can handle.

There are two main reasons to convert to LiveCycle Designer JavaScript:

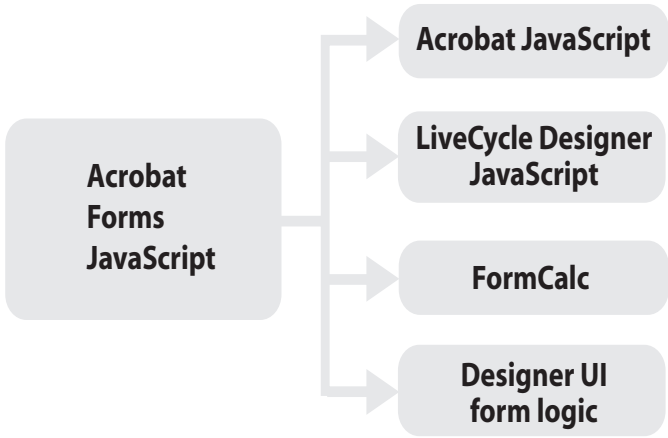
- The XML data in the form is the true representation of the form appearance and data. If you script against the Acrobat scripting model, there is a danger that the PDF and the XML representations might not stay aligned with each other.
- As the XML form object model is expanded to include additional functionality, it is possible that the new functions might replace Acrobat JavaScript functions. By converting to LiveCycle Designer logic, you will avoid having deprecated code in your forms.

The Acrobat JavaScripts that should be retained are those that deal with the form's environment and peripheral operations such as adding attachments or multimedia, performing searches or creating reports, and handling document metadata.

Appendix A: "Acrobat JavaScript Conversion Table" on page 19 can be used to determine the category for each JavaScript expression, which will help you to plan whether or not the code should be converted.

JavaScript conversion options

Five options for how the logic expressed by Acrobat JavaScript can potentially be implemented in a LiveCycle Designer form are shown below.



Note: This document focuses on only the two JavaScript options; for information on the other choices, see LiveCycle Designer Help.

The following describes the conversion options shown in illustration above.

- **Acrobat JavaScript** — Most Acrobat JavaScript that does not change the appearance or content of the form, will work in a LiveCycle Designer form. Acrobat JavaScript can handle a variety of tasks that cannot be done using LiveCycle Designer JavaScript, so you can use it to complement what you can do with LiveCycle Designer JavaScript.

  Note: See "Acrobat JavaScript not supported in LiveCycle Designer forms" on page 17 for an overview of the expressions that will not work.

- **LiveCycle Designer JavaScript** — Uses the native scripting object model for LiveCycle Designer forms. It can handle all tasks related to structure and content. Converting to LiveCycle Designer JavaScript usually requires a steeper learning curve, but the knowledge gained can be applied to all future form development.

- **LiveCycle Designer UI form logic** — The form developer can implement a variety of form functions and logic using the LiveCycle Designer UI. The resulting logic is expressed in XML as an integral part of the form, without any need for scripting. The advantage is that it can be implemented more easily and without a specialist programmer. However, if any aspect of the task must be customized, then scripting must be used. Appendix C, "LiveCycle Designer UI Form Logic Example" on page 43 illustrates examples of logic that can be added for a text field object (for example) using the LiveCycle Designer UI.

- **FormCalc scripting** — FormCalc is a simple calculation language that facilitates fast and efficient form design without requiring knowledge of traditional scripting techniques. It contains a range of built-in functions for dates and times, mathematics, finance logic, handling strings, and for web operations.

  FormCalc is especially useful for situations where many calculations need to be performed, as it is more efficient than using JavaScript. It can be used alongside LiveCycle Designer and Acrobat JavaScript. It cannot be used for client-side HTML calculations as no FormCalc engine is present.

Another option is that some functions can be handled by server-side processing, but that i usually not handled by Acrobat JavaScript in the original form, hence it is beyond the scope of this document.

Object model differences that affect scripting

In addition to the basic architectural differences between Acrobat and LiveCycle Designer form objects, the table below lists topics that are especially important to consider for forms migration because they are likely to directly affect scripting.

Topic	Acrobat forms	LiveCycle Designer forms
Events	Events can be used to trigger actions on the form, page, or field levels.	LiveCycle Designer model for events is similar to that of Acrobat, but it has more levels and offers more control. Form events can be used to trigger actions on the form, subform, page, and field levels.
LiveCycle Designer subforms	Not available in Acrobat forms. Acrobat fields are static.	LiveCycle Designer subforms allow the form to expand or contract based on the amount of data that must be handled, such as for the result of a data merging process or as a result of entries.
Form submission	In Acrobat forms, form submission is done using JavaScript.	The LiveCycle Designer UI offers more choices and control over how submissions are handled in LiveCycle Designer forms without using scripting.
Data connections	In Acrobat forms, most data connections are handled using JavaScript.	LiveCycle Designer allows you set up Web Services Definition Language (WSDL) connections, and bind that web service to form fields.
Field naming conventions	In Acrobat, fields with the same name are considered the same field and always share the same value.	In LiveCycle Designer, duplicate field names are allowed in the XML tree structure, so the fields need not have the same value for all fields that share the same name. When converting, field names need to be checked to make sure scripts behave as planned.
Security	In Acrobat, security related to signatures is handled by using JavaScript; security settings for the PDF file is set using the Acrobat UI.	You can use LiveCycle Designer to add signature fields and other security measures, but security is mainly handled by setting security options in Acrobat and by using Acrobat JavaScript.

Many aspects of the above operations can be handled without using scripting because the logic can be set using the LiveCycle Designer UI. Appendix C, "LiveCycle Designer UI Form Logic Example" on page 43 shows examples of the type of logic that can be added to a Text field object using the LiveCycle Designer UI.

Strategy and options

Some scripting conversions will be straightforward, such as when Acrobat JavaScript is the only way to do a particular task, but other JavaScript may present some choices you'll have to make as to how to best implement similar logic in the LiveCycle Designer form.

Ideally, you should convert as much Acrobat JavaScript as possible, and use Acrobat JavaScript only for those functions that only it can handle.

Other factors which might influence your choices include the available resources (that is, available programming skills), your design goals, and schedule constraints. For example, if the priority is high

enough and skilled resources are available, the best choice may be to re-architect your forms to take full advantage of LiveCycle Designer, and thus improve the forms beyond what was done using Acrobat.

It may be significant to consider the type of skills required to implement each type of form logic. The kind of skills needed to implement each type are listed below, ranging from the highest skill level down to the lowest.

Designer implementation type	**Required skills**
LIveCycle Designer JavaScript	Programming ability required; easier than C language programming.
Acrobat JavaScript	Some programming skills required. Similar to LiveCycle Designer JavaScript; the Acrobat JavaScript may be slightly easier to learn.
FormCalc scripting	Simple calculation scripting language; Some tasks can be done simply, but other tasks may require some programming skills.
LiveCycle Designer UI form logic	Programming skills not required. Easy to create and maintain.

Converting an Acrobat form

Acrobat forms are converted to LiveCycle Designer forms by using LiveCycle Designer to import the PDF form file. The process is explained in detail in the LiveCycle Designer Help topic *Importing Forms from Other Applications* > *Importing PDF Files*.

When a form is imported into LiveCycle Designer, the Acrobat JavaScript is placed in the resulting XML file without conversion, and the code is set so it will not execute (using an intentionally-invalid function call `comments()` at the beginning of all Acrobat JavaScripts). You must decide whether to use the Acrobat JavaScript as it is (after adapting it for the LiveCycle Designer form), or select one of the other conversion options shown in "JavaScript conversion options" on page 8.

➤ **To import an Acrobat form into LiveCycle Designer:**

1. Use LiveCycle Designer to import the Acrobat form by choosing File > Open and selecting the PDF form file.
2. The New Form Assistant dialog will come up and prompt you for information about Import Options (preserving editability vs. appearance), and the Return Method (how it will be distributed and how the form will be returned).
3. Resolve all issues about missing fonts, text object merging, etc. See LiveCycle Designer Help topic *Importing Forms From Other Applications*.)
4. Convert field names as appropriate to suit your scripts and the LiveCycle Designer field naming convention.
5. Convert or adapt all scripting according to the options you have chosen; for information see the following references.

Type of form logic	Useful reference
Acrobat JavaScript	For Acrobat JavaScript that you hope to use in LiveCycle Designer, see: • "Using Acrobat JavaScript in a LiveCycle Designer Form" on page 14 • "Acrobat JavaScript Conversion Table" on page 19
LiveCycle Designer JavaScript	• Adobe XML Form Object Model Reference Version 2.2 • LiveCycle Designer Help topic: *Creating Calculations and Scripts*
FormCalc scripting	LiveCycle Designer Help topics: • *Creating Calculations and Scripts* • *FormCalc User Reference*
LiveCycle Designer UI form logic	LiveCycle Designer Help topic: • *Defining Object Properties*

Note: Remember that when you have converted to a LiveCycle Designer form, the result is an XML text file. That means that you can use tools like XML parsers or Style Sheets (XSLT) to perform bulk transformations on the converted forms using a search-and-replace type of operation.

Testing and debugging

You can use the Acrobat JavaScript Debugger to test your converted scripts; it will work for both Acrobat and LiveCycle Designer JavaScript.

If you set up the Debugger in LiveCycle Designer, you will get error messages in the Debugger window, and you can embed code to print to the console, but you cannot set breakpoints, step through code, or examine variables like you can if you test the scripting using Acrobat.

To enable the Acrobat JavaScript Debugger, select Edit > Preferences, and then choose JavaScript from the Categories pane on the left. In the resulting display, check at least the following checkboxes:

- Enable Acrobat JavaScript
- Enable JavaScript Debugger after Acrobat is restarted
- Store breakpoints in PDF file
- Enable interactive console
- Show console on errors and messages
- Use Acrobat JavaScript Editor

Note: If you are using the Acrobat JavaScript Debugger in LiveCycle Designer, and need to use it in Acrobat to test and debug scripts, you first need to close the Debugger window in LiveCycle Designer.

If you are using the JavaScript Debugger in Acrobat, and need to use it in LiveCycle Designer, you need to both close Acrobat Professional and stop the Acrobat.exe process in the Windows Task Manager. See the Adobe XML Form Object Model 2.2 Reference.

For information on how to use the Acrobat JavaScript Debugger, see the *Acrobat JavaScript Scripting Guide*.

Application version issues

When developing and testing LiveCycle Designer forms, remember that if your users have older versions of Adobe Reader or Acrobat, they may experience problems:

- Users must be using Acrobat 6.0.2 (PDF language version 1.5) or higher. If not, LiveCycle Designer forms will not be recognized, and none of the LiveCycle Designer scripting will work. The user will receive a warning message if they don't have the minimum version.
- If version 6.0.2 is being used, the Adobe Interactive Forms Update SP1 plug-in must be installed (available from http://www.adobe.com/support/downloads/).
- Acrobat 7.0.5 will relax the restrictions on using Acrobat JavaScript to change form content or structure. However, even if you can change, for example, a form's appearance using Acrobat JavaScript, it is not recommended. It still will only change the PDF representation of the form, and the changes will not persist when a form is saved and reopened, or when a dynamic form is regenerated.

Performance issues

While scripting is generally a very efficient way to add business logic to a form, extensive use of scripting can lead to performance issues. If you do experience performance problems, experiment with some of the following options:

- If a form needs to do heavy-duty database look-ups or intensive validation processing, try doing those operations on the server.
- If a large amount of calculations need to be performed, consider using FormCalc scripting, which executes more efficiently than JavaScript. (However, FormCalc cannot be used with forms which will be rendered in HTML using LiveCycle Forms.)
- Use LiveCycle Designer UI form logic when possible. This may not have a significant effect on performance, but it has the advantage of making the business logic easier to maintain in the sense that small future changes may not require a skilled specialist.

2 Using Acrobat JavaScript in LiveCycle Designer Forms

This chapter explains why the Acrobat object model is restricted, and what you need to do to prepare Acrobat JavaScript for use in an XML form.

Using Acrobat JavaScript in a LiveCycle Designer Form

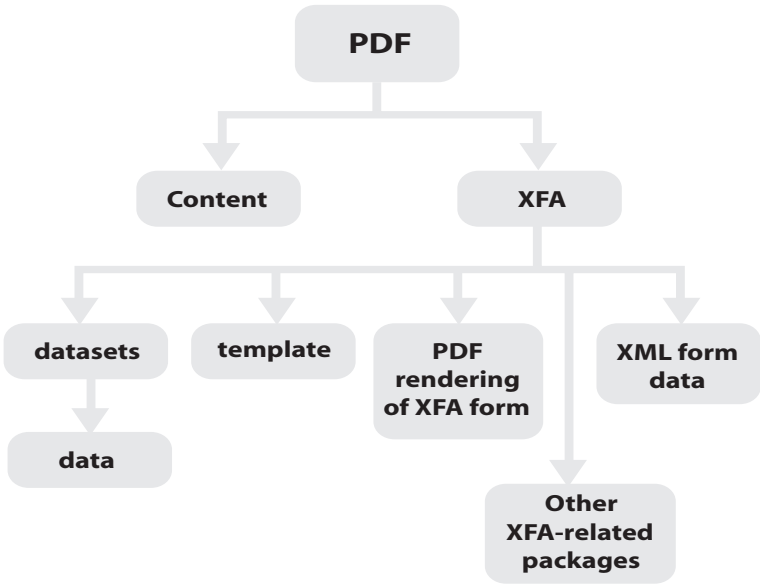
A large amount of Acrobat JavaScript can be used in a LiveCycle Designer form, as long as it doesn't try to change the content or structure of the form. See "Acrobat JavaScript not supported in LiveCycle Designer forms" on page 17 for a quick overview of the types of Acrobat JavaScript expressions which should not be used in a LiveCycle Designer form.

Acrobat JavaScript allows you to handle a variety of tasks beyond the basic form description and contents. For example, some of the functions you can perform using Acrobat JavaScript in a LiveCycle Designer form include:

- Adding annotations to a PDF document
- Utilizing Acrobat's search facility
- Adding attachments to a PDF document
- Creating PDF-based reports with the Acrobat `Report` object
- Utilizing Acrobat's rich multimedia support
- Creating bookmarks or other navigational aids
- Accessing or updating document metadata
- Supporting collaboration, review, and markup with the `Collab` object

Understanding the Acrobat and LiveCycle Designer form object models

To better understand what you can do with Acrobat JavaScript in a LiveCycle Designer form, it is important to understand that the PDF component of the LiveCycle Designer form is used only for the presentation of the form, while the structure and content are specified using the LiveCycle Designer form object model. The following illustration shows the contents of a LiveCycle Designer form packaged in a PDF file.



When a LiveCycle Designer form is saved in LiveCycle Designer as a PDF file, LiveCycle Designer creates a PDF wrapper for the form which contains the components shown in the illustration above. One of the components is a PDF representation of the LiveCycle Designer form, which is used solely for presentation purposes (that is, to display and print the form).

If you use Acrobat JavaScript to change the contents or appearance of the form, you would only be changing the PDF representation, *not* the actual form itself, which is specified by the XML data.

Hence, you should not use the Acrobat form object model to change the appearance or content of the form. For the same reason, you should try to convert as much Acrobat JavaScript as possible to LiveCycle Designer form logic or scripting.

Note: In Acrobat/LiveCycle Designer 7.0, you are not allowed to use Acrobat JavaScript to change the LiveCycle Designer form. In Acrobat/LiveCycle Designer 7.0.5, that restriction will be relaxed, but you still should not attempt such operations.

Enabling the converted Acrobat JavaScript

When LiveCycle Designer converts an Acrobat form, all Acrobat JavaScript is placed in the resulting form, but it is disabled. LiveCycle Designer places an invalid function call, `comment()` at the beginning of the script. To make the code usable for testing, simply remove the `comment()` function call from the beginning of each script.

Accessing the Acrobat form object model

Most objects from the Acrobat JavaScript Object can be accessed directly as you would use them in Adobe Acrobat, with the one exception being the `Doc` object, as explained below.

For example, associating the following JavaScript code with the click event of a form button will initiate a search in the form for the word “oranges” using Acrobat’s search object:

```
var mySearch = search;
mySearch.query("oranges");
```

Using the Acrobat Doc object

If you need to use the Acrobat Doc object in a LiveCycle Designer form, you must adapt your code to use the following technique of assigning a variable to the target of, for example, a *click* event, to access the current document:

```
var myDoc = event.target;
myDoc.importDataObject("pdfAttachment");
```

This is necessary because the `Doc` object is typically referenced in an Acrobat form using "this", as in "`this.getAnnots()`" to refer to the current document. But in LiveCycle Designer JavaScript, "`this`" refers to the current XML form object, not the current document. Hence the above convention is used.

Once the `Doc` object has been accessed, most of its methods and properties can be used in the usual way, as in the above example where the user is prompted to add an attachment to the PDF file.

Note: When testing Acrobat JavaScript in LiveCycle Designer, it's possible that you will not be able to view the results of some scripts when viewing in the PDF Preview tab window. For example, if you have a button whose attached script contains `app.execMenuItem("Open")`, it will not work in LiveCycle Designer's PDF Preview window, but will work if tested in Acrobat.

Note: If you are having trouble accessing the JavaScript console in Acrobat after using it in LiveCycle Designer, switch back to LiveCycle Designer and close the console window in that application; then the console can be accessed in Acrobat.

Referencing LiveCycle Designer JavaScript objects

If you use Acrobat JavaScript to address a LiveCycle Designer form field object, you must use the fully-qualified reference to that object, including the occurrence numbers (bracketed numbers following the object name). For example, if you have a text field in an Acrobat form named "`TextField1`", you reference it by using:

```
var f1 = this.getField("TextField1");
```

In a LiveCycle Designer form, it is necessary to use the fully qualified reference including the occurrence numbers. So, the equivalent code for a LiveCycle Designer form would look like:

```
var myDoc = event.target;
var f1 = myDoc.getField("form1[0].mySubForm[0].TextField1[0]")
```

To get the required form of the reference, you can use the following procedure where you export the form data from Acrobat as a CSV (Comma -delimited Spreadsheet file) file, which will provide you with the fully-qualified paths with the occurrence numbers, for each field in the form:

1. In Acrobat, choose: File > Create Spreadsheet from Data Files...
2. Click on the Add Files button and navigate to, and select the PDF file.
3. Click the Export button, and in the resulting dialog window, edit the suggested file name if necessary, and click the Save button.
4. A dialog window will pop-up with a button labeled "View File Now"; and click that button.
5. The resulting spreadsheet file (for example, `report.csv`), will show the fully qualified path with the occurrence numbers for all fields in the form.

There are two additional ways to get fully qualified references. In the first, you will see the occurrence numbers, but you will not be able to copy and paste them:

- In Acrobat: you can get the fully qualified reference for a Designer form object by opening the form's PDF file using Acrobat (rather than LiveCycle Designer), and choosing *View* > *Navigation Tabs* > *Fields*. If you then select the *Fields* tab in the resulting window, and expand the field icon for the current document, you will see the full reference including the occurrence numbers (but you will not be able to copy and paste them from that window).

In the second, you will see the fully qualified references, but you will not see the occurrence numbers:

- In Designer: You can also get the fully qualified reference (but without the occurrence numbers) in Designer by putting your cursor in the Script Editor and while holding down the `ctrl` key move your mouse to the desired object. The cursor will change to a "v" symbol icon. Then click on the chosen field. The absolute reference for that object is put into the Script Editor wrapped in an `xfa.resolveNode` statement. Remove the `xfa.resolveNode` to get the required reference.

Remember, that just because you can reference LiveCycle Designer fields using Acrobat JavaScript, doesn't mean that you can should use that method to change field values or other properties. However, you can use, for example, the `field.rect` property to get the coordinates of the bounding box of a field object using the following code (for this example, the result is displayed in the JavaScript Console):

```
var myDoc = event.target;
var f1 = myDoc.getField("form1[0].mySubForm[0].TextField1[0]")
console.println(f1.rect);
```

Acrobat JavaScript not supported in LiveCycle Designer forms

You should not try to use Acrobat JavaScript in LiveCycle Designer forms to change field or page contents. The table below lists a number of Acrobat JavaScript methods and properties that are not supported in LiveCycle Designer forms.

The table below is not a comprehensive list, but you should be able to tell from the property and method names that expressions that try to change page or field contents are not supported. For a more complete list of Acrobat methods and properties which can be used in a LiveCycle Designer 7.0 form, see Appendix A "Acrobat JavaScript Conversion Table" on page 19.

`app.newFDF`	`doc.getNthTemplate()`	`doc.spawnPageFromTemplate()`
`doc.templates`	`doc.gotoNamedDest()`	`doc.setPageRotation()`
`doc.addField()`	`doc.insertPages()`	`doc.setPageTabOrder()`
`doc.addLink()`	`doc.mailForm()`	`doc.templates`
`doc.addThumbnails()`	`doc.movePage()`	`field.comb`
`doc.calculate`	`doc.movePage()`	`field.deleteItemAt()`
`doc.createTemplate()`	`doc.removeField()`	`field.clearItems()`
`doc.deletePages()`	`doc.removeTemplate()`	`field.insertItemAt()`
`doc.extractPages()`	`doc.replacePages()`	`field.value`
`doc.getLinks()`	`doc.setAction()`	

Acrobat document and folder-level scripts

Acrobat document-level scripts are converted by LiveCycle Designer into LiveCycle Designer script objects. Like other Acrobat JavaScripts, they will be disabled in the converted form, so they must be checked to make sure that they will work in a LiveCycle Designer form environment; remove the `comments()` function call, and carefully test the code.

Acrobat folder-level scripts will also work with LiveCycle Designer forms in a PDF file, but you must check to make sure they are appropriate for use with LiveCycle Designer as with standard Acrobat JavaScript (see Appendix A: "Acrobat JavaScript Conversion Table" on page 19).

Testing Acrobat JavaScript in LiveCycle Designer

When you import an Acrobat form into LiveCycle Designer, all Acrobat JavaScript will be disabled by the (intentionally invalid) `comments()` function call. To use the JavaScript, first make sure that the Acrobat expressions are supported in LiveCycle Designer (see Appendix A: "Acrobat JavaScript Conversion Table" on page 19). If they are, then you must remove the `comment()` code and test the code. See also "Testing and debugging" on page 12.

Troubleshooting

If you're having trouble getting Acrobat JavaScript expressions to work in a LiveCycle Designer form (beyond the special treatment for the `Doc` object), check the following:

- Check if the Acrobat JavaScript expression is allowed in LiveCycle Designer forms; see Appendix A: "Acrobat JavaScript Conversion Table" on page 19 for more details.
- Check version compatibility issues: Acrobat JavaScript that attempted to change the content or structure of a form — worked in version 6.X, was not allowed in 7.0; but may be allowed again in 7.0.5. However, just because those operations are allowed does not mean that they are recommended; see "Understanding the Acrobat and LiveCycle Designer form object models" on page 14.
- Check field object references to make sure the scope is correct. (Fully qualified names for references outside of the current container, etc.)
- Check security restrictions in the *Acrobat JavaScript Reference*; some Acrobat objects can only execute in batch mode, as folder-level JavaScript, or from the Acrobat JavaScript console.
- Check to make sure that all references to LiveCycle Designer field object names are fully qualified references, including the occurrence numbers.
- If you have a JavaScript that seems not to work while testing in LiveCycle Designer, see if it works when viewing it in Acrobat Professional.

A Acrobat JavaScript Conversion Table

The following table lists the most commonly used Acrobat JavaScript expressions. The second and third columns show whether the Acrobat JavaScript expression can be used in a LiveCycle Designer form, and whether there is a LiveCycle Designer equivalent (Y = Yes, N = No).

Some Acrobat objects are not listed. For example, multimedia objects are not listed because they should all work in a LiveCycle Designer form. Some special-purpose objects are not listed because they are rarely used for forms.

In cases where no equivalent LiveCycle Designer JavaScript is listed, that does not mean that it is impossible to do with LiveCycle Designer scripting, it only means that there is no single equivalent expression.

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	LiveCycle Designer JavaScript	Description
Annot Object Properties/Methods				
< All properties and methods >	Y	N	No equivalent	All Acrobat `Annot` properties and methods can be used in LiveCycle Designer forms. **Note:** Only Static forms support the annotation layer; Dynamic forms do not.
App Object Properties				
`app.calculate`	N	N	`objectname.execCalculate()`	The closest equivalent, the LiveCycle Designer JavaScript method `objectname.execCalculate()` gives you control over what can be calculated.
`app.language`	Y	Y	`xfa.host.language`	
`app.monitors`	Y	N	—	
`app.platform`	Y	Y	`xfa.host.platform`	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	LiveCycle Designer JavaScript	Description
app.plugins	Y	N	—	
app.toolbar	Y	N	—	
app.viewerType	Y	Y	xfa.host.appType	
app.viewerVariation	Y	Y	xfa.host.variation	
app.viewerVersion	Y	Y	xfa.host.version	
App Object Methods				
app.addMenuItem()	Y	N	—	
app.addSubMenu()	Y	N	—	
app.addToolButton()	Y	N	—	
app.alert()	Y	Y	xfa.host.messageBox()	
app.beep()	Y	Y	xfa.host.beep()	
app.browseForDoc()	Y	N	—	
app.clearInterval()	Y	N	—	
app.clearTimeOut()	Y	N	—	
app.execDialog()	Y	N	—	
app.execMenuItem()	Y	N	—	Executes the specified menu item. Can be used in LiveCycle Designer for menu items such as Close, Find, Save, Save As, Open, Print, etc.
app.getNthPluginName()	Y	N	—	
app.getPath()	Y	N	—	
app.goBack()	Y	N	—	
app.goForward()	Y	N	—	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`app.hideMenuItem()`	Y *	N	—	
`app.hideToolbarButton()`	Y	N	—	
`app.launchURL()`	Y	Y	`xfa.host.gotoURL`	
`app.listMenuItems()`	Y	N	—	
`app.listToolbarButtons()`	Y	N	—	
`app.mailGetAddrs()`	Y	N	—	
`app.mailMsg()`	Y	N	—	
`app.newDoc()`	Y	N	—	For Acrobat forms, this can only be executed during batch; console; or menu events.
`app.newFDF()`	N	N	—	
`app.openDoc()`	Y	N	—	
`app.openFDF()`	N	N	—	
`app.popUpMenuEx()`	Y	N	—	
`app.popUpMenu()`	Y	N	—	
`app.removeToolButton()`	Y	N	—	
`app.response()`	Y	Y	`xfa.host.response` ("Question", "Title", "Default Value")	
`app.setInterval()`	Y	N	—	
`app.setTimeOut()`	Y	N	—	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`app.trustedFunction()`	Y	N	—	Acrobat JavaScript: marks a function as "trusted," meaning that it is capable of increasing the current privilege level for their stack frame. Only available during batch, console, and application initialization.
`app.trustPropagatorFunction()`	Y	N	—	
Bookmark Object Properties/Methods				
< All properties and methods >	Y	N	No equivalent	
Doc Object Properties				
`doc.author`	Y	N	—	
`doc.baseURL`	Y	N	—	
`doc.bookmarkRoot`	Y	N	—	
`doc.calculate`	N	N	—	
`doc.dataObjects`	Y	N	—	
`doc.delay`	N	N	—	

`doc.dirty`	Y	N	—	You can save a copy of the form at initialization:
				`var sOrigXML = xfa.data.saveXML;`
				and test if anything has changed:
				`if(sOrigXML != xfa.data.asveXML){...}`

`doc.disclosed`	Y	N	—	
`doc.documentFileName`	Y	N	—	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`doc.dynamicXFAForm`	Y	N	—	
`doc.external`	Y	N	—	
`doc.filesize`	Y	N	—	
`doc.hidden`	Y	N	—	
`doc.icons`	Y	N	—	
`doc.keywords`	Y	N	—	
`doc.layout`	Y	N	—	
`doc.media`	Y	N	—	
`doc.metadata`	Y	Y	`xfa.form1.desc`	
`doc.modDate`	Y	N	—	
`doc.mouseX` `doc.mouseY`	Y Y	N N	—	
`doc.noautocomplete`	Y	N	—	A boolean that specifies whether to *auto complete* the form (called *prepopulating* in LiveCycle Designer forms).
`doc.nocache`	Y	N	—	
`doc.numFields`	Y	Y	`var allFormObjects = form1.Page1.nodes; for(i=0; i< allFormObjects.length;i++){ app.alert(allFormObjects.item(i).rawValue); }`	
`doc.numPages`	Y	Y	`xfa.host.numPages`	
			`xfa.layout.absPageCount()` `xfa.layout.pageCount()`	
			`xfa.resolveNode("FieldName[*]").length`	
`doc.pageNum`	Y	Y	`xfa.host.currentPage`	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`doc.pageNum--`	Y	Y	`xfa.host.currentPage--` or `xfa.host.pageUp()`	
`doc.pageNum++`	Y	Y	`xfa.host.currentPage++` or `xfa.host.pageDown()`	
`doc.path`	Y	N	—	
`doc.securityHandler`	Y	N	—	
`doc.templates`	N	N	—	No equivalent. Use Subforms in LiveCycle Designer forms. You can add, remove, move and set subform instances.
`doc.title`	Y	N	`xfa.host.title`	
Doc Object Methods				
`doc.addAnnot()`	Y	N	—	
`doc.addField()`	N	N	—	Must use dynamic subforms in LiveCycle Designer along with hiding/unhiding the field. For example, `this.presence = "visable"` (or `"invisible"`).
`doc.addIcon()`	Y	N	—	
`doc.addLink()`	N	N	—	No exact equivalent; example shown can be used for a link to a web page but *not* to other pages in the form. For example: `xfa.host.gotoURL` ("http://www.adobe.com")
`doc.addRecipientListCryptFilter()`	Y	N	—	
`doc.addScript()`	Y	N	—	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`doc.addThumbnails()`	N	N	—	
`doc.addWatermarkFromFile()`	Y	N	—	
`doc.addWatermarkFromText()`	Y	N	—	
`doc.addWeblinks()`	Y	N	—	
`doc.appRightsSign()`	Y	N	—	
`doc.appRightsValidate()`	Y	N	—	
`doc.bringToFront()`	Y	N	—	
`doc.calculateNow()`	N	Y	`xfa.form.recalculate(1);`	Acrobat JavaScript: Forces computation of all fields in the document, but it's not allowed in LiveCycle Designer forms. The LiveCycle Designer JavaScript `recalculate` method forces a specific set of scripts located on calculate events to execute. The boolean value indicates whether True (default) — all calculation scripts are re-executed; or False — only pending calculation scripts should be executed.
			`xfa.form.calculate()`	The LiveCycle Designer JavaScript `calculate` object controls the calculation of a field's value as to whether the user can override the calculated value (that is, not the equivalent of the Acrobat JavaScript, but included here for clarify the difference).
`doc.closeDoc()`	Y	?	—	
`doc.createDataObject()`	Y	N	—	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`doc.createTemplate()`	N	N	—	LiveCycle Designer templates don't have the same meaning as for Acrobat templates. The equivalent is to use LiveCycle Designer subforms.
`doc.deletePages()`	N	N	—	
`doc.embedDocAsDataObject()`	Y	N	—	
`doc.encryptForRecipients()`	Y	N	—	
`doc.encryptUsingPolicy()`	Y	N	—	
`doc.exportAsText()`	Y	N	—	Security restrictions limit its use to the JavaScript console or batch
`doc.exportAsFDF()`	N	Y	`xfa.host.exportData()`	The LiveCycle Designer equivalent exports an XML or XDP file, rather than an FDF file.
`doc.exportAsXFDF()`	N	Y	`xfa.host.exportData("filename.xml", 0);`	
`doc.exportDataObject()`	Y	N	—	
`doc.exportXFAData()`	N	Y	`xfa.host.exportData()`	
`doc.extractPages()`	N	N	—	
`doc.flattenPages()`	Y	N	—	
`doc.getAnnot()`	Y	N	—	
`doc.getAnnots()`	Y	N	—	
`doc.getDataObjectContents()`	Y	N	—	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`doc.getField("FName")`	Y	Y	`xfa.resolveNode("FName")`	The Acrobat code maps a variable name to the named field object; the Designer JavaScript `resolveNode` accesses the specified tree object. You should be able to use doc.getField() to access field properties in 7.0, but not to set values or change properties.
`doc.getLegalWarnings()`	Y	N	—	
`doc.getLinks()`	N	N	—	
`doc.getNthFieldName()`	Y	Y	For LiveCycle Designer forms; you can loop through all pages using `xfa.host.numPages;` get number of field nodes using `xfa.layout.pageContent()`, and count fields with a class name of "textEdit".	Can use the Acrobat JavaScript: `event.target.getNthFieldName(2)` to get the field name.
`doc.getNthTemplate()`	N	N	—	
`doc.getOCGs()`	Y	N	—	
`doc.getOCGOrder()`	Y	N	—	
`doc.getPageBox()`	Y	N	—	
`doc.getPageLabel()`	Y	N	—	
`doc.getPageNthWord()`	Y	N	—	
`doc.getPageNthWordQuads()`	Y	N	—	
`doc.getPageNumWords()`	Y	N	—	
`doc.getPageRotation()`	Y	N	—	
`doc.getPrintParams()`	Y	N	—	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`doc.getTemplate()`	N	N	—	
`doc.getURL()`	Y	Y	`xfa.host.gotoURL("http://www.adobe.com")`	
`doc.gotoNamedDest()`	N	N	—	
`doc.importAnFDF()`	N	N	—	
`doc.importAnXFDF()`	Y	N	—	
`doc.importDataObject()`	Y	N	—	
`doc.importIcon()`	Y	N	—	
`doc.importTextData()`	Y	N	—	
`doc.importXFAData()`	N	Y	`xfa.host.importData("filename.xdp");`	
`doc.insertPages()`	N	N	—	
`doc.mailDoc()`	Y	N	—	
`doc.mailForm()`	N	N	—	
`doc.movePage()`	N	N	—	
`doc.newPage()`	N	N	—	
`doc.openDataObject()`	Y	N	—	
`doc.print()`	Y	Y	`xfa.host.print();`	
`doc.removeDataObject()`	Y	N	—	
`doc.removeField()`	N	N	—	
`doc.removeIcon()`	Y	N	—	
`doc.removeLinks()`	N	N	—	
`doc.removeScript()`	Y	N	—	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`doc.removeTemplate()`	N	N	—	
`doc.removeThumbnails()`	N	N	—	
`doc.removeWeblinks()`	Y	N	—	
`doc.replacePages()`	N	N	—	
`doc.resetForm()`	N	Y	`xfa.host.resetData()`	
			`xfa.event.reset()`	

`doc.saveAs()`	Y	N	—	Saves the file. With no arguments, it will bring up a dialog box. In LiveCycle Designer, the file must be saved at the application level, so you must use Acrobat JavaScript:
				`app.executeMenuItem("SaveAs");`
				or:
				`var myDoc = event.target;`
				`myDoc.saveAs();`

`doc.spawnPageFromTemplate()`	N	N	—	
`doc.setAction()`	N	N	—	
`doc.setPageLabel()`	Y	N	—	
`doc.setPageRotation()`	N	N	—	
`doc.setPageTabOrder()`	N	N	—	The tab order is set using the LiveCycle Designer UI.
`doc.setScript()`	N	N	—	
`doc.submitForm()`	Y	N	Must use the submit form buttons created in the LiveCycle Designer UI.	
event Object Properties				

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	LiveCycle Designer JavaScript	Description
`event.change()`	Y	Y	`XFA.eventPseudoModel.change`	
`event.targetName`	Y	Y	`XFA.eventPseudoModel.target`	
Field Object Property				
`field.comb`	N	Y	—	
`field.charLimit`	N	N	—	For static LiveCycle Designer forms, character limit can be set in the LiveCycle Designer UI; in dynamic forms fields can expand to accommodate all data.
`field.display = display.noView`	N	Y	`this.presence = "visible"` in the prePrint event, and `this.presence = "invisible"` in the postPrint event.	This property can be set using the LiveCycle Designer UI; also can set other presence options. JavaScript can be used to set the form field so it is invisible on screen, but visible when printed.
`field.display = display.noPrint`	N	Y	`this.presence = "invisible"` in the prePrint event, and `this.presence = "visible"` in the postPrint event.	
`field.defaultValue`	N	N *	—	Default values are set using the LiveCycle Designer UI.
`field.exportValues`	N	N *	—	Export values are set using the LiveCycle Designer UI.
`field.fillColor`	N	Y	`xfa.Form2.NumericField3.fillColor = "200, 150, 250";`	
`field.hidden`	N	Y	`this.presence = "invisible"` `this.presence = "visible"`	
`field.multiline`	N	Y	—	Property is set in the LiveCycle Designer UI.
`field.password`	N	N	—	Controlled through the LiveCycle Designer UI settings; there is a specific password field object.

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`field.page`	N		—	Not applicable for LiveCycle Designer forms.
`field.print`	N	Y	`TextField1.relevant "-print";`	Code to left specifies that a text field is visible but will not print.
`field.radiosInUnison`	N	N	—	In LiveCycle Designer, all radio buttons in a single *exclusion group* are mutually exclusive, by definition; groups are set in the LiveCycle Designer UI.

`field.rect`	Y	N	You can get the height and width of a LiveCycle Designer form field:
			`xfa.layout.h(textField1);`
			`xfa.layout.w(textField1);`
			Or you can get the x and y coordinates of the form object:
			`xfa.layout.x(textField1);`
			`xfa.layout.y(textField1);`	

`field.required`	N	N	—	In LiveCycle Designer forms, this property is set by using the UI.
`field.textColor`	N	Y	`TextField1.fontColor`	
`field.textSize`	N	Y	`TextField1.font.size = 14;`	
`field.textFont`	N	Y	`TextField1.font.typeface = "Viva Regular";`	
`field.value`	N	Y	`<fieldName>.rawValue`	**Note:** Designer form fields have a value property but it is not the same as the Acrobat value property.
Field Object Methods				
`field.clearItems()`	N	Y ?	`DropDownList1.clearItems();`	XFA: a drop-down list is similar in function (but not exactly the same).
`field.deleteItemAt()`	N	?	—	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
`field.getItemAt()`	N	N	—	
`field.insertItemAt()`	N	N	`DropDownList1.addItem(.....).value`	See Adobe XML Form Object Model 2.2 Reference for arguments and sample code in the section "JavaScript Examples".
`field.isBoxChecked()`	N	?	`if(CheckBox1.rawValue == 1)....`	
`field.isDefaultChecked()n`	N			
`field.setAction()`	N	N	—	Not applicable for LiveCycle Designer forms.
`field.setFocus()`	Y (?)	Y	`xfa.host.setFocus("`<field reference>`").`	
`field.setItems()`	N	N	—	
`field.setLock()`	Y	N	—	
`field.signatureGetModifications()`	Y	N	—	
`field.signatureGetSeedValue()`	Y	N	—	
`field.signatureInfo()`	Y	N	—	
`field.signatureSetSeedValue()`	Y	N	—	
`field.signatureSign()`	Y	N	—	
`field.signatureValidate()`	Y	N	—	
Search Object Methods				
`search.query("`*<your text>*`");`	Y	N	No equivalent	

Acrobat JavaScript	Works in LiveCycle Designer	LiveCycle Designer equivalent	**LiveCycle Designer JavaScript**	**Description**
SOAP Object Methods				
`SOAP.connect(cURL)` < All properties and methods >	Y	N	No equivalent	

Setting document actions

When designing forms with Acrobat, the form developer is able to configure the form so that JavaScript is executed on various document actions, via the Document Actions' dialog box. To achieve the same results with LiveCycle Designer, select the root element of the form from the Hierarchy panel, then select the appropriate event from the event drop down list and add your script as necessary.

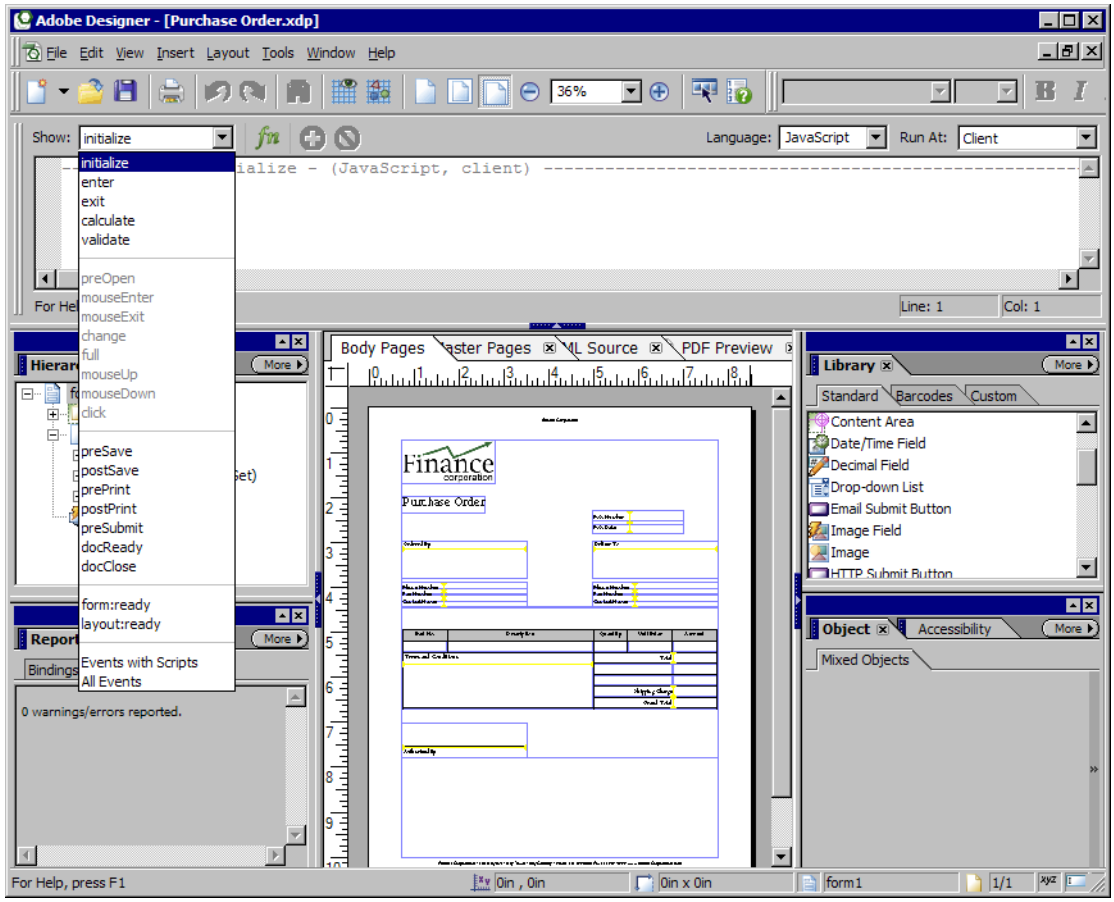
Note: The following table lists only the most commonly used Acrobat events.

Acrobat Event	LiveCycle Designer Form Event	Description
Document		
Open		
Will Close	docClose	Executes prior to the close of the document.
Will Save	preSave	Executes prior to the document being saved.
Did Save	postSave	Executes after the document has been saved.
Will Print	prePrint	Executes before the document is printed.
Did Print	postPrint	Executes after the document has been printed
	docReady	Executes prior to the rendering of the document, but after data binding of the data takes place.
Field		
	initialize	
Focus	Enter; mouseUP;	
Blur	mouseEnter	
Validate	validate	
Calculate	calculate	
Format	—v	
MouseEnter	mouseEnter	
MouseExit	mouseExit	
MouseDown	mouseDown; Click	
MouseUp	mouseUp	
KeyStroke/Change	Change/Full	

Acrobat Event	LiveCycle Designer Form Event	Description
Page		
PageOpen		
Page Close		

Creating form-level events

Form-Level Events



Document-level scripts

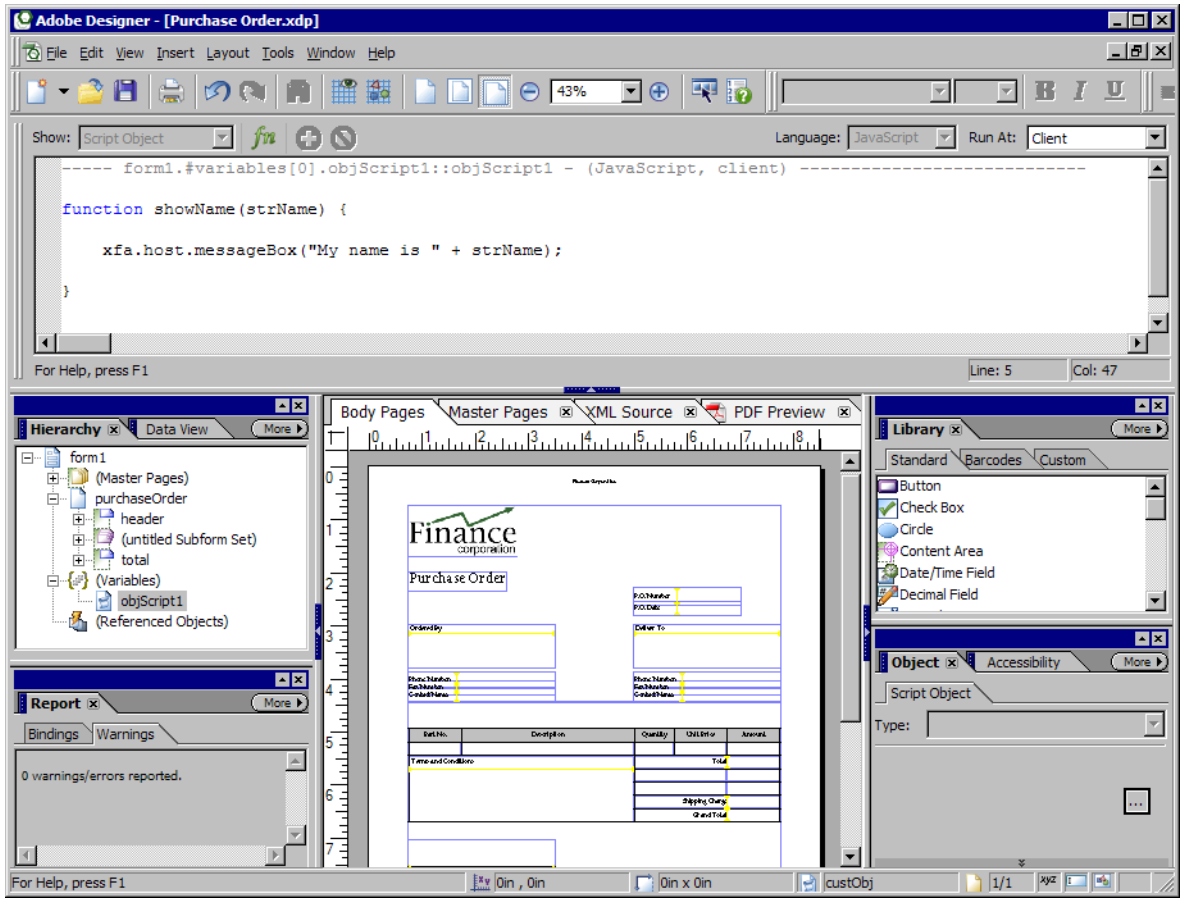
One of the major reasons for utilizing document-level JavaScript is to promote reuse of code throughout a form. Rather than having a particular piece of code stored behind several form controls (i.e. form buttons, drop down lists, etc.), the code can be placed in a global location that can be called by any control throughout the form.

In Acrobat, this was best achieved via the Document JavaScripts dialog. LiveCycle Designer introduces the concept of a *script object* that can be used to store common, re-usable document-level scripts.

The following illustration shows a script object named `objScript1` existing at the form level, which can be accessed from anywhere throughout the form. In this script object is a function named `showName`, which accepts a string parameter. This function can be called using the following:

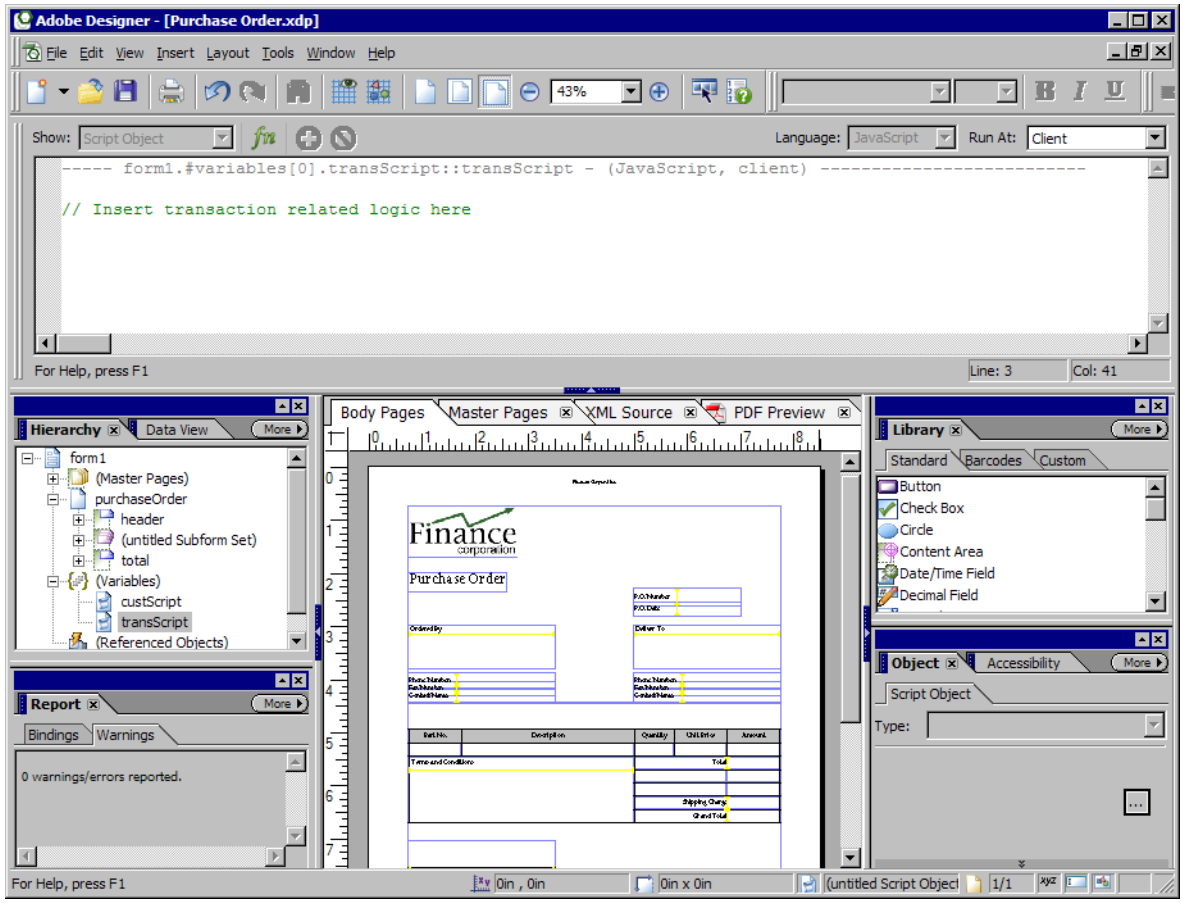
```
objScript1.showName("Enter Name");
```

Creating A Script Object



Script objects are a great way to organize and maintain your JavaScript form logic. FormCalc is not supported in the script object. An example of a script object is where a form contains data relating to a customer and transactions made by that customer. Rather than maintaining the one long mix of logic, the form developer can use separate script objects to organize the logic; one for the customer related logic, and another script object for the transaction related logic.

Creating Multiple Script Objects for Clarity

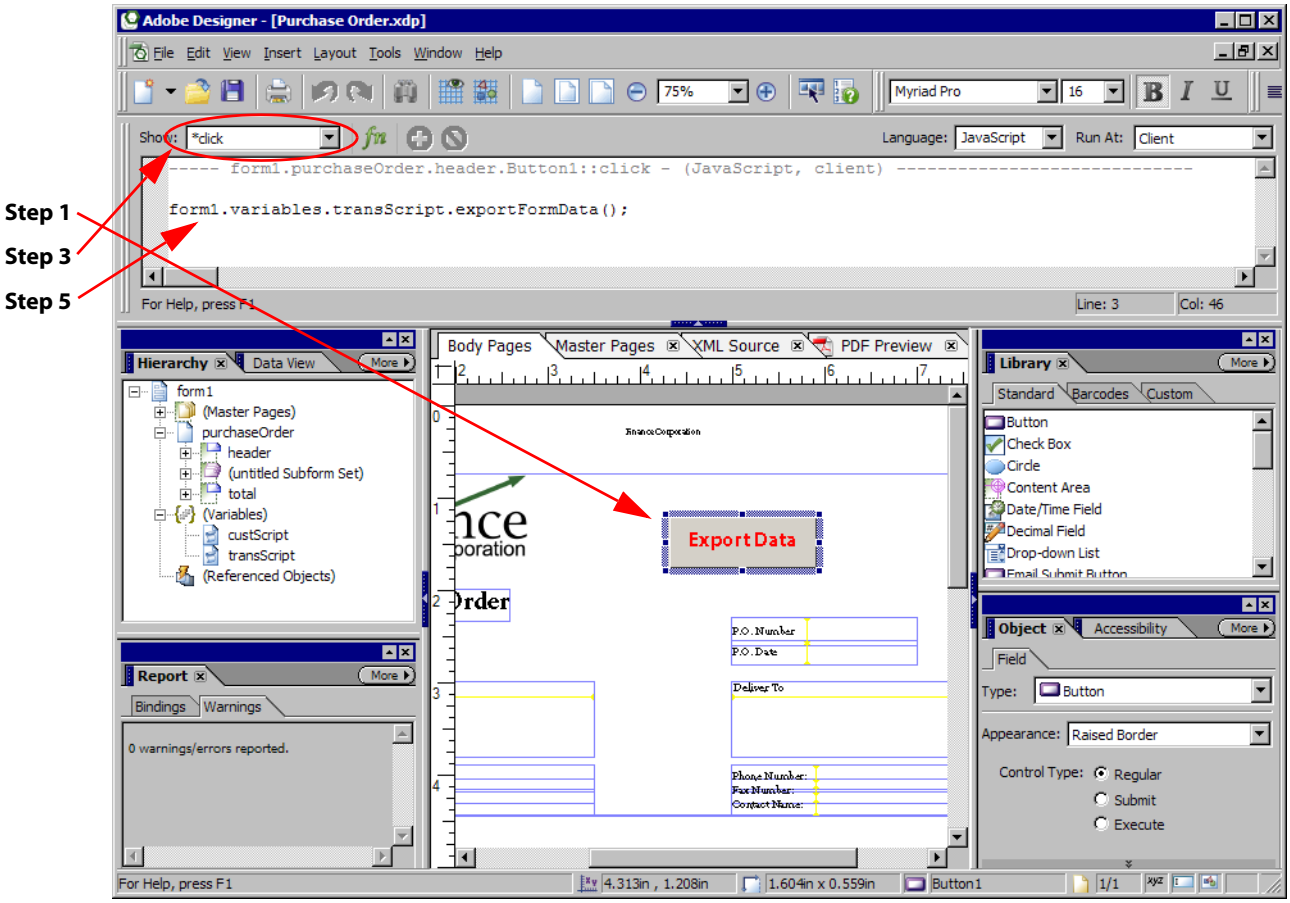


Field-level scripts

While it is useful to take advantage of script objects for document-level scripts, situations still exist where it makes sense for the form developer to associate script with just a particular form control. To add field-level script in LiveCycle Designer, follow these steps:

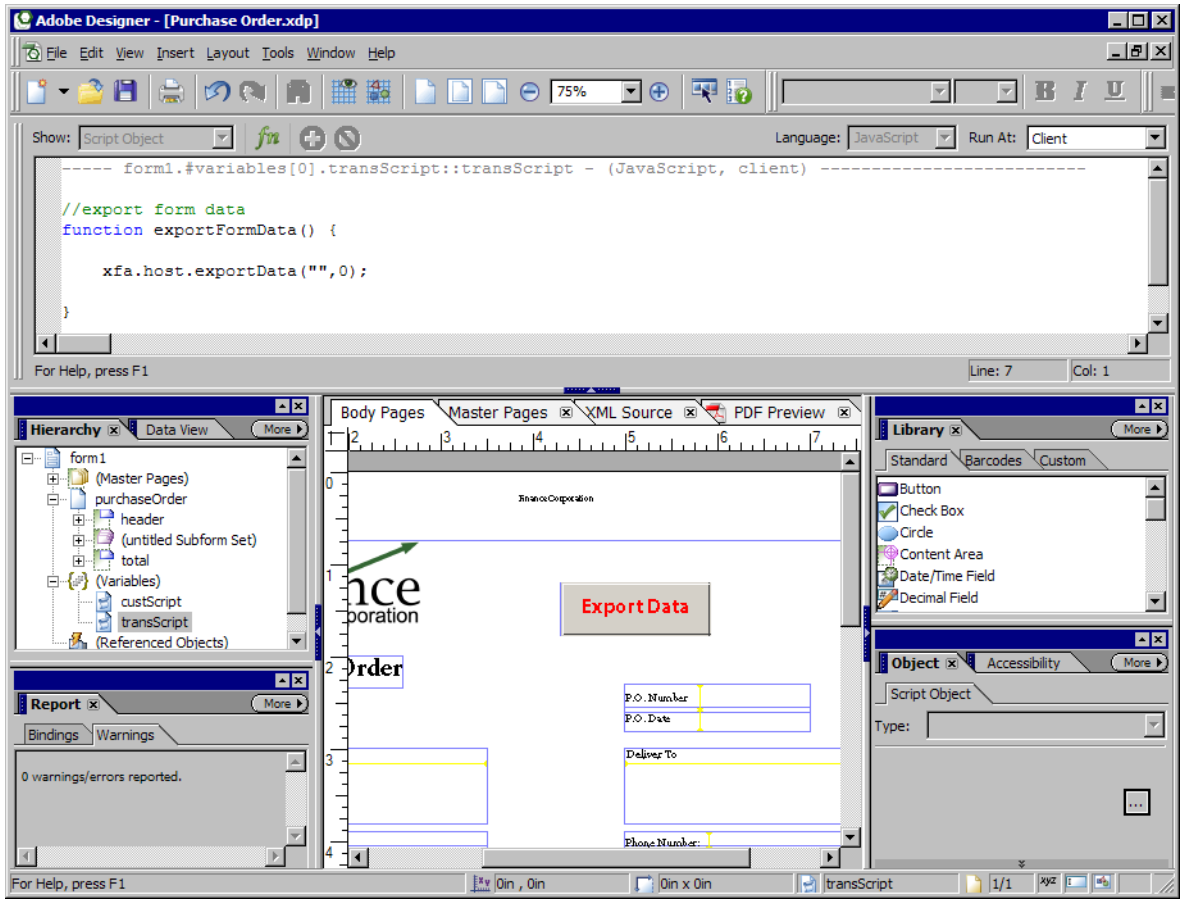
1. Select the field you want to associate the script with.
2. Expand the script editor.
3. Select the event you wish to associate the script with from the drop-down menu.
4. Select to use either JavaScript or FormCalc.
5. Select to execute on either Client, Server, or Client & Server. For most Acrobat applications, select Client. (For more information see LiveCycle Designer Help — *Where calculations and scripts are run*.)
6. Enter the script in the editor.

Creating Field-Level Scripts

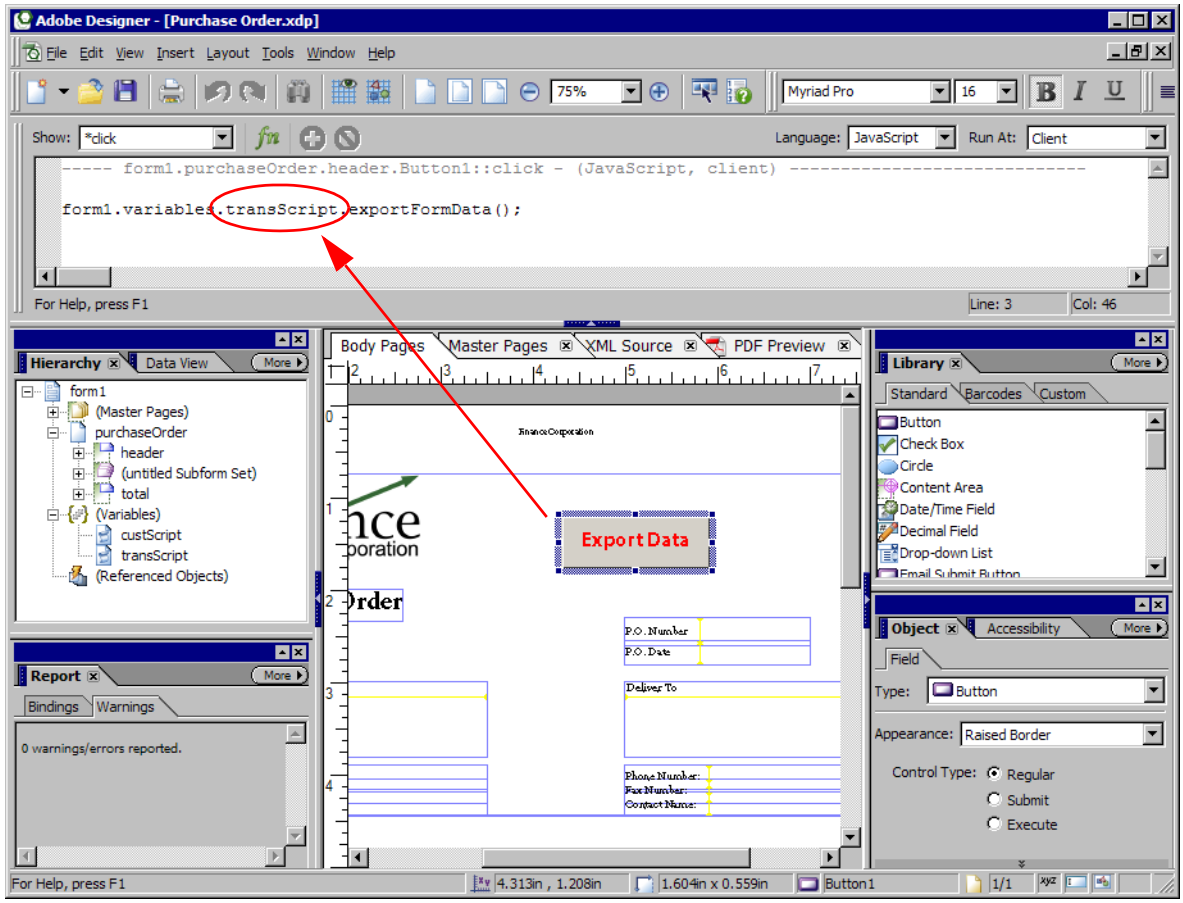


Field level scripts are also used to call functions stored in a script object. That is, if you have a function stored in a script object you would like to call on the click of a form button, you would associate a field level script to this button to call the function in the script object. An example of this can be seen in the below screenshots.

Function Stored in transScript Script Object



Associating click event with JavaScript function in Script Object "transScript"



Interfacing with a database

Having the ability to bind a database to form fields in LiveCycle Designer results in a lot less scripting being required on the form developer's side. When a database is bound to the form fields, LiveCycle Designer does a lot of this work for you. To illustrate interfacing with an ODBC database using LiveCycle Designer and iterating through records, follow these few steps.

1. Create a connection to a database as per the LiveCycle Designer Help section *Creating a data connection to a database*.

2. After the connection is created, it will be visible in the Data View palette. Drag this data connection icon onto the form. This will create the form fields, binding them to the database schema of your data connection.

3. Create two new form buttons of type Regular. Name one Next Record and the other Previous Record. These buttons will be used to iterate through each record from your datasource.

4. Select the Next Record form button, and in the Script Editor under the **click** event for this button enter `sourceSet.DataConnectionName.next();`. Select **JavaScript** as the scripting language.This will move to the next record in the record set. For example, if your data source name is 'Customers', then it should read:

   ```
   sourceSet.Customers.next();
   ```

5. Do the same for the Previous Record form button, however the script should be `sourceSet.DataConnectionName.previous();`. This will move to the previous record in the record set.

Additionally, to add a new record to the database via your form, create two more form buttons, and label them Add New Record and Save. Select the Add New Record form button, and in the Script Editor under the Click event for this button enter `sourceSet.DataConnectionName.addNew();`. Select JavaScript as the scripting language. When this button is clicked, it will create a new record to the record set. You can then fill in the form fields with the new data, and then click the Save form button, which should have the script `sourceSet.DataConnectionName.Update();` associated with the click event.

The following table illustrates the methods available for a form developer to interface with a data connection in LiveCycle Designer.

Method	Description
`addNew()`	Appends a new record to the record set.
`cancel()`	Cancels any changes made to the current or new row of a record set object, or the fields collection of a record object, before calling the update method.
`cancelBatch();`	Cancels a pending batch update.
`close();`	Close the connection to the data source.
`delete();`	Delete the current record from the record set.
`first();`	Move to the first record in the record set, and populate the LiveCycle Designer Data DOM with the record data.
`hasDataChanged();`	This is a pre-commit test. It compares the current data record data with the current data source's record data. If any fields/column information are different then this method will return True. Any of the move methods (previous, next, first, last) will perform an implicit update. This method is a specific test of the current active record.
`isBOF();`	Returns True if at the BOF of the recordSet. The bofAction property has to be set to '**`stayBOF`**'.
`isEOF();`	Returns True if at the EOF of the recordSet. The eofAction property has to be set to '**`stayBOF`**'.
`last();`	Move to the last record in the record set, and populate the LiveCycle Designer Data DOM with the record data.
`next();`	Move to the next record in the record set, and populate the LiveCycle Designer Data DOM with the record data.
`open();`	Connect to the data source and populate the LiveCycle Designer Data DOM with the results of the current record.
`previous();`	Move to the previous record in the record set, and populate the LiveCycle Designer Data DOM with the record data.

Method	Description
`requery();`	Refreshes the current bindings which updates the data by re-executing the query on which the object is based. Calling this method is equivalent to calling the close and open methods in succession.
`resync();`	Forces a refresh of the underlying record set or connection if necessary.
`update();`	Update the current record in the record set.
`updateBatch();`	Writes all pending batch updates to the data source.

C LiveCycle Designer UI Form Logic Example

This appendix lists examples of form logic which can be set for a Text field added to a form using settings accessed through the LiveCycle Designer UI. The advantage of setting logic in this way is that it is much easier than using JavaScript. It t also means that long-term maintenance and updating will be easier because it will not require skilled programmers.

Logic	Comments
Set the Locale	Sets the locale for the language. You can select a specific language, a default locale, or use the user's system locale.
Set a message to encourage user to enter data	In Acrobat, you can only display a message as an event action that executes a custom script. In LiveCycle Designer, you can specify a custom message which will be displayed when the user tabs out of a field without entering a value, or when the user tries to submit the form without entering required data.
Set a validation pattern & error message	In Acrobat, you can only specify an acceptable range of numbers which are allowed, or run a custom JavaScript to do validation. In LiveCycle Designer, you can set a validation pattern to which the data must conform, and you can specify an error message to be displayed when the data doesn't match that pattern.
Define run-time value	Allows you to set a variety of run-time values such as the current page number; the total number of pages; or the Viewer language, name, or version number.
Define data binding pattern	Sets the pattern for storing and retrieving bound data.
Override message	Sets a custom message to inform users that they are changing the value of a calculated field.
Set the object Type	Enables run-time calculations and prompts the user when interaction is required. You can write a script or select a run-time property that will perform a special function to control the type of field it is and how it will be used.

For information on what other logic you can add using the LiveCycle Designer UI, see LiveCycle Designer Help topic *Defining Object Properties*.