Creating Multiple Levels of Repeating Headers..............................................................48
Notes.................................................................................................................................50

Preserving decimal digits when calculating .................................................................51

Fitting a Large Graphic .................................................................................................54
  Background: ....................................................................................................................54
  Solution: .........................................................................................................................55

Creating a URL based read-only graphic .................................................................57
  Solution: .........................................................................................................................57

Setting Overflow Leaders/Trailers ..............................................................................59
  Creating Overflow leaders/trailers .............................................................................60
  Important Notes: ...........................................................................................................61

Printing Duplex Forms ................................................................................................62
  Creating a duplex form ...............................................................................................62

Scripting ........................................................................................................................64
  Scripting Overview .......................................................................................................64
  Events ............................................................................................................................67
  Creating Calculations and Scripts.............................................................................87
  Scripting with Form Calc and JavaScript ..................................................................95
  Variables .......................................................................................................................105
  Creating and Reusing JavaScript Functions ..............................................................108
  Using Script Fragments ..............................................................................................110
  Debugging Calculations and Scripts .......................................................................111

Hyphenating Text .......................................................................................................116
  Hyphenating Text.......................................................................................................116

Page Breaks ..................................................................................................................119
  Page Breaks Controls ...............................................................................................119

Typography ..................................................................................................................121
  Typography ................................................................................................................121

Tabbing Order ...............................................................................................................123
  Tabbing Order ............................................................................................................123

Text RTL .........................................................................................................................130
  Using text RTL ..........................................................................................................130

Barcode .........................................................................................................................134
  Barcode .......................................................................................................................134

Strict Scoping ...............................................................................................................137
  Typography ................................................................................................................137

Making Secure Form ....................................................................................................148
  Secure Form ...............................................................................................................148

Validating Data ............................................................................................................154
  Validating Data..........................................................................................................154

Form performance .......................................................................................................168
  General design considerations for printing.........................................................168
  Optimizing performance for images ......................................................................168
Optimizing performance for discrete option choices ................................................................. 169
Optimizing performance for subforms .................................................................................... 169
Optimizing performance for text fields .................................................................................. 170
Data binding options for performance .................................................................................. 170
Evaluating data structure for performance .......................................................................... 171
Tips for reducing the number of objects ............................................................................. 173
Tips for reducing the complexity of objects ........................................................................ 173
Overview

This document outlines the best practices to follow when creating forms using the Adobe® LiveCycle™ forms technology.

Software components

The LiveCycle forms technology includes the following software components.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adobe LiveCycle Designer</td>
<td>A graphical user interface (GUI) application for creating form designs in Adobe XML Data Package (XDP) format.</td>
</tr>
<tr>
<td>Application or database</td>
<td>An application or database that exposes data in XML format.</td>
</tr>
<tr>
<td>Adobe document services</td>
<td>An intelligent data-capturing and data-merging solution that combines a form design, XML data, and configuration settings. It then renders the form as an Adobe PDF, an Adobe PostScript®, a Printer Control Language (PCL), or a Zebra Programming Language (ZPL) file.</td>
</tr>
</tbody>
</table>

Steps to render forms

To render forms with the LiveCycle forms technology, follow these steps:

1. **Define the XML data:** Define the data structure that the application or database will expose and that the form design will use.

2. **Create the form design:** In LiveCycle Designer, drag fields to create the form design. The form design is stored on a server and used in the rendering process.

3. **Render the form:** Submit the form design (XDP), data (XML), and configuration settings (XCI) to Adobe document services. Adobe document services build an output independent version of the form, and then convert this form layout to PDF, PostScript, PCL, or ZPL format, as required.

As illustrated below, a form consists of a set of XML files that are combined by Adobe document services. It is not necessary to view or edit XML files to use the Adobe LiveCycle suite of products. However, it is
essential to understand the operations that occur behind the scenes if you want to develop complex forms.

Attachments

This document includes attachments that provide detailed examples for some of the tasks described in this document. You can access the attachments on the Attachments tab. The following table summarizes where the attachments are used.

<table>
<thead>
<tr>
<th>Task</th>
<th>Related attachments</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Create Accessible Multiline Tables” on page 25</td>
<td>multiline_output.pdf</td>
</tr>
<tr>
<td></td>
<td>multiline_table.xdp</td>
</tr>
<tr>
<td></td>
<td>multiline_table.xml</td>
</tr>
<tr>
<td>“Create Accessible Tables with Out-of-Place Rows” on page 42</td>
<td>out_of_place.pdf</td>
</tr>
<tr>
<td></td>
<td>out_of_place.xdp</td>
</tr>
<tr>
<td></td>
<td>out_of_place.xml</td>
</tr>
<tr>
<td>Create Accessible, Multiline, Heterogeneous Header Tables page 37</td>
<td>mhh_table.xdp</td>
</tr>
<tr>
<td></td>
<td>mhh_table.xml</td>
</tr>
<tr>
<td></td>
<td>mhh_table.pdf</td>
</tr>
<tr>
<td>Create Accessible Horizontal Tables</td>
<td>horz_tables.xdp</td>
</tr>
<tr>
<td></td>
<td>horz_tables.xml</td>
</tr>
<tr>
<td></td>
<td>horz_tables.pdf</td>
</tr>
</tbody>
</table>
You can open and save attachments in Adobe Acrobat® Professional, Acrobat Standard, or Adobe Reader®. However, you must have an application installed that can handle the file format of the attachment. Any changes you make to an attachment are not applied to the attachment. Instead, save a copy of the attachment, make the required changes to the file, and then reattach it to the PDF document.

To open an attachment in this PDF document:
1. In the Attachments tab, select the attachment.
2. Click Open.

To save a copy of one or more attachments:
1. In the Attachments tab, select one or more attachments.
2. Click Save.
3. Save the attachments:
   1. To save a single attachment, name the file, specify a location, and then click Save.
   2. To save multiple attachments, specify a location, and then click OK.

### Related Documentation

For additional information about LiveCycle Designer, refer to the following online resources:

**Adobe website:** The website contains documents that you can download, such as the *Adobe XML Forms Architecture (XFA) Specification*:


**LiveCycle Designer Help:** *LiveCycle Designer Help* contains reference information and recommendations to follow when creating form designs.
Select Form Content in the Hierarchy Palette

Use the Hierarchy palette to select objects in a form design. Some objects in a form design are difficult to see without inspecting the Hierarchy palette. In form designs that use tables or nested subforms, adjacent objects have little discernible space between them, which makes it difficult to select an object. Using the Hierarchy palette makes it easier to select, resize, or copy objects when there is little space between them.

For example, in the table illustrated below, use the Hierarchy palette instead of selecting the table row in the form design. To add another header row, select the SpanningHeaderRow object in the Hierarchy palette, and then right-click to insert the row.
Choose Fonts That Ensure Accurate Output and Minimal Output Size

Font selection

Adobe document services follow these steps when a font is used in a form design and rendered in a form:

1. **The form developer chooses the fonts**: LiveCycle Designer displays the list of fonts that are installed on the form developer’s computer. All of the fonts look the same on-screen and when printed.

2. **Adobe document services render the form design**: Adobe document services use an XML device control (XDC) file to determine which fonts are supported for a particular output device, such as Acrobat, PostScript, or a PCL device. The font-mapping table, Designer.xci, lists the font-substitution rules.

3. **Adobe document services embed fonts in the output stream, as required**: If an output device does not support a particular font, Adobe document services can embed all or part of the font as part of the output stream. A setting in the font-mapping file (XCI) controls the ability to embed a font.

Font substitution

Using the Missing Fonts dialog box, you can select substitute fonts for form designs that use fonts that are not available on your computer. Use the Missing Fonts dialog box to select a substitute font for the unavailable fonts. After you select the substitute fonts or accept the default fonts for a document, the document is displayed using those fonts.
In the following example, the form developer used the Times font to render a form on a PostScript printer that contains the Times font (Times is a printer-resident font). In some cases, a form must use a particular font for legal reasons. Because LiveCycle Designer cannot accurately display the Times font, it displays this dialog box and prompts the user to select a substitute font. The form design is not modified in this case.

Fonts that ensure output device support

For print forms, use fonts that all the output devices support to ensure that the output device can render a form design as intended. The following fonts are supported by Adobe document services for PDF and most PCL, and PostScript output devices:

- Courier
- Arial
- Times New Roman

Helvetica and Times fonts are not part of the recommended list but will render on a generic PDF or older PostScript printers. However these fonts are not installed on a desktop PC and therefore cannot be previewed in Designer. If a legal form requires the use of the Times font that is resident on the printer, then the Times font can be specified by manually entering the name on the font field in Designer. However, the preview in Designer will not be accurate.

For interactive forms it is best to use fonts that are included with Adobe Reader. The best font choices are:

- Myriad Pro
- Minion Pro

For form designs that contain non-Latin-1 characters, use the Kazuko, Adobe Ming, Adobe Song, Adobe Heiti and MyungJo fonts for Asian characters or the Adobe Arabic, Adobe Thai, and Adobe Hebrew fonts. Notice that these fonts will perform well in Acrobat but will be sent as a embedded fonts to the printer.
If an output device does not support a particular font, Adobe document services can embed the font as part of the output stream. However, embedding fonts enlarges the size of the output stream. To minimize the size of the output stream, Adobe document services embed only the subset of the font that the rendered form uses. These settings are controlled within the font-mapping (XCI) file.

**Font mapping on the server and client**

It is possible to map one font to another on both the client machine, where for the form is being developed and on the server, when the form is rendered for use by the end user.

On the client machine, in Designer, it is possible to map fonts using the dialog shown below. This dialog is available by clicking on Tools/Options, selecting “Document Handling” and clicking on the “Modify Font Substitutions...” button.

If you map fonts, the mapping of these fonts only affects the output on the client machine. The contents of the template are not affected in any way.

If you wish to make font substitution on the server, then it is necessary to modify a text file on the server called CUSTOM_XFA.XCI. This file is located in the server in the following directory:

C:\usr\sap\<sapsid>\SYS\global\AdobeDocumentServices\lib (Windows)

/usr.sap/<sapsid>/SYS/global/AdobeDocumentServices/lib (UNIX)

where: <sapsid> is a three letter ID associated with your installation of the SAP server software.
If you want data to appear in a form design, you need to create a data binding. A *data binding* is an association between the objects in a form design and a node in a data source. It lets you build a form that captures data for enterprise infrastructures and use an external data source that populates a form at run time.

You can use either explicit or implicit data binding in a form design:

**Implicit binding** Adobe document services creates implicit bindings by matching the names of the objects in the form design to the corresponding nodes in the data file. This automatic data matching is the default behavior for objects that are added from the Library palette. It corresponds to the Normal option in the Default Binding list in the Binding tab of the Object palette. For implicit binding to work, the nodes in the data file and the fields on the form design must have the same names.

**Explicit binding** Adobe document services uses the specified data references, instead of object names, to map nodes in the data source to objects in the form design. Explicit binding overrides the automatic data matching and points directly to a data node.

**Note:** It is strongly recommended that you do not use both explicit and implicit binding in the same form design, particularly if the form design is interactive.

### Absolute and relative binding expressions

You can create explicit bindings using absolute or relative binding expressions.

**Absolute binding expressions** An absolute binding expression is a fully qualified SOM expression describing a data node, which means that it starts from $record or $data. To create an explicit binding with an absolute SOM expression, drag an individual node from the Data View palette onto the form design. For example, if you drag the Field1 node from the Data View palette onto the page to create a new form object, the default binding value is $record.maingroup.Field1.

**Relative binding expressions** A relative binding expression is a partial binding expression. It is evaluated relative to the data node that is bound to the containing object. To create an explicit binding with a
relative binding expression, drag a parent node and its children from the Data View palette onto the form design. This creates an absolute binding expression for the parent object and relative binding expressions for its children.

As illustrated below, when you drag the maingroup node and its children onto the form design the binding value for the maingroup object is $record.maingroup. The children of the maingroup object automatically bind to the corresponding nodes in the data source.

The rendered form contains the values from the bound data source.

Use explicit binding and relative binding expressions

Use explicit binding instead of implicit binding whenever possible to ensure that the bound data in the form is accurate.

Use relative binding expressions, because they require less processing when Adobe document services merge the data with a form design. Absolute binding expressions specify each node in the path to the bound node, such as $record.group.field. For each absolute binding expression, Adobe document services must navigate the data hierarchy to locate the specified node, which is time consuming. Relative binding expressions specify only as much of the path as required to identify the node, such as field.

For information about how data is merged with form designs, see the LiveCycle Designer Help and the Adobe XML Form Object Model Reference.

Note: Avoid using a lot of subforms; they can degrade performance.
Use Repeating Subforms to Show or Hide Objects

You can use either repeating subforms or scripting to conditionally show or hide objects when a form is rendered. Use repeating subforms instead of scripting when you need to process a large number of records.

Scripting object presence

You can script with the `presence` property to show or hide an object based on some condition.

For example, on an invoice, if the customer does not have a balance owing, the rendered form should not display the statement about the amount to pay.

```
if (Balance.rawValue ne 0) then
    PaymentRequired.presence = "visible"
else
    PaymentRequired.presence = "invisible"
endif
```

Repeating subforms

If you want to show or hide objects on a form and you need to process a large quantity of records, it is more efficient to use repeating subforms than to use scripting. A subform can repeat the rendering of its objects as long as it is bound to a data source and nested inside a subform that flows content.

To use repeating subforms, you must set up the data structure to match the subform structure. In the data structure, nest the objects that you want to show or hide under a parent node and then bind the parent node to a repeating subform.

In the form design, the objects are bound to the corresponding data nodes. For example, in the form design for the invoice, the PaymentRequired object is a repeating subform that is bound to the PaymentRequired node in the data source. The DateTimeField object corresponds to the DueDate node.
and the DecimalField1 object corresponds to the Amount node. A static text object contains the statement about the balance owing.

Note: Floating fields are objects that support the merging of text, numeric values, run-time properties, and scripting within a text object when the form is rendered. You can insert floating fields into text objects only. You can bind floating fields to a data source to display specific text or numeric values.

In the Object palette, the Binding tab displays the Repeat Subform For Each Data Item option. Set the Min Count value to 0 or leave it blank, and set the Max value to 1 so that the subform repeats a minimum of zero times and a maximum of one time. If the Min Count value is set to 0 and no data is provided for the objects in the subform at data-merge time, the subform is hidden when the form is rendered.
Use Dynamic Properties to Bind Data to a List

Use the Dynamic Properties feature to bind objects in an XML data stream to a list, such as a drop-down list. Using dynamic properties to populate drop-down lists is faster than using a script-based solution.

When creating form designs that use drop-down lists, structure the XML data so that a list of child nodes is displayed below a parent node. This data structure makes it easier to map the XML data to the drop-down list, as shown in this example:

```xml
<currencies>
  <Currency>USD</Currency>
  <Currency>GBP</Currency>
</currencies>
```

To bind data to a drop-down list:

1. To enable Dynamic Properties in LiveCycle Designer, select Tools > Options > Data Binding.

![Dynamic Properties Screen](image)

2. To bind the following XML data to a drop-down list, select the drop-down list in the form design.

```xml
<?xml version="1.0"?>
<data>
  <currencies>
    <Currency>USD</Currency>
    <Currency>GBP</Currency>
    <Currency>CAN</Currency>
    <Currency>EUR</Currency>
  </currencies>
</data>
```
3. In the Object palette, click the Field tab and select List Items.

![Object palette screenshot]

4. In the Dynamic Properties dialog box, select the following binding.

![Dynamic Properties dialog box screenshot]

The resulting interactive form contains the list of items in the drop-down list.

![Interactive form screenshot]
Use Field Patterns for Universal Formatting

Use field patterns to ensure that field values can be displayed in all the required language locales.

Field patterns

Field patterns are strings of characters that control how field values, such as text fields, numeric fields, and date/time fields, are formatted at run time:

- A display pattern describes how data will be displayed in the form. If you define an initial default value, it is formatted according to the display pattern. The display pattern is also responsible for formatting user input and any bound values retrieved at run time.
- An edit pattern describes the syntax for entering data into a date/time field, numeric field, text field, or password field at run time.
- A validation pattern validates user input at run time.
- A data pattern describes the syntax of bound or saved data.

Use field patterns only if the results of the pattern are correct for all the affected locales.

The field pattern syntax is locale independent and the rendered value is locale dependent. For example, if the decimal separator character (.) is used in a display pattern, such as 99.99, the character that is rendered depends on the locale of the form. If the locale is German, the decimal separator character is the comma (,).

In the following example, all three fields contain the value 1234567.89. The second and third fields in the example use the display pattern $zzz,zzz,zz9.99. The dollar sign ($) specifies the currency symbol, and the value is rendered using the locale of the object. When the locale of the object is German, the dollar sign ($) is mapped to the Euro symbol.

<table>
<thead>
<tr>
<th>Field Type</th>
<th>Format</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plain Decimal Field</td>
<td>1,234,567.89</td>
<td></td>
</tr>
<tr>
<td>Picture Mask (Locale: Canadian)</td>
<td>$1,234,567.89</td>
<td></td>
</tr>
<tr>
<td>Picture Mask (Locale: German)</td>
<td>€1,234,567.89</td>
<td></td>
</tr>
</tbody>
</table>

For more information about field patterns, see the LiveCycle Designer Help.
Determine the locale of a form

The locale of a form is determined by the following properties:

1. The Locale property of the object if it is set.
2. The Locale property of the form if it is set.
3. The ambient locale. If a form is designed using the ambient locale, the form uses the locale of the computer on which it is rendered. This is true for all forms, including print forms and read-only fields on interactive forms.

**Note:** The ambient locale is also known as the viewer locale.
Ensure Forms Are Accessible

Many government agencies require electronic forms to be accessible to users with vision impairment. To make PDF forms accessible when rendered, specify a tagging structure that is included with the rendered PDF form. The tagging structure uses elements that resemble HTML tags and is read aloud by third-party screen reader software.

Screen reader software
For example, JAWS from Freedom Scientific

Tag structure exposed by the document
You can use Acrobat Professional to view the accessibility tags for a particular PDF form:

- To see the tags in Acrobat 7.0 Professional, select View > Navigation Tabs > Tags.
- To view the corresponding fields for a particular tag, in the Tags window, select Options > Highlight Content.
Create accessible forms

To ensure that forms are accessible, each object should include screen reader text. For each field object in a form, you can specify one of several settings for screen reader text:

- Custom screen reader text, which you set in the Accessibility palette.
- Tool tips for objects, which you set in the Accessibility palette.
- Captions for fields.
- Names of objects, as specified in the Name option of the Binding tab. Set the screen reader text in the Accessibility palette.

**Note:** For objects with captions, such as text fields, radio buttons, and check boxes, use the captions and do not set custom text.

To design accessible forms, follow these recommendations:

**Use tables wherever possible:** Tables help structure the content and enable the screen reader to identify the current location within a table. The screen reader uses the accessibility tags to read the column heading and row location.

**Use subforms that flow content:** Use subforms that flow content instead of subforms that position content to ensure that the accessibility tags are generated in the appropriate order. In subforms that position content, accessibility tags are generated left to right and top to bottom, which may not be the expected reading order.
Use Subforms That Flow Content

Subforms are container objects that you can use to group form design objects, including fields, boilerplate objects, and other subforms. You can configure subforms either to flow content or to position content.

Subforms that flow content

Subforms that flow content allow the layout engine in the Adobe document services to position the objects within the subform. Subforms can hold varying amounts of data and flow content according to the Flow Direction setting of its associated content area. The objects within the subform move together during the data-merging process so that none of its objects interfere with each other.

By default, the root subform is defined to flow content. Consequently, all subforms nested under the root subform can flow from one form page or content area into the next when merged with data.

In subforms that flow content, the content can flow either from top to bottom or as Western text (left to right).
Subforms that position content

Subforms that position data can expand to fit any amount of merged data, but the objects within the subform cannot move from their anchor points. By default, all subforms except the root subform are defined to position content.

If a subform contains objects that merge with variable sizes of data, you should verify that those objects do not expand to the extent that they overrun the area occupied by another object. It is possible that expandable objects, such as text fields, might be rendered on top of other objects. To avoid this design concern, set the subform to flow and expand to fit the content.

When designing a dynamic form, complete the form design first and then set the default body page subform to flow content. Set the default body page subform to flow content last so that the subform remains visible and the objects that you place within the subform remain in the intended position on the body page.

Flowed and positioned content

When designing forms, use the subforms that flow content property to avoid accessibility or navigation problems. Accessibility tags are generated from left to right and then from top to bottom. Accessibility problems can occur in subforms that position content when the objects are not positioned precisely; for example, the accessibility tags are not generated and read in the correct order.

The example below illustrates the accessibility problems that can occur in a subform that positions content. The objects in the subform appear to be aligned vertically; however, the vertical heights actually differ by 1/10 inch.
In this example, the screen reader will read the text field on the right before the one on the left, which is not the intended reading order.

Accessibility tags are read from left to right and then from top to bottom.

To avoid accessibility problems, enclose the objects in a subform that flows content. To create space between the objects, use the margin values in the Layout palette. To position the caption and text for an object, use the Paragraph palette.
Use Nested Subforms to Structure Form Content

Use nested subforms to structure form content and to ensure that screen readers read the form content in the correct order. You can use nested subforms to create a form that looks and functions like a table. Use nested subforms instead of tables because the screen reader software reads the table structure. For example, the screen reader software reads “table with 4 columns” before it reads the contents of the table.

Use subforms to structure content because they provide anchoring, layout, and geometry management for objects. You can arrange the objects in a subform in rows, columns, or some other kind of balanced arrangement.

Nested subforms function like tables. Tables are flow content subforms that contain a repeatable subform with data, such as a body row. They ensure that the heights and widths of adjacent subforms remain consistent and export accessibility information (see $paratext>$ on page 20). Like tables, nested subforms are useful for structuring form content when the content originates from a relational database and contains multiple repeated record types or when nested data exists.

To structure the form content using nested subforms, do not set the Accessibility Role for the subforms. For more information about how to create accessible form designs using nested subforms, see the example in “Create Accessible Multiline Tables” on page 27.

A form design that uses nested subforms to structure the content nests the subforms in a hierarchy.

Form design with hierarchy of nested subforms

| No. of subscribers as per last month return | 0 |
| (+) No. of new subscribers-Vide Form No. 4(FPF) | 1 |
| (-) No. of subscribers left service-Vide Form 5(FPF) | 0 |
| Total | 1 |
Create Accessible Multiline Tables

These guidelines use an example to explain how to create an accessible, multiline table.

**Note:** See the attached files for the complete solution used in this example: multiline_output.pdf, multiline_table.xdp, and multiline_table.xml.

This example explains the steps required to create the following accessible multiline table.

<table>
<thead>
<tr>
<th>Carrier ID Amount</th>
<th>Number Currency</th>
<th>Flight Date Order Date</th>
<th>Luggage Weight</th>
<th>Weight Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>AZ 1,704.45</td>
<td>0555 SGD</td>
<td>Feb 2, 2005, Aug 31, 2004</td>
<td>25.3</td>
<td>KG</td>
</tr>
<tr>
<td>AZ 461.00</td>
<td>0789 EUR</td>
<td>Jul 21, 2004, Sep 17, 2004</td>
<td>22.3</td>
<td>KG</td>
</tr>
<tr>
<td>JL 961.00</td>
<td>0407 EUR</td>
<td>Apr 30, 2004, May 4, 2004</td>
<td>9</td>
<td>KG</td>
</tr>
<tr>
<td>JL 0407</td>
<td></td>
<td>Jul 23, 2004</td>
<td>27.5</td>
<td>KG</td>
</tr>
</tbody>
</table>
Using subforms to emulate a table

Instead of using Table Designer, build this form design with a set of nested subforms and then set the accessibility role of the subforms to emulate a table. Use nested subforms instead of Table Designer because a table with multiple header and body rows is not accessible. As illustrated below, when a table has multiple rows, the form generates multiple lines of accessibility tags. The form design generates these accessibility tags even if the body rows are grouped in a table section.

This example assumes that a data schema is set up for the form design. If you develop forms in the SAP environment, the data schema is generated from the context when the form is opened in LiveCycle Designer. As illustrated the Data View palette contains a set of fields that you want to put in the table.
To turn off the table generation option:

1. Drag data fields onto the form without creating a table. Tables do not support wrapped header and body rows, which are necessary to make this form design accessible.

2. Click the Data View palette menu and deselect Allow Tables To Be Generated.

To create the data bindings:

1. Drag the BOOKINGS data node from the Data View palette to the far left side of the form.

To create the Header subform:

1. In the Hierarchy palette, click the DATA subform.

2. To create the header subform, select Edit > Duplicate. Duplicating the subform makes it easier to create the headings from the DATA subform text field captions.

   There are now two subforms named DATA (DATA[0] and DATA[1]).
3. Rename the first DATA subform to Header. The form design should look like this illustration.

When the DATA subform was duplicated, the Header subform inherited two properties that are not required.

4. In the Object palette, click the Binding tab and deselect Repeat Subform For Each Data Item.

5. From the Default Binding list, select Normal.
To create captions for the Header subform:

1. In the Hierarchy palette, select all of the text fields in the Header subform.
2. In the Object palette, click the Draw tab and select Text from the Type list. Setting the type to text converts all of the text fields to static text and copies the text field captions.

To remove the captions from the DATA subform:

1. In the Hierarchy palette, select all of the text fields in the DATA subform.
2. In the Layout palette, under Caption, select None from the Position list.
Making the table accessible

For the screen reader software to recognize the subforms as a table, the `<Table>`, `<TR>`, `<TH>`, and `<TD>` tags must be generated as part of the PDF file. To generate the accessibility tags, you must set the Subform Role field for the BOOKINGS, Header, and DATA subforms.

To set the accessibility role for each subform:

1. In the Hierarchy palette, select a subform.

2. In the Accessibility palette, select the Subform Role for each subform as follows:

<table>
<thead>
<tr>
<th>Subform</th>
<th>Subform Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOKINGS</td>
<td>Table</td>
</tr>
<tr>
<td>Header</td>
<td>Header Row</td>
</tr>
<tr>
<td>DATA</td>
<td>Body Row</td>
</tr>
</tbody>
</table>

Enabling multipage tables

The parent subform on the body page should be set to flow content instead of position content. When the subform is set to flow content and the form is rendered with XML data, the table can span multiple pages.

To enable the table to span multiple pages:

1. Select the parent subform.
2. In the Object palette, click the Subform tab and select Flowed from the Content list.

3. From the Flow Direction list, select Top to Bottom so that the form can render data on multiple pages. However, the Header appears only on the first page.
To set the Header subform to appear at the top of each page:

1. In the Hierarchy palette, select the DATA subform.

2. In the Object palette, click the Pagination tab. Under If Dataset Must Be Paginated, select Header from the Overflow Leader list.

3. (Optional) To use the subform as an overflow leader, in the Subform tab, select Positioned from the Content list.
Arranging and resizing the fields

Configure the subforms so that the rendered output resembles a table. Before you arrange and resize the fields, the form design looks like this illustration.

![Table Illustration]

After you arrange and resize the fields, the form design should look like this illustration.

![Table Illustration]

To arrange the fields as a body row in a table, you must set the flow content of the subform to Western Text, which flows the fields from left to right and top to bottom within the subform. Changing the flow content of the subform initially sets the direction to top to bottom; therefore, you must ensure that the width of the subform is fixed and will not automatically adjust. Otherwise, when you set the flow content of the subform, it will not change the width of the subform to a size that exceeds the content area.
To arrange the DATA and Header subform fields:

1. In the Hierarchy palette, select the DATA subform.
2. In the Layout palette, deselect Auto-Fit.
3. In the Object palette, click the Subform tab and select Flowed from the Content list. This resizes the subform so that all fields are vertically arranged.
4. From the Flow Direction list, select Western Text. The subform now arranges the fields in two rows, each with four fields. The fields are arranged next to each other without requiring manual resizing.

5. In the Hierarchy palette, select all of the fields in the DATA subform.

6. In the Object palette, click the Field tab and select Solid from the Appearance list.

7. In the Layout palette, set the margins to 0.

8. In the Hierarchy palette, select the Header subform and then repeat steps 1 to 4.

To resize the Header and DATA subform fields:

1. In the Hierarchy palette, select the first field in the DATA subform.

2. Ctrl+click the first field in the Header subform and then select Layout > Make Same Size > Width. This resizes the Header field to match the DATA field.

3. Resize the Header and DATA subforms so that they are the same width as the four fields in the first row in the DATA subform.

4. Resize all of the Header fields so that they are the same size as the first field. The form design should now look like this illustration.

5. To use the Header subform as an overflow leader that appears on each page, in the Object palette, click the Subform tab and select Positioned from the Content list.

6. To set different column widths, do these tasks:
   1. Select the DATA subform and select Positioned from the Content list.
   1. Adjust the width of a field in the Header subform and then use the Layout > Make Same Size > Width command to set width of the corresponding DATA field.

   Note: Resetting the DATA subform to flow content helps arrange the fields but is not mandatory.
7. Select the DATA subform, click the Layout palette, and set the margins to 0.

8. (Optional) You can adjust the alignment of columns to suit the type of data that is merged with the form design.

The rendered form looks like this illustration.

<table>
<thead>
<tr>
<th>Carrier</th>
<th>Order #</th>
<th>Purchase</th>
<th>Purchase Date</th>
<th>Order Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9800</td>
<td>6000</td>
<td>2004-01-01</td>
<td>2004-01-01</td>
</tr>
<tr>
<td>B</td>
<td>8765</td>
<td>5432</td>
<td>2004-02-02</td>
<td>2004-02-02</td>
</tr>
<tr>
<td>C</td>
<td>7654</td>
<td>4567</td>
<td>2004-03-03</td>
<td>2004-03-03</td>
</tr>
<tr>
<td>D</td>
<td>8901</td>
<td>7890</td>
<td>2004-04-04</td>
<td>2004-04-04</td>
</tr>
</tbody>
</table>

The rendered form looks like this illustration.
Create Accessible, Multiline, Heterogeneous Header Tables

A multiline, heterogeneous header table has a multiline header row that spans several columns.

| Amount of Contributions due as per recoveries made in the Wages/equittance register |
|---------------------------------|---------------------------------|------------------|
| Worker’s Share (8.33%)          | Employer’s Share (8.33%)        | Total            |
| (3)                             | (4)                             | (5)              |
|                                 | 1.666,00                        | 1.666,00         |

To create this table, use the Table Designer and then use column spanning on the first header row. The sample files for this document are mhh_table.xdp, mhh_table.xml and mhh_table.pdf.

To create a multiline, heterogeneous header table:

1. Select Table > Insert Table. Create a simple table with three columns, one body row, and one header row.
2. In the Hierarchy palette, right-click the first header row and then select Insert > Row Below.
3. Select all three cells in the first header row, and then right-click and select Merge Cells.

The form design looks like this illustration.
The screen reader software, such as JAWS, should read the main header along with individual cells from the second header for each body row cell in the table.

For example, for the following table:

<table>
<thead>
<tr>
<th></th>
<th>H 1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H 2 1</td>
<td>H 2 2</td>
<td>H 2 3</td>
<td></td>
</tr>
<tr>
<td>Data 1</td>
<td>Data 2</td>
<td>Data 3</td>
<td></td>
</tr>
</tbody>
</table>

JAWS would read the table positions as follows:

| Cell Data 1 | H 1 H 2 1 |
| Cell Data 2 | H 1 H 2 2 |
| Cell Data 3 | H 1 H 2 3 |

For the following, more complex, example:

<table>
<thead>
<tr>
<th></th>
<th>H 1 1</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>H 2 1</td>
<td>H 2 2</td>
<td>H 2 3</td>
<td></td>
</tr>
<tr>
<td>Data 1</td>
<td>Data 2</td>
<td>Data 3</td>
<td></td>
</tr>
</tbody>
</table>

JAWS would read the table positions as follows:

| Cell Data 1 | H 1 1 H 2 1 |
| Cell Data 2 | H 1 1 H 2 2 |
| Cell Data 3 | H 1 2 H 2 3 |

**Note:** At this time (August 15, 2012), only the following supported software solution is accessible:

- JAWS 7.0
- Adobe Reader 7.07 (available in early 2006)

It is recommended that forms requiring multiline, heterogeneous header tables use this kind of design. For this solution to comply with accessibility requirements, the JAWS software may require fixes.
Create Accessible Nested Tables

It is possible to design nested tables by using the LiveCycle Designer table features. As illustrated here, you can create a nested table by selecting a cell on the outer table and then inserting a second table by using the Insert Table menu command.

JAWS 7.0 reads nested tables as layout tables. Layout tables are used to arrange objects on the screen but do not imply any hierarchical relationship. Although Adobe Reader provides a hierarchy of table structures to JAWS, JAWS cannot read the inner table.
Create Accessible Horizontal Tables

If you need to flow data horizontally across each page, use a set of nested subforms instead of using Table Designer. The sample files are this best practice are horz_tables.xdp, horz_tables.xml and horz_tables.pdf.

<table>
<thead>
<tr>
<th>CARRID</th>
<th>AZ</th>
<th>AZ</th>
<th>AZ</th>
<th>DL</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNID</td>
<td>0555</td>
<td>0788</td>
<td>0789</td>
<td>1699</td>
</tr>
<tr>
<td>LUGGWEIGHT</td>
<td>25.3000</td>
<td>20.2000</td>
<td>22.3000</td>
<td>27.7000</td>
</tr>
<tr>
<td>WUNIT</td>
<td>KG</td>
<td>KG</td>
<td>KG</td>
<td>KG</td>
</tr>
<tr>
<td>FORCURAM</td>
<td>185.00</td>
<td>1704.45</td>
<td>944.95</td>
<td>481.00</td>
</tr>
<tr>
<td>FORCURKEY</td>
<td>EUR</td>
<td>SGD</td>
<td>USD</td>
<td>EUR</td>
</tr>
</tbody>
</table>

For more information about creating an accessible multiline table, see ?$paratext>? on page 27.

To create this form, you must make the following changes:

- For the Header subform, set the Content option to Positioned.
- For the DATA subform, set the Content option to Positioned.
- For the DATA subform, turn off the Allow Page Breaks within Content option.

**Caution:** At this time, this is not an officially recommended solution for creating accessible horizontal tables. JAWS 7.0 is not reading the accessibility tags for this solution as intended. JAWS 7.0 uses the physical location of the cells on the page to analyze the table structure. This document will be updated when this problem is resolved.
The form renders as follows.

<table>
<thead>
<tr>
<th>CARRIED</th>
<th>SL</th>
<th>AL</th>
<th>AL</th>
<th>AL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUGGAGE/</td>
<td>11.3000</td>
<td>17.8000</td>
<td>23.4000</td>
<td></td>
</tr>
<tr>
<td>WEIGHT</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CARRIED</th>
<th>SL</th>
<th>AL</th>
<th>AL</th>
<th>AL</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUGGAGE/</td>
<td>21.3000</td>
<td>17.8000</td>
<td>23.4000</td>
<td></td>
</tr>
<tr>
<td>WEIGHT</td>
<td>2.0</td>
<td>2.0</td>
<td>2.0</td>
<td></td>
</tr>
</tbody>
</table>
Create Accessible Tables with Out-of-Place Rows

This example of a table with out-of-place rows has six columns. The seventh data field is comment text that appears in some of the body rows of the table.

<table>
<thead>
<tr>
<th>Beleg-Nummer</th>
<th>Beleg-Datum</th>
<th>Vorgang</th>
<th>GsBer</th>
<th>Währung</th>
<th>Betrag</th>
</tr>
</thead>
<tbody>
<tr>
<td>18000000021</td>
<td>16.04.2005</td>
<td>Debitoren Rechnung</td>
<td>JPY</td>
<td></td>
<td>105.00</td>
</tr>
<tr>
<td>18000000022</td>
<td>16.04.2005</td>
<td>Debitoren Rechnung</td>
<td>EUR</td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>18000000023</td>
<td>16.04.2005</td>
<td>Debitoren Rechnung</td>
<td>USD</td>
<td></td>
<td>133.37</td>
</tr>
<tr>
<td>18000000024</td>
<td>16.04.2005</td>
<td>Debitoren Rechnung</td>
<td>EUR</td>
<td></td>
<td>147</td>
</tr>
<tr>
<td>18000000025</td>
<td>16.04.2005</td>
<td>Debitoren Rechnung</td>
<td>EUR</td>
<td></td>
<td>122.79</td>
</tr>
<tr>
<td>18000000026</td>
<td>01.05.2005</td>
<td>Debitoren Rechnung</td>
<td>EUR</td>
<td></td>
<td>720</td>
</tr>
<tr>
<td>18000000003</td>
<td>01.08.2005</td>
<td>Debitoren Rechnung</td>
<td>JPY</td>
<td></td>
<td>1.500.00</td>
</tr>
<tr>
<td>01000000000</td>
<td>15.08.2005</td>
<td>Buchhaltungsbeleg</td>
<td>JPY</td>
<td></td>
<td>335.18</td>
</tr>
<tr>
<td>01000000001</td>
<td>15.08.2005</td>
<td>Buchhaltungsbeleg</td>
<td>EUR</td>
<td></td>
<td>60.500</td>
</tr>
<tr>
<td>18000000000</td>
<td>15.08.2005</td>
<td>Debitoren Rechnung</td>
<td>EUR</td>
<td></td>
<td>115.73</td>
</tr>
<tr>
<td>18000000001</td>
<td>15.08.2005</td>
<td>Debitoren Rechnung</td>
<td>USD</td>
<td></td>
<td>230.85</td>
</tr>
<tr>
<td>18000000004</td>
<td>15.08.2005</td>
<td>Debitoren Rechnung</td>
<td>JPY</td>
<td></td>
<td>23.000</td>
</tr>
<tr>
<td>18000000005</td>
<td>15.08.2005</td>
<td>Debitoren Rechnung</td>
<td>USD</td>
<td></td>
<td>423.55</td>
</tr>
</tbody>
</table>

Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten
Gesamtsaldo: Zu unseren Gunsten

For more information about this example, see the attached form design, data, and PDF file (BP15.*).

To avoid the appearance of out-of-place rows, create the form design by using a set of nested subforms, and then add an extra subform for the comment row. In the Accessibility palette, set the accessibility role of the extra subform to None.

Using nested subforms is better than creating the form design as a wrapped multiline table because the screen reader software would read the wrapped table as a 7-column table, including the headers.

For an example of how to create a set of nested subforms, see the example in ?§paratext§ on page 27.
The Hierarchy palette displays the subform structure.
Create Accessible Tables with Multiple Data Fields

If you need to place more than 10 data fields in a table, see ?paratext> on page 27. Because a large number of data fields are involved, the header and body subforms contain many rows.

Avoid creating such forms if the number of fields is large. The following example of a form that contains 57 data fields illustrates how such a form design may be very usable.

<table>
<thead>
<tr>
<th>Cl. Material</th>
<th>Created</th>
<th>Created by</th>
<th>Last Chg</th>
<th>Changed by</th>
<th>Complete status</th>
<th>Maint. status</th>
<th>Chl</th>
<th>MTyp I</th>
<th>Matl Group</th>
<th>Old material no.</th>
<th>BU n</th>
<th>OU n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gross Weight</td>
<td>Net weight</td>
<td>WUn</td>
<td>Volume</td>
<td>WUn</td>
<td>CR</td>
<td>SC</td>
<td>Temp</td>
<td>Cde</td>
<td>TGroup</td>
<td>Haz</td>
<td>material no.</td>
<td>Dv</td>
</tr>
<tr>
<td>R S Seas</td>
<td>LT</td>
<td>LF Field</td>
<td>deact.</td>
<td>EAN/UPC</td>
<td>Cl</td>
<td>Length</td>
<td>Width</td>
<td>Height</td>
<td>Unit of Dimension</td>
<td>Product hierarchy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Table Example]

![Table Example]
Creating Multiple Levels of Repeating Headers

These guidelines use an example to explain how to create an accessible table that contains multiple levels of repeating headers for a nested set of tabular data.

**Note:** See the attached files for the complete solution used in this example:

mlrh_output.pdf, mlrh_template.xdp, and mlrh_data.xml.

This example explains the steps required to create the following accessible table, with multiple levels of repeating headers:

The objective is to produce an output as follows:

```
<table>
<thead>
<tr>
<th>Date</th>
<th>LH No.</th>
<th>Date</th>
<th>LH No.</th>
<th>Date</th>
<th>LH No.</th>
<th>Date</th>
<th>LH No.</th>
<th>Date</th>
<th>LH No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>booking</td>
<td>cust no.</td>
<td>booking</td>
<td>cust no.</td>
<td>booking</td>
<td>cust no.</td>
<td>booking</td>
<td>cust no.</td>
<td>booking</td>
<td>cust no.</td>
</tr>
</tbody>
</table>
```
Creating Multiple Levels of Repeating Headers

The objective is to show detailed information for each record of data. For example, in the report shown below, details about each flight appear beneath each record:

<table>
<thead>
<tr>
<th>ID</th>
<th>LH</th>
<th>No.</th>
<th>1111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Depart. city</td>
<td>BANGALORE</td>
<td>Arrival city</td>
<td>OTTAWA</td>
</tr>
<tr>
<td>Date</td>
<td>Booking</td>
<td>Cust. No.</td>
<td>B/P cust</td>
</tr>
<tr>
<td>------</td>
<td>---------</td>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>Jun 15, 2006</td>
<td>00011111</td>
<td>00022222</td>
<td>12,1</td>
</tr>
<tr>
<td>Jun 15, 2006</td>
<td>00011111</td>
<td>00022222</td>
<td>12,1</td>
</tr>
<tr>
<td>Jun 15, 2006</td>
<td>00011111</td>
<td>00022222</td>
<td>12,1</td>
</tr>
<tr>
<td>Jun 15, 2006</td>
<td>00011111</td>
<td>00022222</td>
<td>12,1</td>
</tr>
</tbody>
</table>

This example assumes that a data schema is set up as follows:

The objective is to create a hierarchy as follows:
The “RecordHeader” static text field is a floating field containing the text fields (CARRID, CITYFROM, CITYTO and CONNID).

The steps to create this are as follows:

1. Name the main subform in the body page “BODY_PAGE” and set the “Content:” attribute in the Object palette to “Flowed”/“Top to Bottom”.

2. Create a subform called “TABLE” and set the “Content:” attribute in the Object palette to “Flowed”/“Top to Bottom”.

3. Create a child subform of “TABLE” called “HEADER”. Make sure that the “Content:” attribute in the Object palette is set to “Positioned”.

4. Create a child subform of “TABLE” called “DATA” and set the “Content:” attribute in the Object palette to “Flowed”/“Western Text”.

5. Add the static text fields shown in the picture above to the HEADER subform. The first field is a floating text field called “RecordHeader”, which will contain the text fields CARRID, CITYFROM, CITYTO and
CONNID. The floating text field should be positioned so that it is directly above and aligned with the next field (as shown below).

6. Add the data fields shown in the picture above to the DATA subform. Note that there are 6 fields in the data subform and 7 in the header subform.

Notes:

A. You cannot convert the nested subforms to a table. The floating text field in the first cell in the HEADER subform will occupy a significant amount of space and there is no corresponding data.

B. For screen reader software JAWS 7.1 (as of July 23, 2006), this solution will work properly. This is because JAWS examines the screen contents in addition to counting the elements in the hierarchy when determining the number of columns in the table. In this case, JAWS will report that this is a table with 6 columns, which is the intention of this design. However, this solution is dependent on the fact that future versions of JAWS will continue to behave this way, which is not certain at this time (July 23, 2006).
Preserving decimal digits when calculating

This example explains the steps that need to be done to preserve the number of decimal digits when performing calculations.

**Note:** See the attached files for the complete solution used in this example:

preserve_decimal_places.xdp and preserve_decimal_places.pdf

**Background:**

In order to format numeric fields, you can set a display pattern (such as “zz9.99”) in the “Field” tab of the “Object” palette. This will ensure that a value such as “42.0” would be shown as “42.00”

This display pattern is also applied if the value is rendered in the “calculate” event.

However, if the number of decimal digits that need to be preserved is dynamic and only determined at run-time, then a script is required to format the data depending on the number of decimal digits are involved in the source value(s).

A typical use case is when it is necessary to calculate the totals for different types of currencies on the same form. For US dollars, two decimal digits are used. For Japanese Yen, there are no decimal digits.

For example, in the example below, the total amount needs to use 2 decimal places.

| Amount 1 | 345.67 |
| Amount 2 | 234.23 |
| Total    | 579.90 |

Without proper formatting, the total would appear as “579.9”
However, if the values had no decimal digits at all, then the value of the “Total” field needs to reflect this.

<table>
<thead>
<tr>
<th>Amount 1</th>
<th>345</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount 2</td>
<td>234</td>
</tr>
<tr>
<td>Total</td>
<td>579</td>
</tr>
</tbody>
</table>

In order to set the number of decimal digits, a script in the “calculate” event field is needed.

This script will perform the following steps:

1. Determine what the decimal separator character is for the form (e.g. “.” for US, “,” for DE, etc.)
2. Determine the number of decimal digits for one of the fields being used in the calculation.
3. If necessary, add trailing zeros to the target field.

The script for the “calculate” event for the Total field will look as follows:

```
This assumes that the fields being summed are an array with the name “fld”.

NOTE: The Calculate script needs a value to be the result of the script, so the resulting variable is the final line of the script.
var Total = sum(fld[*])

// Extract the character that is used to indicate the decimal place
// For example: For US locale, this is the period (".")
//              For German locale, this is the comma (",")
// This is done by extracting the character from a formatted
// preset value (9.9)
var chardec = substr(format("9.9",1.2),2,1)

// Compute the # of decimal places in the source field
var SrcValue = fld[0].formattedValue
// now add in the decimal digits into the total
if (At(SrcValue , chardec) ne 0) then
  var DigitCount = Len(SrcValue ) - At(SrcValue ,chardec)
  // format the total using the same # of decimal places
  Total = Ltrim(Str(Total,len(Total)+DigitCount+1,DigitCount))
endif

Total

*Note that the script is written in FormCalc.*

The JavaScript version is as follows:

```
// amt1 and amt2 are the fields involved
// amt1 will be the source field
var Total = Number(amt1.rawValue) + Number(amt2.rawValue);

// determine the number of decimal digits by subtracting the number of
// characters in the whole number version of the value from the number of
// characters in the original numeric value
// substract one to account for the decimal separator
var NumDigits = String(amt1.rawValue).length -
    String(Math.floor(amt1.rawValue)).length - 1;

Total.toFixed(NumDigits);
```

In the attached template (preserve_decimal_places.xdp/pdf), the “calculate” event of the “Total” field contains the script. The attached sample is an interactive form, so you can test out various input scenarios.

Both the JavaScript and FormCalc versions are included on the attached form.
Fitting a Large Graphic

This example explains the steps that need to be done if you wish to fit a large graphic onto a page. If the graphic is too large for the page, then make the graphic as large as possible and auto-fit the graphic to fit on the page. Otherwise, render the graphic using the original dimensions.

**Note:** See the attached files for the complete solution used in this example:

fit_large_graphic.xdp, fit_large_graphic.xml and fit_large_graphic.pdf

**Background:**

Auto sizing of the image fields is done by setting the “Sizing” drop down list to “Scale Image Proportionally” in the “Field” tab of the “Object” palette.

For example, in the example below, the image is set to “Scale Image Proportionally”.

If the source of the image is XML data, then it will not be known in advance if the image is larger than the form. If it is desirable to preserve the image size if it is smaller than the form, then a script will be needed to compare the image size with that of the form to determine if scaling needs to be done.
Solution:

In order to set the scaling on the image field, do the following:

1. Create two image fields, one that will remain invisible and contain the original image in the original size (GRAPHIC_HIDDEN) and one that will be visible (GRAPHIC_PRINT).

2. Make sure the parent subform of the image fields is a “positioned content” subform. This is the “Content:” drop down list on the “Subform” tab of the “Object” palette.

3. Place the following JavaScript to “run at client” in the layout_ready event of the GRAPHIC_PRINT image field:

   // NOTES:
   // 1. The parent subform has to be a positional content subform in order for this code to work.
   // 2. This routine has to be executed in the layout_ready event

   // Stores the dimensions of the image fields
   var lw1 = xfa.layout.w(GRAPHIC_HIDDEN, "mm");
   var w2 = GRAPHIC_PRINT.w;
   var lh1 = xfa.layout.h(GRAPHIC_HIDDEN, "mm");
   var h2 = GRAPHIC_PRINT.h;

   var lw2 = parseFloat(w2.substring(0, w2.length-2));
   var lh2 = parseFloat(h2.substring(0, h2.length-2));

   // Compares the dimensions of the dummy field with the print field
   if ((lw1>lw2)||(lh1>lh2)) {
      // If greater, then change Sizing to "Scale to fit"
      GRAPHIC_PRINT.value.image.aspect = "fit";
   } else {
      // If lower, then change Sizing to "Use Image Size"
      GRAPHIC_PRINT.value.image.aspect = "actual";
   }

   GRAPHIC_PRINT.rawValue = GRAPHIC_HIDDEN.rawValue;

The script compares the dimensions of the hidden graphic image field with visible image field. Since the visible image field is expanded to fit the width of the form and the parent subform is a positional subform, comparing these values will determine if the image is larger than the parent subform. If this is the case, the sizing of the image field is set to “scale to fit”.
Creating a URL based read-only graphic

This example explains the steps that need to be done if you wish to place a graphic on the form whose source data is from a URL. Additionally, the sample explains how to disable the browsing dialog for the image if the form was to be rendered as interactive.

**Note:** The solution for this example is URL_readonly_graphic.xdp and URL_readonly_graphic.pdf

**Solution:**

A graphic whose source is a URL can be created by using an Image Field on the template. For example, see below:

For this example, a URL on the web that contains an image will be used. This URL could change in the future, but the steps in this example will still apply.

In order to create the image field, do the following:

1. Create an image field and place the URL of the graphic in the URL: field in the “Field” tab of the “Object” palette.

2. Place the following JavaScript to “run at client” in the form_ready event of the image field:

```javascript
// Disables the image browsing dialog for an interactive form:
this.access = "readOnly";
```
The script will ensure that the field cannot be clicked on in the event that it is used as an interactive form.
This example explains how to create an accessible table that contains an overflow leader and trailer.

**Note:** See the attached files for the complete solution used in this example: overflow_lead_trail.pdf, overflow_lead_trail.xdp, and overflow_lead_trail.xml.

The objective is to produce an output as follows:

<table>
<thead>
<tr>
<th>Currency</th>
<th>Code</th>
<th>Date</th>
<th>Value</th>
<th>Exchange Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>EUR</td>
<td>EUR</td>
<td>2023-03-15</td>
<td>422.91</td>
<td>USD</td>
</tr>
<tr>
<td>CAD</td>
<td>CAD</td>
<td>2023-04-02</td>
<td>1.50</td>
<td>CDN</td>
</tr>
<tr>
<td>USD</td>
<td>USD</td>
<td>2023-07-21</td>
<td>109.72</td>
<td>USD</td>
</tr>
<tr>
<td>JPY</td>
<td>JPY</td>
<td>2023-02-13</td>
<td>185.00</td>
<td>JPY</td>
</tr>
<tr>
<td>TWD</td>
<td>TWD</td>
<td>2023-11-02</td>
<td>236.88</td>
<td>TWD</td>
</tr>
<tr>
<td>SGD</td>
<td>SGD</td>
<td>2023-01-03</td>
<td>175.44</td>
<td>SGD</td>
</tr>
<tr>
<td>MXN</td>
<td>MXN</td>
<td>2023-07-21</td>
<td>244.20</td>
<td>MXN</td>
</tr>
<tr>
<td>HKD</td>
<td>HKD</td>
<td>2023-05-18</td>
<td>1411.00</td>
<td>HKD</td>
</tr>
<tr>
<td>AUD</td>
<td>AUD</td>
<td>2023-10-13</td>
<td>2232.00</td>
<td>AUD</td>
</tr>
<tr>
<td>CAD</td>
<td>CAD</td>
<td>2023-02-01</td>
<td>0.41619</td>
<td>CAD</td>
</tr>
<tr>
<td>HKD</td>
<td>HKD</td>
<td>2023-10-11</td>
<td>411.02</td>
<td>HKD</td>
</tr>
<tr>
<td>TWD</td>
<td>TWD</td>
<td>2023-10-11</td>
<td>422.91</td>
<td>TWD</td>
</tr>
<tr>
<td>EUR</td>
<td>EUR</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>EUR</td>
</tr>
<tr>
<td>USD</td>
<td>USD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>USD</td>
</tr>
<tr>
<td>TWD</td>
<td>TWD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>TWD</td>
</tr>
<tr>
<td>HKD</td>
<td>HKD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>HKD</td>
</tr>
<tr>
<td>AUD</td>
<td>AUD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>AUD</td>
</tr>
<tr>
<td>CAD</td>
<td>CAD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>CAD</td>
</tr>
<tr>
<td>EUR</td>
<td>EUR</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>EUR</td>
</tr>
<tr>
<td>USD</td>
<td>USD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>USD</td>
</tr>
<tr>
<td>TWD</td>
<td>TWD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>TWD</td>
</tr>
<tr>
<td>HKD</td>
<td>HKD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>HKD</td>
</tr>
<tr>
<td>AUD</td>
<td>AUD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>AUD</td>
</tr>
<tr>
<td>CAD</td>
<td>CAD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>CAD</td>
</tr>
<tr>
<td>EUR</td>
<td>EUR</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>EUR</td>
</tr>
<tr>
<td>USD</td>
<td>USD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>USD</td>
</tr>
<tr>
<td>TWD</td>
<td>TWD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>TWD</td>
</tr>
<tr>
<td>HKD</td>
<td>HKD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>HKD</td>
</tr>
<tr>
<td>AUD</td>
<td>AUD</td>
<td>2023-01-01</td>
<td>422.91</td>
<td>AUD</td>
</tr>
</tbody>
</table>
Creating Overflow leaders/trailers

The assumption for this solution is that a form hierarchy exists as illustrated below (the attached solution files can be used in this case):

The “Summary” subform will appear at the end of the rendered form. The “Footer” subform will appear at the bottom of each page. The “Header” subform will appear at the top of each page.

The steps to create this are as follows:

1. Place the required fields in the Header, DATA, Summary and Footer subforms. The attached template solution already contains these subforms with the required fields. In this case, the BOOKINGS subform is the subform that represents the table. The “Container” subform is the parent of BOOKINGS.

2. Make sure that the “Container” and “BOOKINGS” subforms are “Flow Content” subforms, with the flow direction set to “Top to Bottom”.

3. For the “BOOKINGS” subform, in the “Pagination” tab in the “Object” palette, set the “Overflow Leader” drop down list to “Header”. Set the “Overflow Trailer” to “Footer”. NOTE: the overflow leader and trailer have to be set in the parent subform of the leader/trailer, not in the affected subform (which in this case is the “DATA” subform).

4. For the “Header” subform, ensure that the “Content:” drop down list in the “Subform” tab of the “Object” palette is set to “Positioned”. NOTE: This is an important step. If the header subform is not “Positioned”, then the fields on subsequent pages are overlaid and not positioned properly.

5. For the “Header” subform, ensure that the “Allow Page Breaks within Content” checkbox in the “Subform” tab of the “Object” palette is turned off. NOTE: This is an important step. For simple one line headers, this may not have impact. However, if the header subform contains other subforms which may overflow onto another page, it is important to turn off the “Allow Page Breaks within Content” setting to avoid creating duplicate headers.
6. Add the data fields shown in the picture above to the DATA subform. Note that there are 6 fields in the data subform and 7 in the header subform.

7. For the “Summary” subform (the subform that will appear at the end of the rendered output), make sure that it appears in the hierarchy right after the main body of the table (the “DATA” subform). The reason for doing this is to ensure that the subform will appear at the end (after the multiple instances of the “DATA” subform have been rendered).

8. For the “Footer” subform, ensure that the “Repeat Subform for Each Data Item/Min Count” is set to ‘1’. This field is in the “Binding” tab of the “Object” palette. Also make sure that the “Content:” drop down list in the “Subform” tab of the “Object” palette is set to “Positioned”.

The rendered form will now contain the “Header” and “Footer” subforms at the top and bottom of each page. The “Summary” subform will appear on the last page.

Important Notes:

A. The overflow leader and trailer have to be set in the parent subform of the leader/trailer, not in the affected subform. Although the drop down list for overflow leader and trailer is available and will work in some situations, it is strongly recommended to set the overflow leader/trailer using the parent subform.

B. The header subform must always be set up as a “Positioned” content subform and the “Allow Page Breaks” option should be off. If these settings are not correct, the rendered output will contain errors such as duplicate headers or overlaid fields.
Printing Duplex Forms

This chapter explains how to generate duplex (i.e. two sided) forms.

**Note:** See the attached files for the complete solution used in this example:

**Note:** This feature is only available in Designer 8/Adobe Reader 8.

Creating a duplex form

You can render an existing form by setting the `<pagination>` element on the XFA.XCI file on the server.

Traditionally, the only option for designing master pages was to use ordered occurrences. This meant that master pages were laid out in sequence. Repetition was controlled by setting the min/max value of the pages.

![Diagram of Master Page and Pagination settings](image)

A sequence of pages (AAA, AABBAA, etc.) could be created using this property. Pages could be grouped in sets which can be combined in a hierarchy.
In order to create duplex forms, a new method of designing master pages must be used. This new method is called “explicit pagination”.

To begin using “explicit pagination”, select the “(Master Pages)” root element in the hierarchy and then select the “Page Set” tab in the “Object” palette. Change the “Pagination:” drop down list to “Print on Both Sides”. When you do, the following dialog will appear:

Once you click on “Yes”, the template cannot revert back to using ordered occurrences for the master pages.

To define a set of master pages within a page set, set the attributes for odd/even and placement. These options give you the choice of defining the first page, last page, remaining page(s) as well as which master page is used on the front (odd) and back (even).

**Note:** If you have more than one page set, you must explicitly move to a new master page using either a page break or conditional page break.
Scripting Overview

As part of the form design process, a form developer can use calculations and scripts to provide a richer user experience. You can add calculations and scripts to most form fields and objects. For example, the following JavaScript script multiplies the values of two numeric fields together and displays the result in a third numeric field:

\[
\text{NumericField3.rawValue} = \text{NumericField1.rawValue} \times \text{NumericField2.rawValue};
\]

At a more advanced level, you can create your own functions tailored towards your own custom form processing needs.

Designer ES2 supports two scripting languages, each geared towards the needs of a particular type of form developer. FormCalc is a straightforward, easy-to-use calculation language that is modelled on common spreadsheet functionality. It includes a variety of built-in functions designed to reduce the amount of time you need to spend developing your form design. JavaScript, a powerful scripting language, provides you with a great deal of flexibility when creating your scripts and allows you to leverage any existing knowledge of the language.

Remember that scripting on a form is entirely optional. You can choose to take advantage of scripting to provide a richer user experience, but many of the most powerful features available during form creation are available in Designer ES2 without the use of scripts. However, through scripting, you can manipulate and control almost all aspects of your form design.

Note: You can also use the Action Builder dialog box on the Tools menu to build common interactive capabilities in forms that have a flowable layout, without writing scripts.

How scripting works

Designer ES2 scripting uses an event-based model that allows you to alter various aspects of objects on a form at run time. As a form designer, you add scripts to objects based on when you want the script to execute. For example, you might place the following script on the click event of a button object so that at run time, when a user clicks the button, a message box appears with a message:

\[
\text{xfa.host.messageBox("This is a message for a form filler.", "User Feedback", 3);}\]

Scripts associated with a particular event execute whenever that event occurs. Some events can occur multiple times within the same form filling session. For example, the following script adds one to the current value of a numeric field:

\[
\text{NumericField1.rawValue} = \text{NumericField1.rawValue} + 1;
\]

If you add this script to the calculate event for NumericField1, when you open the form for the first time, NumericField1 displays the value 2. This indicates that the calculate event occurred twice in the sequence of events that occurred when the form was opened.

Objects that support calculations and scripts

The following table provides a quick reference of scripting support for the standard objects that are included in the Library palette in Designer ES2.

<table>
<thead>
<tr>
<th>Objects that support calculations and scripts</th>
<th>Objects that do not support calculations and scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barcodes</td>
<td>Circle</td>
</tr>
<tr>
<td>Button</td>
<td>Content Area</td>
</tr>
<tr>
<td>Check Box</td>
<td>Line</td>
</tr>
<tr>
<td>Date/Time Field</td>
<td>Rectangle</td>
</tr>
<tr>
<td>Decimal Field</td>
<td>Image</td>
</tr>
<tr>
<td>Signature Field</td>
<td>Subform Sets</td>
</tr>
<tr>
<td>Drop-Down List</td>
<td>Table sections</td>
</tr>
<tr>
<td>Objects that support calculations and scripts</td>
<td>Objects that do not support calculations and scripts</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>Email Submit Button</td>
<td>Text</td>
</tr>
<tr>
<td>HTTP Submit Button</td>
<td></td>
</tr>
<tr>
<td>Image Field</td>
<td></td>
</tr>
<tr>
<td>List Box</td>
<td></td>
</tr>
<tr>
<td>Numeric Field</td>
<td></td>
</tr>
<tr>
<td>Paper Forms Barcode</td>
<td></td>
</tr>
<tr>
<td>Password Field</td>
<td></td>
</tr>
<tr>
<td>Print Button</td>
<td></td>
</tr>
<tr>
<td>Radio Button</td>
<td></td>
</tr>
<tr>
<td>Reset Button</td>
<td></td>
</tr>
<tr>
<td>Subform</td>
<td></td>
</tr>
<tr>
<td>Table (including body rows, header rows, and footer rows)</td>
<td></td>
</tr>
<tr>
<td>Text Field</td>
<td></td>
</tr>
</tbody>
</table>

**Understanding relationships between objects in the Object Library**

When you create calculations and scripts in Designer ES2, you should be aware that the objects on which you are adding scripts are actually defined as XML objects in the underlying XML Forms Architecture. That means while the Standard tab of the Object Library palette contains a wide variety of objects, many of those objects are defined by the same XML object. As a result, the various scripting properties and methods that are available are based on the definition of the XML object, and not the object in the Object Library palette.

 Objects available in the Standard tab of the Object Library palette that are based on the same base XML object definition share a set of common properties and methods.

**Script Editor**

The Script Editor is where you create, modify, and view the calculations and scripts of a particular form. For example, you can use the Script Editor to write a simple calculation that adds two numeric fields or complex scripts that alter the appearance of the form based on end-user actions. Designer ES2 supports scripting either in its own scripting language called FormCalc or in JavaScript.
By default, the Script Editor appears at the top of the Designer ES2 workspace, but you can dock it anywhere. It has both a single-line view and a multiline view that you can switch between, depending on your needs. Single-line view is designed to maximize the amount of space dedicated to the Layout Editor and other palettes. Multiline view is designed to maximize the amount of space for writing script.

**Single-line view:** A. Script editing field

**Multiline view:**

A. Reference syntax  
B. Show Events for Child Object button  
C. Functions button  
D. Check Script Syntax button  
E. Script editing field

**Show**  
Lists all form design events that support user-defined scripting. Any events that do not apply to a particular object appear dimmed. Events that contain a calculation or script display an asterisk (*) beside the name of the event.

**Show Events for Child Objects**  
Displays the event you have currently selected in the Show list for the current object and all of its child objects. If you select the uppermost object in the Hierarchy palette, this option displays the event you have currently selected in the Show list for all objects on your form.

**Functions**  
Displays a list of available built-in FormCalc or JavaScript functions, depending on the scripting language you currently have selected in the Language list.

To place a function onto your script editing field, select a function from the list and press Enter.

**Check Script Syntax**  
Checks all of the scripts in a form for correct syntax and reports any errors on the Warnings tab in the Report palette.

**Language**  
Specifies the scripting language you want to use for the current calculation or script. Two options are available:

- **FormCalc**  
  FormCalc is a native Adobe calculation language typically used for shorter scripts, such as simple calculations. FormCalc is the default scripting language for new forms.

- **JavaScript**  
  JavaScript is a robust and flexible scripting language suitable for more complex scripts.

The scripting language that is displayed in the Language list matches the scripting language option you select as the default for new forms in the Workspace panel in the Options dialog box. However, if you change the scripting language setting for the current form on the Defaults tab in the Form Properties dialog box, the scripting language that is displayed in the Language list changes similarly for any new scripts on new events. Changing the scripting language option in the Form Properties dialog box does not change the scripting language for existing scripts. If an event already contains script and that script is deleted, the Script Editor continues to use that same scripting language for the duration of your Designer ES2 working session.

**Run At**  
Specifies where the calculation or script will execute. Three options are available:

- **Client**  
  Calculations and scripts execute while the client application (for example, Acrobat, Adobe Reader, or a web browser) processes the form.

- **Server**  
  Calculations and scripts execute while the server application (for example, Forms ES2) processes the form.
• **Client and server** Calculations and scripts execute while the server application (for example, Forms ES2) processes the form, except in cases where the HTML client application supports client-side scripting. For example, a script that accesses a database to prefill data on a form.

**Run At** Specifies where the calculation or script will execute. Three options are available:

• **Client** Calculations and scripts execute while the client application (for example, Acrobat or web browser) processes the form.

• **Server** Calculations and scripts execute while the server application (for example, Adobe document services) processes the form.

• **Client and server** Calculations and scripts execute while the server application (for example, Adobe document services) processes the form. For example, a script that accesses a database to pre-fill data on a form.

**Enable Event Propagation** Specifies that form events propagate to ancestor containers.

---

**Events**

Every calculation or script you attach to a form object is associated with a specific event. An *event* is defined as a particular occurrence or action that can change the state of a form and, when the change of state occurs, automatically invoke a calculation or script associated with the event. Events occur at various times, from the beginning of the form rendering process when merging data with a form design, all the way through to a form filler interacting with objects on a form in a client application. By applying calculations and scripts to specific events, you can control every aspect of how you present form objects, as well as form data, and how the objects and data respond to form filler interaction.

A single change of state or form filler action may trigger multiple events. For example, tabbing from the current field to the next field triggers both the *exit* event for the current field and the *enter* event for the next field. If the current and next fields are in different subforms, a total of four events are triggered; namely, *exit* events for the current field and subform, and *enter* events for the next field and subform. In general, each of the different categories of form events follow a predictable ordering.

**Types of events**

Form events fall into one of the following categories:

**Process events** This type of event initiates automatically as the result of an internal process or action related to objects on a form. For example, if a form filler clicks a button that adds a new page to the form, the *initialize*, *calculate*, *validate*, and *layout:ready* process events initiate automatically for the new page.

**Interactive events** This type of event initiates as a direct result of form filler actions. For example, if a form filler moves the pointer over a field on a form, the *mouseEnter* event initiates in response to the action.

**Application events** This type of event initiates as a result of the actions that either a client application or a server application performs. For example, you can create a calculation or script to perform a task immediately after the form is saved by using the *postPrint* event.

**Process events**

Process events initiate automatically as the result of an internal process or action related to a form or objects on a form. These events initiate immediately following significant form changes; for example, after a form design is merged with data or after the form pagination process finishes. Process events also initiate immediately after interactive events initiate. For example, immediately after any interactive event initiates, the *calculate* event initiates followed by the *validate* event.

The following list contains the process events, which are available from the Show list in the Script Editor:

• *calculate*
• *form:ready*
• *indexChange*
• *initialize*
Process events can initiate many times as a result of dependencies; that is, actions associated with a single event that ultimately initiates one or more additional events. Using an example of a form filler clicking a button to reveal a previously hidden portion of the form, after the form filler clicks the button, not only does a series of interactive and processing events initiate for the button itself, but a number of process events for the new subform initiates as well.

The following image represents the general flow of events leading up to a PDF form opening in Acrobat or Adobe Reader.

After the form opens in Acrobat or Adobe Reader, these process events may still initiate as the result of changes made to the form. For example, the calculate, validate, and layout:ready events for an object initiate immediately after some interactive events occur; therefore, calculations and scripts attached to the processing events will run multiple times.

Interactive events

Interactive events initiate as a direct result of form filler actions, which makes these events useful for a variety of calculation and scripting tasks. For example, you can add a script to the mouseEnter event for a text field that changes the border color of the field to blue and a script to the mouseExit event that changes the border color back to the original color. This action creates a highlighting effect when form fillers move the pointer over the field to visually assist them while filling the form. Interactive events are also useful for changing form data in response to a form filler selection. For example, you can add a script to the change event for a drop-down list that updates the data values in multiple fields in response to the value the form filler selects in the drop-down list.

The following list contains the interactive events, which are available from the Show list in the Script Editor:

- change
The following image displays the general flow of events for form fillers who use the mouse to select an object and change its value.
Note: This image provides a general flow of events; however, certain form filler actions and form objects can cause alternate event ordering. For example, if a form filler selects a value from a drop-down list, the `mouseExit` event occurs after the `click` event but before the `change` or `full` events. Similarly, if a form filler selects a field, holds down the mouse button, and then exits the field while still holding down the mouse button, the `mouseUp` event occurs out of the order described in this image.

1 The `calculate` and `validate` events trigger after each interactive event to update any calculations associated with the field, and to verify the field value.
The following image displays the general flow of events for form fillers who use the keyboard to select an object and change its value.

Application events
Application events initiate as a result of the actions that a client application or a server application perform, either due to a form filler action or an automated process. Application events do not exist as part of a general flow of events. They are single events that correspond to actions that the client or server application performs.

The following list contains the processing events, which are available from the Show list in the Script Editor:

- docClose
- docReady
- postPrint
- postSave
postSubmit
prePrint
preSave
preSubmit

For example, the following image displays the general flow of events for the preSave event.

If a form filler saves the form in Acrobat or Adobe Reader, the preSave event initiates immediately before the save operation, followed by the calculate, validate, and layout:ready events, in that sequence for all objects on the form. The same event sequence initiates if the form contains a script that programmatically saves the form.

A similar sequence of events occurs for each of the other application events previously listed.

**calculate event**

**Description**

Initiates in the following situations:

- When your form design and data merge into your finished form.
• When a change occurs to any value that the calculation is dependent on, such as the value of a particular field, unless the form filler has manually overridden the calculated value. As a result, the object will display the return value of the event. The properties for manually overridden fields are located in the Value tab of the Object palette.
• When a field loses focus; for example, when a form filler clicks or uses the Tab key to exit a field.

When using the calculate event to perform calculations or scripts, consider the following potential issues:

• Calculations and scripts on the calculate event must not make any changes to the structure of the form, except for the form field and data values.
• Content inserted by the calculate event must conform to the associated validations for the object; otherwise, validation errors will occur.
• Calculations and scripts must not include an infinite loop because it causes the form to update the value continuously. For example, a script that increments the value of a field as part of a looping expression, such as a while or for loop, could create an infinite loop.
• The last expression evaluated in the calculate event is used to populate the value of the current form object. For example, if the script on the calculate event first sets the value of the current field to 500 and then sets the value of another field to 1000, both fields will display the value 1000 at run time. As a result, you need to limit the scripting that you add to the calculate event to those that deal specifically with setting the value of the current field.

Type
Processing event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

Version
XFA 2.1

Example
Use the calculate event for updating numeric values in fields because this event initiates immediately after most other events. For example, on a purchase order form, you can use the calculate event for a field to determine the percentage of sales tax due based on the cost of the order. The calculation will initiate every time a change is made to the values in the form fields, ensuring that the value displayed for the sales tax is always correct.

However, because the calculate event can initiate many times, you must ensure that the calculation or script you add to the event will not cause data values to increment unnecessarily. For example, if your sales tax calculation adds the value of the sales tax to the total cost each time the calculate event initiates, the resulting total cost value on your form may be too large.

change event

Description
Initiates when a form filler changes the content of a field by performing one of these actions:

• Types a keystroke providing the field has keyboard focus
• Pastes data into the field
• Makes a selection from a list box or drop-down list
• Selects or deselects a check box
• Changes the setting of a group of radio buttons

This event does not initiate in response to changes in the object values as a result of calculations or scripts, or by the merging of the form
design with data.

Type
Interactive event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

(Only for drop-down lists)

Version
XFA 2.1

Example
Use this event for any calculations or scripts that must initiate in response to a form filler changing the value of a field. For example, you can
use the change event for a drop-down list to highlight specific rows in a table. Using this technique, each time the form filler selects a value
in the drop-down list, the corresponding row of the table appears highlighted.

Note: Scripting against an object’s ‘this.rawValue’ does not work. Use the event model property $event.fullText instead to get the
object’s current value.

click event

Description
Initiates when a mouse click occurs within the region. When a click event initiates for a text or numeric field, calculations or scripts execute
immediately. However, the value of the field does not change in response to calculations and scripts until the field loses focus.

Note: You cannot place a calculation or script on the click event of a submit button because the calculation or script will override the
submission action. Instead, place any calculations and scripts on the preSubmit event for a submit button. For more information about form
submission actions, see the Designer ES2 Help.

Type
Interactive event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

Version
XFA 2.1
Example
Use this event for performing an action as a direct response to a form filler clicking a button or selecting a radio button or check box on a form. For example, you can use the click event for a check box to hide and show a field on the form.

docClose event

Description
Initiates at the very end of processing a form, only if all form validations complete with no errors.

Type
Application event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

Version
XFA 2.1

Example
This event initiates too late to modify a saved form and is intended to provide the ability to generate an exit status or completion message. For example, you can use the docClose event to display a message to a form filler indicating that the form is completed.

docReady event

Description
Initiates immediately after the form opens in Acrobat or Adobe Reader.

Type
Application event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

Version
XFA 2.1

Example
This event is the first one that initiates after the form opens in Acrobat or Adobe Reader. Any calculation or scripting tasks that require the full form, or that should only run once when the form filler first opens the form, should use this event. For example, you can use the docReady event to check the version of Acrobat or Adobe Reader and return a message to the form filler if the form filler must upgrade the application before filling the form.
enter event

Description
Initiates when a field or subform gains keyboard focus, whether caused by a form filler action (tabbing into a field or clicking in it) or by a script programmatically setting the focus.

Type
Interactive event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

Version
XFA 2.1

Example
You can use this event to provide help text or other messages to a form filler while entering the current field or subform. For example, if a field requires a value in a specific format, or if filling a field requires special instructions, you can use this event to provide a message to the form filler indicating the special needs.

exit event

Description
Initiates when the field or subform loses keyboard focus, whether caused by a form filler action (tabbing to another field or clicking outside it) or by a script programmatically removing the focus.

Note: If the purpose of your script is to manipulate the value of the current field, you need to consider attaching your script to the calculate event.

Type
Interactive event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

Version
XFA 2.1

Example
You can use this event to provide verification of field data as a form filler moves the focus away from a field. For example, if a field requires a value, you can use this event to provide a message to the form filler indicating that the field requires some data before the form can be submitted.
**form:ready event**

**Description**
Initiates after the form design and data are merged, the finished form exists in memory, and the `initialize`, `calculate`, and `validate` events are complete.

*Note: The form:ready event only applies to Design View objects, and does not apply to Master Page objects (see ?$paratext?> on page 65).*

**Type**
Processing event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

**Version**
XFA 2.1

**Example**
You can use this event to perform tasks after the form design and data are merged but before the layout is established. For example, you can use this event to customize the ordering or placement of subforms on your form before the form is paginated and rendered.

**full event**

**Description**
Initiates when the form filler attempts to enter more than the maximum allowed amount of content into a field. For example, if the Limit Length property for a field is set to 5, and a form filler attempts to enter the string `abcdef`, the full event initiates when the form filler types the letter f.

*Note: The Limit Length property for a field is located in the Field tab in the Object palette.*

**Type**
Interactive event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

**Version**
XFA 2.1

**Example**
Use this event to indicate to a form filler that a field has reached its maximum capacity. For example, you can output a message to the form filler indicating that the field is full and provide any steps that should be taken to correct the issue.
indexChange event

Description
Initiates as a result of a subform being inserted, moved, or removed from the form by merging new data with the form or by using scripting.

Keep in mind that the indexChange event does not fire when deleting the last row of a table.

Note: This event is received only by the subform instances that are controlled by the instance manager; the event is ignored for subform sets.

Type
Processing event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

Version
XFA 2.5

Example
You can use this event to set properties based on the instance value of a particular object. For example, you can use this event to coordinate the shading of alternate rows in a table.

initialize event

Description
Initiates for all objects after the form design is merged with data.

Type
Processing event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

Version
XFA 2.1

Example
You can use this event to perform actions when an object is first created, either as the result of a form filler action or during the form creation process. For example, you can use this event to control settings for new instances of a subform object that a form filler adds to the form by using a button.
layout:ready event

Description
Initiates after the form design and data are merged, the form exists, and the form’s layout is applied. At this time, the finished form has not been rendered; therefore, a calculation or script set to run on this event could modify the layout before the form is rendered. This event also occurs after the form is rendered if a calculation or script changes the data or causes a change to the form in Acrobat or Adobe Reader.

*Note:* Scripts that fire on layout:ready should not do anything that would cause the layout of the form to change. For example, this would include anything involving subforms or tables that grow or shrink, adding fragments dynamically at run time, adding or removing subform instances, and toggling the presence setting of an object between hidden and visible.

Fields in interactive forms that contain the layout:ready event are supported in Acrobat 7.0.5 and later.

Type
Processing event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

Version
XFA 2.1

Example
You can use this event to perform tasks immediately after the form layout is established. For example, you can use this event to determine the number of pages the form contains.

mouseDown event

Description
Initiates when a form filler presses the mouse button at the same time that the pointer is within a field.

*Note:* When a mouseDown event initiates for a text or numeric field, calculations or scripts run immediately. However, the value of the field does not change in response to calculations and scripts until the field loses focus. When a mouseDown event initiates for a signature field, the event initiates before the signature process begins.

Type
Interactive event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

Version
XFA 2.1
Example
You can use this event to perform an action as a direct response to a form filler clicking a button, or selecting a radio button or check box on a form. For example, you can use the `mouseDown` event for a check box to hide and show a field on the form. This event is conceptually similar to the `click` event and has a similar purpose.

**mouseEnter event**

**Description**
Initiates when the form filler moves the pointer into the area of the field, without necessarily pressing the mouse button. This event is not initiated when the pointer moves into the field for a different reason; for example, because an overlapping window closes.

**Type**
Interactive event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

**Version**
XFA 2.1

Example
You can use this event to provide visual feedback to a form filler in conjunction with the `mouseExit` event. For example, you can use this event to change the border or background color of an object to help visually indicate to form fillers that they are working in a specific field.

**mouseExit event**

**Description**
Initiates when a form filler moves the pointer out of the field, even if the form filler is pressing the mouse button. It is not initiated when the pointer moves out of the field for a different reason; for example, because an overlapping window opens.

**Type**
Interactive event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

**Version**
XFA 2.1
Example
You can use this event to provide visual feedback to a form filler in conjunction with the `mouseEnter` event. For example, you can use this event to return the border or background color of an object to its original value to help visually indicate to form fillers that they are no longer working in a specific field.

**mouseUp event**

**Description**
Initiates when a form filler releases the mouse button at the same time that the pointer is within a field.

*Note: When a `mouseUp` event occurs for a text or numeric field, calculations or scripts run immediately. However, the value of the field does not change in response to calculations and scripts until the field loses focus.*

**Type**
Interactive event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Version**
XFA 2.1

**Example**
You can use this event to perform actions as a direct response to a form filler clicking a button, or selecting a radio button or check box on a form. For example, you can use the `mouseUp` event for a check box to hide and show a field on the form. This event is conceptually similar to the `click` event and has a similar purpose.

**postOpen event**

**Description**
Initiates immediately after a form filler performs an action that causes the data in a drop-down list to appear, such as clicking the arrow icon on the drop-down list or tabbing into the drop-down list and then using the down arrow. This event initiates after the contents of the drop-down list are displayed.

*Note: This event applies only to the Drop-down List object.*

**Type**
Interactive event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>
Version
XFA 2.8

Example
You can use this event to handle errors or unexpected outcomes as a result of processing the opening of the drop-down list. For example, if the `preOpen` event is dispatched via scripting instead of user interaction, or if the opening of the drop-down list data does not occur as a result of an error, the `postOpen` event is still dispatched to let error handling scripts execute.

postPrint event

Description
Initiates immediately after the rendered form is sent to the printer, spooler, or output destination.

Type
Application event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

Version
XFA 2.1

Example
You can use this event to display information messages to the form filler after the form is printed. For example, you can create a script on the `postPrint` event to remind form fillers what steps they need to take to submit the form by hand.

postSave event

Description
Initiates immediately after a form filler saves a form in PDF or XDP format. This event does not initiate when you export a subset of the form (for example, only form data) to XDP.

Type
Application event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

Version
XFA 2.1
Example
You can use this event to display information messages to the form filler after the form data is saved. For example, you can create a script on the postSave event to remind form fillers how much time remains for them to successfully complete and submit the form.

postSign event

Description
Initiates immediately after a form filler performs an action that applies a digital signature to a form.

Note: This event applies only to the Signature Field object.

Type
Interactive event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

Version
XFA 2.8

Example
You can use this event to inform a user about any restrictions that are imposed now that the form is digitally signed.

postSubmit event

Description
Initiates immediately after a form submits data to the host through the HTTP protocol.

Note: This event does not distinguish between submissions that are initiated by instances of clicking buttons, or submissions made to different URLs. Any script that needs to make these distinctions must include a script to determine which button was clicked. In general, the postSubmit event is conceptually similar to the postSave event and serves a similar purpose.

Type
Application event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

(Only for submit buttons)
Example
You can use this event to perform actions immediately after the form data is submitted. For example, you can create a script on the `postSubmit` event to display confirmation that the submission performed successfully.

**preOpen event**

**Description**
Initiates when a form filler performs an action that causes the drop-down list to appear, such as clicking the arrow icon on the drop-down list or by tabbing into the drop-down list and using the down arrow. This event initiates before the contents of the drop-down list are displayed.

*Note: This event applies only to the Drop-down List object.*

**Type**
Interactive event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

**Version**
XFA 2.4

**Example**
You can use this event to control the loading of large numbers of list items. For example, you can use this event to load a fixed number of records from a data source into a drop-down list. This improves the performance of the form for the form filler at run time.

**prePrint event**

**Description**
Initiates immediately before the process of rendering a form for printing begins. You cannot cancel printing using this event.

*Caution: Avoid using this event to hide or show form objects. For example, if a form filler has already digitally signed the form, using this event to hide all button objects prior to printing will impact the state of the signature.*

**Type**
Application event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Version**
XFA 2.1
Example
You can use this event to change the presence of an object to prevent it from printing. For example, you can use this event to hide text or instructions intended for the form filler to use while filling the form online.

**preSave event**

**Description**
Initiates immediately before form data is saved in PDF or XDP format. This event does not initiate when the form data or another subset of the form is exported to XDP.

**Type**
Application event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes</td>
</tr>
</tbody>
</table>

**Version**
XFA 2.1

Example
You can use this event to change form data immediately before the data is saved. For example, you can create a script on the `preSave` event to scan the data and display a reminder message to the form filler if certain required fields remain empty.

**preSign event**

**Description**
Initiates immediately before a form filler performs an action that applies a digital signature to a form.

*Note: This event applies only to the Signature Field object.*

**Type**
Interactive event

**Support**

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>no</td>
</tr>
</tbody>
</table>

**Version**
XFA 2.8

Example
You can use this event to validate the data that the digital signature covers or to provide any information to a user before they apply the digital signature.
preSubmit event

Description
Initiates when a form submits data to the host through the HTTP protocol. At this point, the data is organized into a data set but has not been sent to the host. Calculations and scripts associated with this event can examine and alter the data prior to the form submission. If the calculation or script is set to run on the server, the form sends the data to the server indicating that it should run the calculation or script before performing any additional processing.

Note: This event does not distinguish between submissions initiated by instances of clicking buttons or to different URLs. Any script that needs to make these distinctions must include code to determine which button was clicked. In general, the preSubmit event is conceptually similar to the preSave event and serves a similar purpose.

Type
Application event

Support

<table>
<thead>
<tr>
<th>Client application</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acrobat and Adobe Reader</td>
<td>yes</td>
</tr>
<tr>
<td>HTML browser</td>
<td>yes (Only for submit buttons)</td>
</tr>
</tbody>
</table>

Version
XFA 2.1

Example
You can use this event to change form data immediately before the data is submitted. For example, you can create a script on the preSubmit event to scan the amount of data and display a message to the form filler estimating how long the data submission may take.

validate event

Description
Initiates when the form design and data merge to create your form and when a field loses focus; for example, when a form filler clicks or uses the Tab key to exit a field. This event initiates again each time the value of a field changes. Calculations and scripts placed on the validate event provide a method to perform validations that are more specific than those available through the Value tab of the Object palette.

Calculations and scripts on the validate event are required to return true or false (expressed in a format appropriate to the scripting language) corresponding to a validation that succeeds or fails, and must not affect the overall form structure of form values. In addition, calculations and scripts should not attempt to provide feedback to a form filler because that form filler may not be using the form in a client application such as Acrobat.

Note: Because validations are performed against the content of the form, they cannot be used to verify presentation formatting caused by field patterns.

Type
Processing event
Creating Calculations and Scripts

Designer ES2 provides a wide range of calculation and scripting features that you can use to perform a variety of tasks. For example, the following script changes the color of a text field border and the font size of the text field value:

```
TextField1.border.edge.color.value = "255,0,0";
TextField1.font.typeface = "Courier New";
```

More complex forms can take advantage of scripting to perform data source connectivity and data manipulation at run time. Creating calculations and scripts in Designer ES2 involves following a general process each time you attach a calculation or script to an object. Although not all aspects of the process are required each time you create a calculation or script, following the process helps to eliminate potential errors and unexpected results.

In general, each time you create a calculation or script, you perform the following tasks:

- Select the object to which you want to attach a calculation or script. Although you can create calculations and scripts that manipulate almost any object on your form design, not all form design objects support form events. For a list of standard objects included in the Object Library palette in Designer ES2 that support scripting.
- Write your calculation or script in the script editing field of the Script Editor.
- Test the calculation or script either by using the Preview PDF tab or in your test environment.

Making scripts global by using event propagation

You can make scripts global by enabling event propagation in the Script Editor. The setting allows form events to propagate to ancestor containers. Event propagation can reduce the number of scripts in a form. For example, you can create a global script to control the appearance of invalid fields, subforms, or exclusion groups. Here are examples of global events:

- An enter/exit/mouseEnter/mouseExit event that colors the active field
- A change event that track keystrokes for a form session

To enable event propagation

In the Script Editor, select Enable Event Propagation.

**Note:** Select the object and write the script at the ancestor level before enabling event propagation.
Naming conventions for form design objects and variables

When creating calculations or scripts to enhance your form, be aware of the form design object and variable names on your form. In general, avoid using the names of XML Form Object Model properties, methods, and objects for form design objects and variables. Using XML Form Object Model property, method, or object names can result in calculations and scripts not executing properly.

For example, if you create a new text field named x within a subform object named Subform1, you access the text field object using the following syntax:

```
Subform1.x.[expression]
```

However, subform objects already have an XML Form Object Model property named x that represents the horizontal position of the subform on the form design.

To avoid naming conflicts, you need to choose field naming conventions that differ from the XML Form Object Model naming conventions. For example, you can use any of the following field names for the text field in the example above:

- horizontalValue
- x_value
- xLetter
- hValue

Choosing a scripting language

Designer ES2 supports scripting with both FormCalc and JavaScript. Each scripting language presents its advantages that you should be aware of before you write any scripts on your form.

FormCalc is a calculation language that includes a wide range of built-in functions to simplify the most common form functionality. For example, you can use FormCalc financial functions to evaluate the size of a loan payment based on the principle amount, interest rate, and number of payment periods.

JavaScript is a more powerful and diverse scripting language, intended to give you more flexibility and leverage your existing scripting knowledge. For example, you can reuse your existing JavaScript functions in Designer ES2 to reduce the amount of new scripting you need to create.

Note: Designer ES2 supports JavaScript version 1.6 or earlier.

You can choose the scripting language that is used for new forms in the Workspace panel in the Options dialog box, and for the current form on the Defaults tab in the Form Properties dialog box.

The scripting language that is displayed in the Language list in the Script Editor matches the scripting language option you select as the default for new forms. However, if you change the scripting language setting for the current form, the scripting language that is displayed in the Language list changes similarly for new scripts on new events. Changing the scripting language option in the Form Properties dialog box does not change the scripting language that was used for existing scripts. If an event already contains some script and that script is deleted, the Script Editor continues to use that scripting language for the duration of your Designer ES2 working session.
The following table highlights some of the key differences between FormCalc and JavaScript.

<table>
<thead>
<tr>
<th>FormCalc</th>
<th>JavaScript</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native Adobe calculation language valid in Designer ES2 and Forms ES2</td>
<td>Standard scripting language used in many popular software applications</td>
</tr>
<tr>
<td>Shorter scripts (typically one line only)</td>
<td>Potential for longer scripts, if necessary, with the ability to use looping</td>
</tr>
<tr>
<td>Supports script looping</td>
<td></td>
</tr>
<tr>
<td>Not supported by form guides</td>
<td>Supported by form guides</td>
</tr>
<tr>
<td>Contains a variety of useful built-in functions to reduce the amount of</td>
<td>Provides access to the Acrobat Object Model and the JavaScript capabilities</td>
</tr>
<tr>
<td>scripting required to accomplish common form design tasks</td>
<td>of Acrobat</td>
</tr>
<tr>
<td>Support for international dates, times, currencies, and number formats</td>
<td>Debugging possible by using the JavaScript debugger in Acrobat</td>
</tr>
<tr>
<td>Built-in URL functions for Post, Put, and Get allow web-based interactions</td>
<td>Create custom functions for your own specific needs</td>
</tr>
<tr>
<td>Compatible on all Designer ES2 and Forms ES2 supported platforms</td>
<td>Compatible on all Designer ES2 and Forms ES2 supported platforms</td>
</tr>
</tbody>
</table>

**To create a calculation or script**

1. Select an object on your form design that supports events. For example, add a button to a new, blank form.

2. In the Script Editor, from the Show list, select one of the events that apply to the object. The event you choose specifies when the script will execute. If you are writing a calculation or script that affects an object that does not support events, you must add your calculation or script to a form design object that does support form events. For example, using the new button object, select the `click` event in the Show list.

3. In the Language list, select your scripting language. For example, for the new button object, select JavaScript.

4. In the Run At list, select where you want the script to execute. For example, for the new button object, select Client.

You can choose to run calculations or scripts on your client-based application (for example Acrobat or a web browser) or on your server-based process (for example, Forms ES2).

When set to Client, processing of calculations and scripts initiates after the form renders. When set to Server, processing of calculations and scripts initiates during the form rendering process. Previewing your form by using the Preview PDF tab simulates opening the form in Acrobat; therefore, scripts set to run at Client or Client and Server execute.
Note: Selecting Client And Server from the Run At list causes a script to execute in either the client application or the server application, depending on which application is used to process the form.

5 In the Script Source field, insert your FormCalc calculation or JavaScript script. You can take advantage of the statement completion functionality of Designer ES2 to help you create reference syntaxes for your calculation or script. For example, add the following JavaScript script to the new button object:

```
xfa.host.messageBox("Hello World!", "Creating a new script", 3);
```

6 After you complete your form design, test and debug your calculations and scripts before putting them into production. For example, for the new button object, preview the PDF of the form using the Preview PDF tab. Click the button object to display the message box specified in step 5.

To create a calculation or script

1 Select an object on your form design that supports events. For example, add a button to a new, blank form.

2 In the Script Editor, from the Show list, select one of the events that apply to the object. The event you choose specifies when the script will execute. If you are writing a calculation or script that affects an object that does not support events, you must add your calculation or script to a form design object that does support form events. For example, using the new button object, select the click event in the Show list.

3 In the Language list, select your scripting language. For example, for the new button object, select JavaScript.

4 In the Run At list, select where you want the script to execute. For example, for the new button object, select Client.

You can choose to run calculations or scripts on your client-based application (for example Acrobat or a web browser) or on your server-based process (for example, Adobe document services).

When set to Client, processing of calculations and scripts initiates after the form renders. When set to Server, processing of calculations and scripts initiates during the form rendering process. Previewing your form by using the Preview PDF tab simulates opening the form in Acrobat; therefore, scripts set to run at Client or Client and Server execute.

Note: Selecting Client And Server from the Run At list causes a script to execute in either the client application or the server application, depending on which application is used to process the form.

5 In the Script Source field, insert your FormCalc calculation or JavaScript script. You can take advantage of the statement completion functionality of Designer ES2 to help you create reference syntaxes for your calculation or script. For example, add the following JavaScript script to the new button object:

```
xfa.host.messageBox("Hello World!", "Creating a new script", 3);
```

6 After you complete your form design, test and debug your calculations and scripts before putting them into production. For example, for the new button object, preview the PDF of the form using the Preview PDF tab. Click the button object to display the message box specified in step 5.

To find text or other items

You can quickly search for every occurrence of a specific word or phrase when you are in the XML Source tab or in the Script Editor.

1 In the XML Source tab or the Script Editor, select Edit > Find or right-click for the context menu.

2 In the Find What box, enter the text that you want to search for.

3 Select any other options that you want.

4 Click Find Next.

To cancel a search in progress, press Esc or select the Cancel button.

Caution: Although it is possible to edit XML source code directly in the XML Source tab, it is recommended that you do not make any changes unless you are familiar with the Adobe XML Forms Architecture. For more information about the XML Forms Architecture, see www.adobe.com/devnet/livecycle/.
To replace text or other items

You can automatically replace text. For example, you can replace Corp. with Corporation.

1. In the Script Editor, select Edit > Replace.
2. In the Find What box, enter the text that you want to search for.
3. In the Replace With box, enter the replacement text.
4. Select any other options that you want.
5. Click Find Next, Replace, or Replace All.
6. To cancel a search in progress, press Esc or select the Cancel button.

To replace text that appears in scripts attached to multiple objects on your form, select the root subform of your form (by default: form1) and select Show Events for Child Objects and then perform the procedure above.

Caution: Although it is possible to edit XML source code directly in the XML Source tab, it is recommended that you do not make any changes unless you are familiar with the Adobe XML Forms Architecture. For more information about the XML Forms Architecture, see www.adobe.com/devnet/livecycle/.

To use statement completion to create calculations and scripts

The statement completion functionality within the Script Editor lets you build your calculations and scripts interactively.

When writing a calculation or script, each time you enter a period (.) immediately following a form object or property name, the statement completion functionality displays a list of available methods and properties. If the statement completion list does not appear, verify that you have typed the object or property name correctly and that the object is within the scope of the object where you are creating your script. For more information about referencing objects in calculations and scripts, see “Referencing Objects in Calculations and Scripts” on page 69.

1. Type the name of a form design object, property, or a valid FormCalc shortcut, followed by a period.

2. Select the method or property you want to apply for the form design object and continue writing the script. To close the statement completion list without selecting a function, press the Esc key.

The list of available XML Form Object Model properties changes depending on the form design object or property that appears before the period.
Note: The statement completion list appears only when accessing objects, properties, and methods in the XML Form Object Model. It does not appear when working with standard JavaScript objects or methods.

To insert object reference syntax automatically
As an alternative to using the statement completion list to create object reference syntax, you can use the insert object reference syntax feature to automatically add reference syntax to your calculation or script. This feature inputs an abbreviated reference syntax for the object you select from the canvas into the Script Source field of the Script Editor. This reduces the time required to create calculations and scripts and ensures that the reference syntax is accurate.

1. Ensure that the Script Source field of the Script Editor has the focus and the cursor is positioned where you want to insert the object reference.
2. On your form, Ctrl+click the object you want to reference. The cursor changes to \( \checkmark \) to assist you when selecting an object.

Determining when to run your calculation or script
When creating calculations and scripts, you must associate each entry with a specific form event. Each form event represents a change in the form's state that initiates at a specific time.

The change in form state can occur during form rendering on the server by Forms ES2, or on the client by Acrobat or Adobe Reader while a user is filling a form.

The change in form state can occur during form rendering on the server by Adobe document services, or on the client by Acrobat or Adobe Reader during form filling by an end user.

When a change in the state of the form occurs, any calculations or scripts associated with the event are processed automatically.

The event you use when creating a calculation or script will, to some extent, determine what you must include in your calculation or script. For example, the amount and type of information available on a form may be different depending on the event timing you choose; therefore, a calculation or script that retrieves a value from a field may have different results if run before instead of after a form filler performs certain actions.

Depending on the type of form you are creating, some events may never occur. For example, if you are creating a form that has a fixed layout and no interactive objects, then interactive events associated with form filler actions may never occur and, as a result, any scripts associated with those events will not run.

Although Designer ES2 includes support for a wide variety of form events, it is possible to accomplish a wide variety of common calculation and scripting tasks by using only a few events that occur at major changes in a form's state, such as the following events:

- **docReady** Initiates immediately after the form opens in Acrobat or Adobe Reader® and immediately before the form filler can begin interacting with form objects. This event is the last event to occur before control of the form is given to the form filler.
- **enter** Initiates when the form filler changes the focus to a particular field, button, or subform.
- **exit** Initiates when the form filler changes the focus from a particular field, button, or subform, to another object.
- **change** Initiates when a form filler makes a change to a field value. This event is most commonly used with drop-down lists or list boxes to execute a script when a form filler makes a change to the current value.
- **click** Initiates when a form filler clicks a field or button. This event is most commonly used with buttons to execute a script when a form filler clicks the button.

To view scripting events and scripts
The Script Editor provides several ways to view scripting events for objects in your form, depending on the type of object or objects you select, and the quantity of events you want to display.
Before you begin, you must perform the following actions:

- If the Script Editor is not displayed on the screen, select Window > Script Editor.
- If the Script Editor is not large enough to display more than one line of script at a time, drag its lower line down to increase its size.

Before you begin, you must perform the following actions:

- If the Script Editor is not displayed on the screen, select Palettes > Script Editor.
- If the Script Editor is not large enough to display more than one line of script at a time, drag its lower line down to increase its size.

**To view a scripting event for a single object in the Script Editor**

1. Select an object in your form.
2. In the Show list, select a valid scripting event.

**To view a scripting event for a container object and its children in the Script Editor**

1. If it is not already in multiline mode, expand the Script Editor to display multiple lines of script and ensure that the Show Events for Child Objects option is selected.
2. Select a container object, such as a subform.
3. In the Show list, select a valid scripting event.

The events appear in the script editing field of the Script Editor, separated by the reference syntax for each event. Note that certain events only apply to specific types of objects. When you select a script event, the script editing field of the Script Editor only displays valid instances of the event. For example, if you select a subform that contains a drop-down list and select the `preOpen` event, the Script editor displays a single entry representing the drop-down list. This is because the `preOpen` event only applies to drop-down lists. Alternatively, selecting the `enter` event displays two entries, one for the drop-down list and one for the subform.

*Note: The Show list denotes events that contain scripts using a trailing asterisk (*) after the name of the event. If an event contains a script, when you select the event from the Show list, the source appears in the script editing field of the Script Editor.*

**To view all scripting events for a single object in the Script Editor**

1. Select an object in your form.
2. In the Show list, select All Events.

The events appear in the script editing field of the Script Editor, separated by the reference syntax for each event.

**To view all scripting events for a container object and its children in the Script Editor**

1. If it is not already in multiline mode, expand the Script Editor to display multiple lines of script and ensure that the Show Events for Child Objects option is selected.
2. Select a container object, such as a subform.
3. In the Show list, select All Events.

The events appear in the script editing field of the Script Editor, separated by the reference syntax for each event.

**To view all scripts for a single object in the Script Editor**

1. Select an object that has scripts attached.
2. In the Show list, select Events With Scripts.

The scripts appear in the script editing field of the Script Editor, separated by the reference syntax for each event.

**To view all scripts for a container object and its children in the Script Editor**

1. If it is not already in multiline mode, expand the Script Editor to display multiple lines of script and ensure that the Show Events for Child Objects option is selected.
2 Select a container object, such as a subform. All events for the container object and any child objects appear in the Script Editor.

3 In the Show list, select All Events.

The scripts appear in the script editing field of the Script Editor, separated by the reference syntax for each event.

Determining where to run your calculation or script

For each calculation and script created in Designer ES2, you must specify the location where you want the calculation or script to run.

Unless you are using server-based processing such as Forms ES2, you should ensure that all of your calculations and scripts are set to run on the client application (for example, on Acrobat or a web browser).

Unless you are using server-based processing such as Adobe document services, you should ensure that all of your calculations and scripts are set to run on the client application (for example, on Acrobat or a web browser).

Note: FormCalc calculations and scripts do not work on forms rendered as HTML and are ignored during form filling.

If you are using server-based processing, you can choose between running calculations on the client application, or running them on the server. By choosing to have calculations and scripts run on the server, you are choosing to run the scripts at a specific point during the form-rendering process.

If you choose Client And Server from the Run At list, your calculation or script is available to both client and server-based applications. This option is useful, for example, if you do not know whether your users will have client or server applications when they attempt to use your form. It is also useful if you want certain form objects to behave one way to a client application and another to a server-based application.

Testing and debugging calculations and scripts

After you create your calculations or scripts and tested your form design, you may discover scripting errors or unexpected field values as a result of scripting syntax errors.

Designer ES2 includes three primary methods for testing and debugging your calculations and scripts:

- Using the Designer ES2 workspace palettes.
- For JavaScript only, using the JavaScript Debugger to assist you in testing your scripts.
- Using the host model and event model properties and methods to troubleshoot your form.

The host model and event model provide functionality that lets you interact with either the host application or the various form events. These models are useful for returning information that can assist you in debugging calculation and scripts.

For example, the following script returns a message at run time indicating the name of the event on which the script is placed. This indicates that a particular event has fired:

```
xfa.host.messageBox(xfa.event.name) // FormCalc
xfa.host.messageBox(xfa.event.name); // JavaScript
```

Another example of using the host model and event model methods is to obtain the value of a field on an interactive form before a user manually changed it. This is useful for observing how the objects on your form design respond to user-entered data:

```
xfa.host.messageBox(xfa.event.prevText) // FormCalc
xfa.host.messageBox(xfa.event.prevText); // JavaScript
```

To check script syntax

While you work on a form design, you can check all JavaScript or FormCalc scripts for syntax errors to ensure that the form functions as expected before you distribute it for use. Any script syntax errors found in the form are displayed in the Report palette on the Warnings tab.

On the Warnings tab in the Report palette, each error is listed on a separate numbered line, along with the event or script object name and a description of the error. If multiple events are displayed, the line numbering for each event begins at 1.
You can click any script error in the list to display the relevant script, highlight the line that contains the error, and locate the insertion point at the beginning of the highlighted line. Scripting syntax errors are also reported in the Warnings tab when you save a form design or preview it using the Preview PDF tab.

**Note:** You can also use the Go To Line dialog box to select the event you want to see. The script event drop-down list includes the System Object Model (SOM) expression, as shown in the header lines, for each event currently visible in the Script Editor.

- In the Script Editor, select Tools > Check Script Syntax.

### Working around security restrictions

Script that modifies the sourceSet model or its children makes the form’s certification invalid and the form can no longer be trusted. Because a form can become certified at any time during its life cycle, it is important to use scripting techniques that prevent the form from failing after it is certified.

You must work with clones of the model rather than with the model if you intend to use scripts that modify the sourceSet model or any of its children. Cloning prevents the form from failing when scripts modify a data model. For example, forms that execute common tasks, such as displaying records in a database or selecting specific records in a database, require the modification of data connection nodes contained within the sourceSet model.

To clone the sourceSet model, you must create a method on the script that defines the data connection that you want to modify within the sourceSet model and make sure that the script keeps using the clone instead of the definition.

Consider the following script from a data drop-down list. The script populates the list from data from a from a data source.

```javascript
... var oDB = xfa.sourceSet.nodes.item(nIndex);
... // Search node with the class name "command"
var nDBIndex = 0;
while(oDB.nodes.item(nDBIndex).className != "command")
nDBIndex++;
odb.nodes.item(nDBIndex).query.recordSet.setAttribute("stayBOF", "bofAction");
odb.nodes.item(nDBIndex).query.recordSet.setAttribute("stayEOF", "eofAction");
```

To clone the sourceSet model, you need to change the line that accesses it by appending the clone (1) method to the end of the statement:

```javascript
var oDB = xfa.sourceSet.nodes.item(nIndex).clone(1);
```

You can store the cloned data connection node in a variable or a variable defined in a script object.

### Scripting with Form Calc and JavaScript

Although FormCalc and JavaScript are intended for two different types of users, there is some overlap between the types of built-in functions they offer. The following table lists all available FormCalc functions and whether a comparable function exists within JavaScript.

<table>
<thead>
<tr>
<th>FormCalc function</th>
<th>Description</th>
<th>JavaScript method equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs(n1)</td>
<td>Returns the absolute value of a numeric value or expression.</td>
<td>Math.abs(n1)</td>
</tr>
<tr>
<td>Apr(n1, n2, n3)</td>
<td>Returns the annual percentage rate for a loan.</td>
<td>None</td>
</tr>
<tr>
<td>At(s1, s2)</td>
<td>Locates the starting character position of a string within another string.</td>
<td>String.search(s1)</td>
</tr>
<tr>
<td>FormCalc function</td>
<td>Description</td>
<td>JavaScript method equivalent</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Avg[n1 [, n2...]]</td>
<td>Evaluates a set of number values and/or expressions and returns the average of the non-null elements contained within that set.</td>
<td>None</td>
</tr>
<tr>
<td>Ceil[n1]</td>
<td>Returns the whole number greater than or equal to a given number.</td>
<td>Math.ceil[n1]</td>
</tr>
<tr>
<td>Choose[n1, s1 [, s2...]]</td>
<td>Selects a value from a given set of parameters.</td>
<td>None</td>
</tr>
<tr>
<td>Concat[s1 [, s2...]]</td>
<td>Returns the concatenation of two or more strings.</td>
<td>String.concat(s1, s2 [, s3 ...])</td>
</tr>
<tr>
<td>Count[n1 [, n2...]]</td>
<td>Evaluates a set of values and/or expressions and returns the number of non-null elements contained within the set.</td>
<td>None</td>
</tr>
<tr>
<td>CTerm[n1, n2, n3]</td>
<td>Returns the number of periods needed for an investment earning a fixed, but compounded, interest rate to grow to a future value.</td>
<td>None</td>
</tr>
<tr>
<td>Date()</td>
<td>Returns the current system date as the number of days since the epoch.</td>
<td>Date.getDate() The JavaScript Date object does not use the epoch as a reference point.</td>
</tr>
<tr>
<td>Date2Num[d1 [, f1 [, k1]]]</td>
<td>Returns the number of days since the epoch, given a date string.</td>
<td>The JavaScript Date object does not use the epoch as a reference point.</td>
</tr>
<tr>
<td>DateFmt[n1 [, k1]]</td>
<td>Returns a date format string, given a date format style.</td>
<td>None</td>
</tr>
<tr>
<td>Decode[s1 [, s2]]</td>
<td>Returns the decoded version of a given string.</td>
<td>Partial support JavaScript only supports URL encoded values that contain no escape characters.</td>
</tr>
<tr>
<td>Encode[s1 [, s2]]</td>
<td>Returns the encoded version of a given string.</td>
<td>Partial support JavaScript only supports URL encoded values that contain no escape characters.</td>
</tr>
<tr>
<td>Eval()</td>
<td>Returns the value of a given form calculation.</td>
<td>eval(s1)</td>
</tr>
<tr>
<td>Exists[v1]</td>
<td>Determines whether the given parameter is a valid reference syntax to an existing object.</td>
<td>None</td>
</tr>
<tr>
<td>Floor[n1]</td>
<td>Returns the largest whole number that is less than or equal to the given value.</td>
<td>Math.floor[n1]</td>
</tr>
<tr>
<td>Format[s1, s2]</td>
<td>Formats the given data according to the specified picture format string.</td>
<td>None</td>
</tr>
<tr>
<td>FV[n1, n2, n3]</td>
<td>Returns the future value of consistent payment amounts made at regular intervals at a constant interest rate.</td>
<td>None</td>
</tr>
<tr>
<td>Get[s1]</td>
<td>Downloads the contents of the given URL.</td>
<td>None</td>
</tr>
<tr>
<td>HasValue[v1]</td>
<td>Determines whether the given parameter is a valid reference syntax with a non-null, non-empty, or non-blank value.</td>
<td>None</td>
</tr>
<tr>
<td>IPmt[n1, n2, n3, n4, n5]</td>
<td>Returns the amount of interest paid on a loan over a set time.</td>
<td>None</td>
</tr>
<tr>
<td>IsoDate2Num[d1]</td>
<td>Returns the number of days since the epoch, given a valid date string.</td>
<td>None</td>
</tr>
<tr>
<td>FormCalc function</td>
<td>Description</td>
<td>JavaScript method equivalent</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>IsoTime2Num(d1)</td>
<td>Returns the number of milliseconds since the epoch, given a valid time string.</td>
<td>None</td>
</tr>
<tr>
<td>Left(s1, n1)</td>
<td>Extracts a specified number of characters from a string, starting with the first character on the left.</td>
<td>String.substring(n1, n2)</td>
</tr>
<tr>
<td>Len(s1)</td>
<td>Returns the number of characters in a given string.</td>
<td>String.length</td>
</tr>
<tr>
<td>LocalDateFmt(n1, k1)</td>
<td>Returns a localized date format string, given a date format style.</td>
<td>None</td>
</tr>
<tr>
<td>LocalTimeFmt(n1, k1)</td>
<td>Returns a localized time format string, given a time format style.</td>
<td>None</td>
</tr>
<tr>
<td>Lower(s1, k1)</td>
<td>Converts all uppercase characters within a specified string to lowercase characters.</td>
<td>String.toLowerCase(s1)</td>
</tr>
<tr>
<td>Ltrim(s1)</td>
<td>Returns a string with all leading white space characters removed.</td>
<td>None You can use JavaScript regular expressions to perform this operation.</td>
</tr>
<tr>
<td>Max(n1, n2...)</td>
<td>Returns the maximum value of the non-null elements in the given set of numbers.</td>
<td>Math.max(n1, n2)</td>
</tr>
<tr>
<td>Min(n1, n2...)</td>
<td>Returns the minimum value of the non-null elements of the given set of numbers.</td>
<td>Math.min(n1, n2)</td>
</tr>
<tr>
<td>Mod(n1, n2)</td>
<td>Returns the modulus of one number divided by another.</td>
<td>Use the modulo (%) operator.</td>
</tr>
<tr>
<td>NPV(n1, n2, ...)</td>
<td>Returns the net present value of an investment based on a discount rate and a series of periodic future cash flows.</td>
<td>None</td>
</tr>
<tr>
<td>Num2Date(n1, f1, k1)</td>
<td>Returns a date string given a number of days since the epoch.</td>
<td>None</td>
</tr>
<tr>
<td>Num2GMTTime(n1, f1, k1)</td>
<td>Returns a GMT time string given a number of milliseconds from the epoch.</td>
<td>None</td>
</tr>
<tr>
<td>Num2Time(n1, f1, k1)</td>
<td>Returns a time string given a number of milliseconds from the epoch.</td>
<td>None</td>
</tr>
<tr>
<td>Oneof(s1, s2, ..., s3...)</td>
<td>Returns true (1) if a value is in a given set and false (0) if it is not.</td>
<td>None This function is similar to the String.search(s1) method and String.match(expression) method.</td>
</tr>
<tr>
<td>Parse(s1, s2)</td>
<td>Analyzes the given data according to the given picture format.</td>
<td>None</td>
</tr>
<tr>
<td>Pmt(n1, n2, n3)</td>
<td>Returns the payment for a loan based on constant payments and a constant interest rate.</td>
<td>None</td>
</tr>
<tr>
<td>Post(s1, s2, s3, s4, s5)</td>
<td>Posts the given data to the specified URL.</td>
<td>None</td>
</tr>
<tr>
<td>PPmt(n1, n2, n3, n4, n5)</td>
<td>Returns the amount of principal paid on a loan over a period of time.</td>
<td>None</td>
</tr>
<tr>
<td>Put(s1, s2, s3)</td>
<td>Uploads the given data to the specified URL.</td>
<td>None</td>
</tr>
<tr>
<td>PV(n1, n2, n3)</td>
<td>Returns the present value of an investment of periodic constant payments at a constant interest rate.</td>
<td>None</td>
</tr>
<tr>
<td>FormCalc function</td>
<td>Description</td>
<td>JavaScript method equivalent</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Rate(n1, n2, n3)</td>
<td>Returns the compound interest rate per period required for an investment to grow from present to future value in a given period.</td>
<td>None</td>
</tr>
<tr>
<td>Ref()</td>
<td>Returns a reference to an existing object.</td>
<td>None</td>
</tr>
<tr>
<td>Replace(s1, s2 [, s3 ])</td>
<td>Replaces all occurrences of one string with another within a specified string.</td>
<td>String.replace(s1, s2)</td>
</tr>
<tr>
<td>Right(s1, n2)</td>
<td>Extracts several characters from a given string, beginning with the last character on the right.</td>
<td>String.substring(n1, n2)</td>
</tr>
<tr>
<td>Round(n1 [, n2 ])</td>
<td>Evaluates a given numeric value or expression and returns a number rounded to the given number of decimal places.</td>
<td>Math.round(n1)</td>
</tr>
<tr>
<td>Rtrim(s1)</td>
<td>Returns a string with all trailing white space characters removed.</td>
<td>None</td>
</tr>
<tr>
<td>You can use JavaScript regular expressions to perform this operation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Space(n2)</td>
<td>Returns a string consisting of a given number of blank spaces.</td>
<td>None</td>
</tr>
<tr>
<td>Str(n1 [, n2 [, n3 ])</td>
<td>Converts a number to a character string. FormCalc formats the result to the specified width and rounds to the specified number of decimal places.</td>
<td>String(n1) or Number.toNumber(radix)</td>
</tr>
<tr>
<td>Stuff(s1, n1, n2 [, s2 ])</td>
<td>Inserts a string into another string.</td>
<td>None</td>
</tr>
<tr>
<td>Substr(s1, n1, n2)</td>
<td>Extracts a portion of a given string.</td>
<td>String.substring(n1, n2)</td>
</tr>
<tr>
<td>Sum(n1 [, n2... ])</td>
<td>Returns the sum of the non-null elements of a given set of numbers.</td>
<td>None</td>
</tr>
<tr>
<td>Term(n1, n2, n3)</td>
<td>Returns the number of periods required to reach a given future value from periodic constant payments into an interest-bearing account.</td>
<td>None</td>
</tr>
<tr>
<td>Time()</td>
<td>Returns the current system time as the number of milliseconds since the epoch.</td>
<td>Date.getTime()</td>
</tr>
<tr>
<td>The JavaScript Date object does not use the epoch as a reference point.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time2Num(d1 [, f1 [, k1 ])</td>
<td>Returns the number of milliseconds since the epoch, given a time string.</td>
<td>None</td>
</tr>
<tr>
<td>TimeFmt([n1 [, k1 ])</td>
<td>Returns a time format, given a time format style.</td>
<td>None</td>
</tr>
<tr>
<td>UnitType(s1)</td>
<td>Returns the units of a unitspan. A unitspan is a string consisting of a number followed by a unit name.</td>
<td>None</td>
</tr>
<tr>
<td>UnitValue(s1 [, s2 ])</td>
<td>Returns the numeric value of a measurement with its associated unitspan after an optional unit conversion.</td>
<td>None</td>
</tr>
<tr>
<td>Upper(s1 [, k1 ])</td>
<td>Converts all lowercase characters within a string to uppercase.</td>
<td>String.toUpperCase()</td>
</tr>
<tr>
<td>Uuid(n2)</td>
<td>Returns a Universally Unique Identifier (UUID) string to use as an identification method.</td>
<td>None</td>
</tr>
</tbody>
</table>
Using FormCalc

FormCalc is a simple yet powerful calculation language modeled on common spreadsheet software. Its purpose is to facilitate fast and efficient form design without requiring a knowledge of traditional scripting techniques or languages. Using several of the built-in functions, inexperienced FormCalc users can quickly create forms that save users from performing time-consuming calculations, validations, and other verifications. This way, you can create a basic set of rules for the form design that allows the resulting form to react according to the data it comes into contact with.

Within Designer ES2, FormCalc is the default scripting language in all scripting locations with JavaScript as the alternative.

**Caution:** If you are developing forms for use with a server-based process (for example, using Forms ES2), with the intent of rendering your forms in HTML, you should develop your calculations and scripts in JavaScript. FormCalc calculations are not valid in HTML browsers, and are removed prior to the form being rendered in HTML.

FormCalc treats each new line in the Script Editor as a new expression to evaluate.

### Using built-in functions

The built-in functions that comprise FormCalc cover a wide range of areas, including mathematics, dates and times, strings, finance, logic, and the web. These areas represent the types of functionality that usually occur in forms. The purpose of the functions is to provide quick and easy manipulation of form data in a useful way.

At the most basic level, a calculation can consist of only a single FormCalc function. However, a single FormCalc function can use other FormCalc functions as parameters.

### To attach a FormCalc function to an object

You can add a FormCalc function to any form design object that allows calculations and scripts, except for the script object.

1. Make sure that you have the multiline version of the Script Editor displayed in the Designer ES2 workspace.
2. Select a field on your form.
3. In the Show list, select the calculate event.
4. Click the Functions icon or F10 to display a list of FormCalc functions.
5. Select the appropriate function and press Enter.
6. Replace the default function syntax notation with your own set of values.
7. Click the Preview PDF tab to test the form.

### Built-in function syntax

Each FormCalc function uses a specific syntax notation that you must follow in order for the function to execute correctly. This table describes, generally, the pieces of syntax notation.

<table>
<thead>
<tr>
<th>Syntax notation</th>
<th>Replacement values</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>A valid date string (for example, 03/15/1996)</td>
</tr>
<tr>
<td>f</td>
<td>A valid date format string (for example, MM/DD/YYYY)</td>
</tr>
<tr>
<td>k</td>
<td>A valid locale identifier (for example, fr_FR)</td>
</tr>
</tbody>
</table>
Creating basic calculations

About basic calculations
Simple expressions are the most basic instances of scripting. These expressions do not involve using FormCalc built-in functions and are never more than a single line. Add simple expressions to the calculate event of a particular field or object so that the expression value can output onto your form.

Examples of basic calculations
These examples are all of simple expressions:

2
"abc"
2 - 3 * 10 / 2 + 7

Each simple expression evaluates to a single value by following a traditional order of operations, even if the order is not always obvious from the expression syntax. For example, the following sets of expressions produce equivalent results.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent to</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;abc&quot;</td>
<td>&quot;abc&quot;</td>
<td>abc</td>
</tr>
<tr>
<td>2 - 3 * 10 / 2 + 7</td>
<td>2 - (3 * 10 / 2) + 7</td>
<td>-6</td>
</tr>
<tr>
<td>(10 + 2) * (5 + 4)</td>
<td>(10 + 2) * (5 + 4)</td>
<td>108</td>
</tr>
<tr>
<td>0 and 1 or 2 &gt; 1</td>
<td>(0 and 1) or (2 &gt;1)</td>
<td>1 (true)</td>
</tr>
<tr>
<td>2 &lt; 3 not 1 == 1</td>
<td>(2 &lt; 3) not (1 == 1)</td>
<td>0 (false)</td>
</tr>
</tbody>
</table>

As implied in the previous table, all FormCalc operators carry a certain precedence when they appear within expressions. The following table illustrates this operator hierarchy.

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>=</td>
</tr>
<tr>
<td></td>
<td>(Unary) -, +, not</td>
</tr>
<tr>
<td></td>
<td>* , /</td>
</tr>
<tr>
<td></td>
<td>+, -</td>
</tr>
</tbody>
</table>
All the previous examples are valid, simple expressions that you can add to a form field or object that accepts calculations and scripts. For example, if you create a form in Designer ES2 with a single numeric field, add the following calculation to the calculate event in the Script Editor.

![Script Editor screenshot]

Then, when you click the Preview PDF tab to view the completed form, the value of the simple expression appears in the text field.

![Preview PDF screenshot]

If the value does not appear in the preview, ensure that your simple expression appears in the calculate event of the form design object. Also, ensure that you installed Designer ES2 and Acrobat correctly.
Using JavaScript

To allow form designers more flexibility and scripting power, Designer ES2 supports the use of JavaScript version 1.6 or earlier in all situations that support scripting.

Form developers who are familiar with JavaScript can apply their existing expertise directly to Designer ES2. Designer ES2 provides several properties and methods that enhance JavaScript so that you to access field and object values. These properties and methods combined with the Designer ES2 reference syntax provide you with easy manipulation of form values and data.

Note: The Script Editor does not provide syntax error checking for scripts created using JavaScript. In addition, statement completion options do not appear for standard JavaScript objects or methods.

Creating scripts using JavaScript

Creating scripts in Designer ES2 using JavaScript is similar to creating JavaScript in other applications. You can take advantage of previous knowledge of JavaScript concepts, reuse JavaScript functions using the Designer ES2 script object, and take advantage of JavaScript language functionality.

However, notice that although previous JavaScript knowledge is transferable, to effectively use JavaScript on your form design, you must understand how to construct Designer ES2 reference syntax. Specifically, you must know how to correctly use the XML Form Object Model reference syntax to access objects on your form design.

Enforcing strict scoping rules in JavaScript

When working with JavaScript in forms, it is important to declare objects and variables within the scope they are intended. Globally declaring objects or variables unnecessarily can cause performance problems. Strict scoping was introduced in Designer 8.1 to improve the run time and memory usage of a form.

What is scope in JavaScript?

Scope works outwardly so that everything within curly brackets ({}) can see outside them. However anything outside the curly brackets cannot access anything inside them.

In the following example, the first curly bracket opens the function scope and the second one closes it. Everything between the curly brackets is within the scope of foo().

```javascript
noutsidevar is outside

the scope of foo() -> var noutsidevar = 2;

function foo()

This opens the function scope -> {

    // everything between the two
    // curly braces is within the
    // scope of foo()
    var nfoovar = 4;

This closes the function scope -> }
```
The scope in the following example is valid because `var nFooVar = nOutsideVar` inside the curly brackets can see `var nOutsideVar = 2` outside the curly brackets.

```javascript
var nOutsideVar = 2;

function foo()
{
    var nFooVar = nOutsideVar; // This is correct;
    // anything inside the
    // braces can see stuff
    // outside

}
```

In contrast, the following example shows an invalid scope because `var nOutsideVar = nFooVar` cannot access `var nFooVar = 4` within the curly brackets.

```javascript
function foo()
{
    var nFooVar = 4;
}

var nOutsideVar = nFooVar; // This is incorrect,
    // nOutsideVar cannot access
    // things declared within
    // foo()'s scope
```

Scope in scripting describes pieces of scripts that can access pieces. The pieces of script can be variables or functions.

**What is scope XML?**
Scope in a form design is about hierarchy. For example, to access the subform `inside` in the following XML source, you must type `outside.inside`.

```xml
<subform name="outside">
    <subform name="inside">
        ...
    </subform>
</subform>
```
You do not type \texttt{inside.outside} because you must access the outermost subform first and drill inwards.

\section*{SOM expressions and scope}

In forms that are targeted for Acrobat or Adobe Reader 8.1, SOM expressions are properly scoped as shown in this example:

```
<subform name="a">
  <subform name="b"/>
</subform>
```

In forms targeted for Acrobat or Adobe Reader 8.0, the SOM expression \texttt{a.b.a} returns the subform \texttt{a}. In forms targeted for Acrobat or Adobe Reader 8.1, the SOM expression \texttt{a.b.a} returns \texttt{null} because subform \texttt{b} does not have a child named \texttt{a}. In Acrobat or Adobe Reader 9.0 or later, the expression returns an error because \texttt{a} is not a valid child of \texttt{b}.

In Acrobat or Adobe Reader 8.1, functions and variable within a node's script do not become global (script objects being the exception) as shown in this example:

```
<field name="field1">
  event activity="initialize">
    <script contentType="application/x-javascript">
      // Function \texttt{bar()} is scoped to \texttt{field1.initialize}; nothing outside \texttt{<event activity="initialize">}
      scope can see inside here (in 8.1)
       
      function bar()
      {
        return "bar";
      }
    </script>
  </event>
</field>

<field name="field2">
  <event activity="click">
    <script contentType="application/x-javascript">
      field1.bar();
    </script>
  </event>
</field>
```

When you click \texttt{field 2} in a form targeting Acrobat or Adobe Reader 8.0, the function \texttt{bar()} executes.

When you click \texttt{field 2} in a form targeting Acrobat or Adobe Reader 8.0, the function \texttt{bar()} does not execute. The reason is because function \texttt{bar()} is available only from within the initialized script of \texttt{field1}. 
Scoping and script objects
Script objects have global scope; therefore, anyone can access them from anywhere. If you have a method that you want both
field1.initialize and field2.click to access, place it in a script object. Strict scoping means that you cannot call bar() from
anywhere in a form. You also get a run-time error indicating that the function bar() could not be resolved. The script engine looked for
bar() within the scope that you have access to and did not find it.

Scoping and target version
If you use strict scoping, remember that you get performance improvements in forms targeted for Acrobat or Adobe Reader 8.1 and later.
Avoid using strict scoping in forms targeted for older versions of Acrobat or Adobe Reader. Otherwise, the scripts in the forms can work
differently. For existing forms, back up the form before you enable strict scoping and always verify the script afterwards. When you enable
strict scoping and then change the target version to earlier than Acrobat or Adobe Reader 8.1, warning messages appear.

When to use scoping
When a form is targeted for Acrobat or Adobe Reader 8.1 and strict scoping is on, declared JavaScript variables are released after each script
executes. When a form is targeted for Acrobat or Adobe Reader 9.0 and later, strict scoping does not release all the JavaScript variables. The
excepting is when you remerge or import new data.

The performance enhancements with strict scoping rules apply to forms targeted for Acrobat or Adobe Reader 8.1 and later. Do not apply
strict scoping rules to forms that are targeted for versions of Acrobat or Adobe Reader earlier than 8. Otherwise, the scripts can behave
differently or not work.

To enable strict scoping
1 Select File > Form Properties and click the Defaults tab.
2 Select Enforce Strict Scoping Rules In JavaScript and then click OK.

To attach a JavaScript script to an object
You can add a JavaScript script to any form design object that allows calculations and scripts, including the script object.
1 Make sure that you have the multiline version of the Script Editor displayed in the Designer ES2 workspace.
2 Select a field on your form. For example, add a new text field to your form design.
3 In the Show list, select a valid event. For example, using the new text field, select the docReady event.
4 In the Run At list, select where you want the script to execute. For example, for the new text field, select Client.
5 Click the Functions icon or F10 to display a list of JavaScript functions.
6 Select the desired function, and press Enter.
7 Replace the default function syntax notation with your own set of values. Alternatively, you can create your own script manually in the
Script Source field of the Script Editor. For example, in the new text field, add the following JavaScript to the Script Source field:
this.border.fill.color.value = "255,0,0";
8 Click the Preview PDF tab to test the form. The text appears red for the new button object when the form is displayed in the Preview
PDF tab.

Variables
You can define form variables in Designer ES2 to store specific information in a central, accessible location. A variable typically acts as a
placeholder for text that you might have to change in the future. Form variables in Designer ES2 are always of the type "string". For example,
a variable can store the text of a message box title. When the text needs to change, all you have to do is open the affected form or template
and update the text once through the variable definition. Designer ES2 automatically propagates the new text across all instances of the
inserted variable.
Keep in mind that form variables are defined outside of the Script Editor, and are accessible by scripts on all objects on a form, as opposed to scripting variables that you create in a specific FormCalc or JavaScript script.

You can create, view, and delete variables without using scripting. However, you must use scripting to access the values stored by variables and manipulate them, or to apply the values to objects on your form.

*Note: Form variable values reset each time you open a form.*

Before you create a variable, decide the name of the variable and the text that it will contain. Variable definitions are saved with the form or template.

### Naming variables

At run time, naming conflicts occur when the names of variables are identical to those used as XML Form Object Model properties, methods, or form design field names. These conflicts can cause scripts to return unexpected values; therefore, it is important to give each variable a unique name. Here a couple of examples:

- Use the variable name `fieldWidth` and `fieldHeight` instead of `x` and `y`.
- Use the form design object name `clientName` instead of `name`.

*Note: Variable names are case-sensitive and should not contain spaces.*

### To define a text variable

1. Select File > Form Properties.
2. In the Variables tab, click New (Insert).
3. In the Variables list, type a unique name for the variable and press Enter. Variable names are case-sensitive and should not contain spaces.
4. Click once in the box to the right and type the text you want to associate with the variable.

The variable appears in the Hierarchy palette at the form level.

![Hierarchy palette](image)

*New form variable*
The variable appears in the Hierarchy palette at the form level.

A. New form variable

To view a text variable definition
1. Select Edit > Form Properties.
2. Click the Variables tab and select the variable from the Variables list. The associated text is displayed in the box to the right.

To delete a text variable
1. Select File > Form Properties.
2. In the Variables tab, select the variable and click Delete (Delete).

Using variables in calculations and scripts
After you have created form variables, you only need to reference the variable name in your calculations and scripts in order to obtain the value of the variable.

Caution: When naming variables, you should avoid using names that are identical to the names of any XML Form Object Model properties, methods, or object names. For information about XML Form Object Model properties, methods, and objects, see the Designer ES2 Scripting Reference.

For example, create the following form variable definitions.

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstName</td>
<td>Tony</td>
</tr>
<tr>
<td>lastName</td>
<td>Blue</td>
</tr>
<tr>
<td>age</td>
<td>32</td>
</tr>
</tbody>
</table>
In FormCalc, you can access the variable values in the same manner that you access field and object values. In this example, the values are assigned to three separate fields:

- TextField1 = firstName
- TextField2 = lastName
- NumericField1 = age

You can also use variables in FormCalc functions in the same way, as shown in this example:

```
Concat( "Dear ", firstName, lastName )
```

In JavaScript, you reference variable values by using the `value` property instead of the `rawValue` property that is used for field and object values, as shown in this example:

```
TextField1.rawValue = firstName.value;
```

### Creating and Reusing JavaScript Functions

The `script object` is an object you can use to store JavaScript functions and values separately from any particular form object. Typically, you use the script object to create custom functions and methods that you want to use as part of JavaScript scripts in many locations on your form. This technique reduces the overall amount of scripting required to perform repetitive actions.

The script object only supports script written in JavaScript; however, there are no restrictions on the location where the scripts are executed, provided that the scripting language for the event that invokes the script object is set to JavaScript.

Both Acrobat and Forms ES2 process scripting from a script object in the same manner, but both are also distinct.

Both Acrobat and Adobe document services process scripting from a script object in the same manner, but both are also distinct.

Only scripts set to run on the client can make use of script objects set to run on the client, and vice versa.

#### To create a script object

There are two parts to creating a script object. The first part involves adding the object to the form design, and the second part is writing the script you want to store in the script object.

1. Create a new form or open an existing form.
2. In the Hierarchy palette, right-click either a form-level object or a subform-level object and select Insert Script Object.

![Hierarchy palette with script object](image)

**A. Form level object**  **B. Subform level object**  **C. Subform level script object**  **D. Form level script object**

3. (Optional) Right-click the script object and select Rename Object.
To add script to a script object

After you have a script object on your form, you can add scripts using the Script Editor.

1. Select the script object in the Hierarchy palette.
   The Script Editor is displayed with both a Script Object value in the Show list and a JavaScript value in the Language list. You cannot change either of these values.

2. Enter your script in the Script Source field.
3. Click the Preview PDF tab to test the form.

Example
For example, create a script object called feedback that contains the following function:

```javascript
function emptyCheck(oField) {
    if ((oField.rawValue == null) || (oField.rawValue == "")) {
        xfa.host.messageBox("You must input a value for this field.", "Error Message", 3);
    }
}
```

To reference JavaScript functions stored in a script object

After you add scripts to a script object, you can reference the script object from any event that supports JavaScript scripts.

1. Select an object on your form and select an event from the Show list.
2. Create a reference to the script object and any functions within the script object. The following generic syntax assumes that the object where you are referencing the script object is at the same level as the script object in the form hierarchy or that the script object exists at the highest level of the form hierarchy.

   `script_object.function_name(parameter1, ...);`

3. Apply the new script to the form object and test it by previewing the form using the Preview PDF tab.

   Similar to referencing other objects on a form, you must provide a valid syntax when referencing the script object that includes where it exists within the form hierarchy.

Using Script Fragments

A script fragment contains a script object. A script object contains reusable JavaScript functions or values that are stored separately from any particular form object, such as a date parser or a web service invocation. Typically, you use the script objects to create custom functions and methods that you want to use in many locations on a form. Using script objects reduces the overall amount of scripting required to perform repetitive actions.

Script fragments include only script objects that appear as children of variables in the Hierarchy palette. Fragments cannot contain scripts that are associated with other form objects, such as event scripts like validate, calculate, or initialize.
You edit script fragments the same way as other fragments.

**Script fragment properties**
When you select a script fragment, the Script Object tab in the Object palette displays the fragment properties.

**Source File**
Sets the source file for the fragment reference. This property is visible only when the selected object is a fragment reference.

**Fragment Name**
Sets the name of the fragment. You can click the Frag Info button to view the fragment information.

This property is visible when a fragment reference or a fragment that is defined in a source file is selected. When the selected object is a fragment reference, this property does not appear if the source file is not specified. The Fragment Name list includes all the fragments in the specified source file. The Custom option directly supports setting a SOM expression or an ID value as the fragment reference and supports the implementation in the XML Forms Architecture.

**To create a script fragment**
You can create a script fragment of common functions that you can reuse in multiple forms. To create a script fragment, you create a script object that contains the functions that you want to reuse in multiple form designs. The script fragment can include only one script object.

1. Create a script object.
2. In the Hierarchy palette, right-click the script object and select Fragments > Create Fragment.
   
   *Note: You can also create a script fragment by dragging the script object from the Hierarchy palette to the Fragment Library palette.*
3. To use a different fragment name, in the Name box, type a name for the fragment.
4. (Optional) In the Description box, type a description of the fragment.
5. Select a method for creating the fragment:
   - To define the fragment in a separate XDP file that is stored in the Fragment Library, select Create New Fragment In Fragment Library. In the Fragment Library list, select the Fragment Library in which to save the fragment file. To use a different file name, in the File Name box, type the file name for the fragment. If you do not want to replace the selection with the new fragment, deselect Replace Selection With Reference To New Form Fragment.
   - To define the fragment in the current file, select Create New Fragment In Current Document.
6. Click OK.

**To insert a script fragment**
You can use script fragments to reuse JavaScript functions in multiple forms. When creating a form design, you insert a reference to an existing script fragment and the fragment appears in the form design.

You cannot insert a fragment in an XFAF document.

*Note: To preview the fragments in the Fragment Library palette, select Show Preview Pane from the palette menu.*

**To insert a script fragment from the Fragment Library palette:**
1. In the fragment library, select the script fragment.
2. Drag the fragment to a subform or variables object in the Hierarchy palette.

**To insert a script fragment from the Insert menu:**
1. Select Insert > Fragment.
2 Navigate to the file that contains the fragment.
3 Select the file and click OK. The fragment appears as a child of the variables object in the root subform

Debugging Calculations and Scripts

Designer ES2 includes various features and strategies for debugging calculations and scripts, depending on the scripting language you choose.

For JavaScript language script debugging, you can use the `alert` or the `messageBox` methods to provide debugging feedback. One disadvantage of this method is that you must close many message boxes. Another problem is that displaying a message box can cause differences in the form's behavior, especially if you are trying to debug a script that is setting focus to an object on your form. It is best to use `console.println` to output text to the JavaScript Console from Acrobat to debug a form.

Designer ES2 Report palette warning and validation messages
The Report palette provides warning and validation messages to help you debug a form as you design it. The Warning tab lets you view errors or messages that Designer ES2 generated as you design a form. The Log tab lets you view the following errors and messages:

- Validation messages
- JavaScript or FormCalc scripting execution errors
- Design-time form rendering errors that are generated when you import or save a form or preview a form from the Preview PDF tab.

Providing debugging feedback using the `messageBox` method
The XML Form Object Model `messageBox` method lets you output information from an interactive form into a dialog box at run time. You can take advantage of the XML Form Object Model `messageBox` method to display messages or field values at runtime. When initiated, the `messageBox` method displays a string value in a new client application dialog box. The string value can be a text message that you create for debugging purposes or the string value of fields or expressions.

For example, consider a scenario with a simple form design that contains a single numeric field (NumericField1) and a button (Button1). In this case, the following FormCalc calculation and JavaScript script each output a message displaying some text and the value currently displayed in the numeric field. By adding either the calculation or the script to the `click` event of the button object, you can interactively display the value of the numeric field in a new dialog box by clicking the button.

FormCalc

```javascript
xfa.host.messageBox(Concat("The value of NumericField1 is: ",
   NumericField1), "Debugging", 3)
```

JavaScript

```javascript
xfa.host.messageBox("The value of NumericField1 is: " +
   NumericField1.rawValue, "Debugging", 3);
```

Caution: The `messageBox` method returns an integer value representing the button that the form filler selects in the message box dialog. If you attach the `messageBox` method to the `calculate` event of a field object, and the `messageBox` method is the last line of the script, the field displays the return value of the `messageBox` method at runtime.

Output information into a text field
You can output information, such as field values or messages, into a text field on your form design. For example, you can append new messages or values to the value of a text field to create a log for future reference.
**JavaScript Debugging**

If you use the JavaScript language for a script, you can use the `console.println("string")` function to output information to the JavaScript Console available in Acrobat Professional. Alternatively, you can use the `alert` method from the Acrobat JavaScript Object Model to debug JavaScript.

**JavaScript Debugger in Acrobat Professional**

The JavaScript Debugger in Acrobat Professional lets you test JavaScript scripts. The debugger includes the JavaScript Console, where you can test portions of JavaScript code in the Preview PDF tab. The JavaScript Console provides an interactive and convenient interface for testing portions of JavaScript code and experimenting with object properties and methods. Because of its interactive nature, the JavaScript Console behaves as an editor that supports the execution of single lines or blocks of code.

To enable the JavaScript Debugger for Designer ES2 and execute code from the JavaScript Console, enable JavaScript and the JavaScript Debugger in Acrobat Professional.

*Note:* You can enable the JavaScript Debugger in Adobe Reader if you have Reader Extensions ES2 installed. To enable the JavaScript Debugger in Adobe Reader, you must get the debugger.js file and then edit the Microsoft Windows Registry.

**To enable the JavaScript Debugger for Designer ES2**

2. Start Acrobat Professional.
3. In Acrobat Professional, select Edit > Preferences.
4. Select JavaScript from the list on the left.
5. Select Enable Acrobat JavaScript if it is not already selected.
7. Select Enable Interactive Console. This option lets you evaluate code that you write in the JavaScript Console.
8. Select Show Console On Errors And Messages. This option ensures that if you make mistakes, the JavaScript Console displays helpful information.
9. Click OK to close the Preferences dialog box.
10. Quit Acrobat Professional.
11. In Designer ES2, click the Preview PDF tab.
12. Press Ctrl+J to open the JavaScript Debugger.

**To prevent the JavaScript Debugger from disappearing in Designer ES2**

If the JavaScript Debugger from Acrobat is active and it disappears when you click components in the Designer ES2 interface, stop the Acrobat.exe process in the Microsoft Windows Task Manager. The Acrobat.exe process continues to run after Acrobat is closed so that Acrobat is displayed faster if it is restarted. Stopping the process ends the association between the JavaScript Debugger and the Acrobat Professional session so that you can use the JavaScript Debugger in Designer ES2.

1. In the Windows Task Manager, click the Processes tab.
2. In the Image Name column, right-click Acrobat.exe and select End Process.

**Evaluating code using the JavaScript Console**

There are three ways evaluate single and multiple lines of code using the JavaScript Console from Acrobat.

**To evaluate a portion of a line of code**

- Highlight the portion in the console window and press either Enter on the numeric keypad or Ctrl+Enter on the regular keyboard.
To evaluate a single line of code

Place the cursor in the appropriate line in the console window and press Enter on the numeric keypad or Ctrl+Enter on the regular keyboard.

To evaluate multiple lines of code

Highlight those lines in the console window and press either Enter on the numeric keypad or Ctrl+Enter on the regular keyboard.

To delete content that appear in the JavaScript Console

Click Clear in the console window.

The result of the most recently evaluated JavaScript script is displayed in the console window.

After evaluating each JavaScript script, the console window prints out `undefined`, which is the return value of the statement. Notice that the result of a statement is not the same as the value of an expression within the statement. The return value `undefined` does not mean that the value of script is undefined; it means that the return value of the JavaScript statement is undefined.

Providing debugging feedback to the JavaScript Console

If you are creating scripts using JavaScript, you can output messages to the JavaScript Console from Acrobat at runtime by using the `console.println` method included with the JavaScript Object Model from Acrobat. When initiated, the `console.println` method displays a string value in the JavaScript Console. The string value can be a text message that you create for debugging purposes or the string value of fields or expressions.

For example, consider a simple form design that contains a single numeric field (NumericField1) and a button (Button1). In this case, the following JavaScript script outputs a message displaying some text and the value currently displayed in the numeric field. By adding either the calculation or the script to the `click` event of the button object, you can interactively display the value of the numeric field in a new dialog box by clicking the button.

```
console.println("The value is: " + NumericField1.rawValue);
```

For more information about the `console.println` method and the JavaScript Object Model from Acrobat, see Developing Acrobat Applications Using JavaScript (English only).

For more information about the JavaScript Console and the JavaScript Debugger, see Developing Acrobat Applications Using JavaScript (English only).

Providing debugging feedback using the alert method

If you want to return a message box during a `calculate` event, you can take advantage of the `alert` method from the JavaScript Object Model from Acrobat. For example, the following script returns the value of a text field:

```
var oField = xfa.resolveNode("TextField1").rawValue;
app.alert(oField);
```

For more information about the `alert` method and the JavaScript Object Model from Acrobat, see Developing Acrobat Applications Using JavaScript (English only).

Debugging tips

Remember the following tips when debugging calculations and scripts.

Sample data

Remember to specify a preview data file in the Form Properties dialog box. Specifying a preview data file does not save the data into the final PDF.
Master pages
To debug master pages, drop a different object on each master page to find out which one is specified.

First page in a form
Designer ES2 looks at the root subform to determine which page to begin the form on. If the root subform does not determine the first page, the first master page is used by default.

Incremental debugging
When debugging a form design, start by removing pieces of the form until you cannot reproduce the problem. Try to isolate the source of the problem after you've reviewed every script and object property. To debug subforms, you can specify a thick colored border around the subform, or use a fill. Colors or fill can help indicate which subform is used and its span. Usually, this technique works well when you want to determine the bounds of an object and can show why it is placed in a certain location.

Hierarchy view
View your form design by using the Hierarchy view to get a better understand of it. The order of the objects that are listed in the hierarchy indicates the order they are placed on the page. Some objects are not clickable if they are below one another.

Script error messages
In Designer ES2, script error messages appear on the Log tab of the Report palette when you preview the form. If the form design contains FormCalc scripts and the error occurs on the server, the warnings appear in the Log tab. If the FormCalc script error occurs on the client, the message appears in Adobe Reader or Acrobat.

An error in a FormCalc script prevents the entire script from executing.

An error in a JavaScript executes until it reaches the error.

Syntax errors in FormCalc
Syntax errors in FormCalc are sometimes difficult to solve. When the "Syntax error near token '%1' on line %2, column %3" appears, %1 usually contains the token (word) nearest to the error. Therefore the token is possibly correct and the error is not related to the error other than its proximity to it. For example, the following script generates the 7008 error: "Syntax error near token 'then' on line x, column y."

```
var b = abc(1)
if (b ne 1) then
  //comment
```

The problem is that an endif token is missing from the script. The last correct token is then (comments do not count as tokens). Adding an endif statement to the end of the script fixes the problem.

Functions defined in a script object
You can only call a function that is defined in a script object with a JavaScript script. Make sure that you change the script language to JavaScript in the Script Editor. If not, you may see a message indication that Designer ES2 cannot resolve the script object. The same error can occur when a syntax issue occurs in the script object.

Web service calls
When creating web service calls, use the postExecute event to see what was returned and whether the web service issued any error messages.

Long SOM expressions
When typing long, multilayered SOM expression, press the Ctrl key and click the object on the canvas. The command inserts the object's SOM expression into the script. The SOM expression is relative to the object hosting the script. To insert the absolute SOM, press Ctrl+Shift and click the object. These commands work when you click objects in the Design view, not in the Hierarchy view.
Test SOM expressions
When a long SOM expression fails, start back at the root of the expression and test each dot with className until you reach the problem. For example, test a.b.c.d by starting at the root:

- console.println(a.className)
- console.println(a.b.className)
- console.println(a.b.c.className)
- console.println(a.b.c.d.className)

Use script objects to debug form designs
Use a script object, such as a fragment, to help you debug form designs:

- Dump out a node hierarchy under a node.
- Output the value of a property or attribute of a node.
- Output whether a node has a property or attribute specified.
- Output the SOM expression of a node.
- Dump out the xml src of a given node.

Here is an example of a script object that contains several debugging functions:

```javascript
<script contentType="application/x-javascript" name="XFADEBUG">
// This script object provides several tracing functions to help debug a form design
// Dump out node hierarchy to console.println() function printNode(node) {... }
// Dump out SOM expression to console.println() function printSOM(node) {... }
// Dump out property or attribute value function printValue(node, attrOrPropertyName) {...}
function printXMLSource(node) {...}
function printHasPropertySpecified(node, prop) {...}
</script>
```

Things to avoid when building forms

- Calling xfa.layout.relayout(). on the docReady even causes problems because the docReady event triggers every time the layout is ready.
- Placing a flowed container inside a positioned container causes problems with page breaks, overlapping objects, and repeating subforms. The root subform is a flowed container. Take advantage of it and place your flowable containers inside the root subform by unwrapping the page subforms after your layout is done. Alternatively, set the flow of the page subforms to flowed.

Blank page issue (Acrobat 7.1 or earlier). At design time, a blank page is displayed when the subform does not fit within the boundaries of the content area. To fix the blank page, either resize the subform or allow it to break between pages. If a user is using Acrobat 7.1 or earlier, the second-level subform appears on a different page.
Hyphenating Text

Use hyphenation to improve text alignment in a given area by reducing the amount of white (empty) space between the last word on a line and the right margin. If a word is too long to fit entirely on a single line, the word is hyphenated at the proper hyphenation point, which forces a line break as close to the right margin as possible. Hyphenation makes each line of text approximately the same length to give the text a more uniform layout. Designer ES2 uses a hyphenation dictionary, metrics such as line spacing and font size, as well as other linguistic information to determine where various words can be legally and optimally hyphenated.

You can hyphenate the text in text objects, in the caption area of objects such as text fields, decimal fields, and numeric fields, and in the value area of text field objects (default text and text the form filler enters). For example, you can indicate the number of letters to allow in a word before it can be hyphenated; hyphenate capitalized words, such as the first word of a sentence; hyphenate words that are all capital letters, such as acronyms; and add or remove all hyphenation from the form.

You can set default hyphenation options for all new forms or customize hyphenation settings for individuals forms.

Set hyphenation options in these areas:

- **Options dialog box (Formatting panel)** Use the options in this dialog box to specify default hyphenation settings for all new forms. Changing these options does not affect the currently opened form.
- **Forms Properties dialog box (Formatting tab)** Use the options in this dialog box to specify hyphenation settings for the currently opened form only.
- **Paragraph palette (Hyphenation option)** Use this option to enable or disable hyphenation in individual objects.

**Note:** When you create a form, the Form Properties dialog box automatically inherits the default hyphenation settings for new forms from the Tools Options dialog box. To create a form that has different hyphenation settings or to change the default settings in an existing form, use the Form Properties dialog box. The settings in the Form Properties dialog box override the settings in the Options dialog box for the current form.

When you change the hyphenation settings in the Forms Properties dialog box for the current form, the settings in the Options dialog box (for new forms) do not change.

**Considerations for setting hyphenation**

Keep the following points in mind regarding the various options you must select for objects that contain hyphenated text:

**Hyphenate option in the Paragraph palette**

Although the hyphenation you select in the Form Properties and Options dialog boxes apply to the entire form, you can use the Hyphenate option in the Paragraph palette to add or remove hyphenation in individual objects. Using the Hyphenate option, you can manually adjust text layout on an object-by-object basis.

The Hyphenate option is available in the Paragraph palette only when you select the Using the Allow Hyphenation in Text and Field Captions option or the Allow Hyphenation in Text Field Values option, or both on the Form Properties (Formatting tab) or the Options (Formatting panel) dialog boxes. The Hyphenate option displays a colored square when these two options are in a mixed state, where either Allow Hyphenation in Text and Field Captions or Allow Hyphenation in Text Field Values is deselected. That is, when you select the Hyphenate option for an object when these two options are in a mixed state, the check box displays a colored arrow, which changes to a colored square to remind you that one of the options is deselected.

*When Hyphenate New Items is selected in the Options dialog box on the Formatting panel, the Hyphenate option is automatically selected with new objects added to a form.*
Currently Editing palette menu
The commands in the Currently Editing palette menu (Edit Caption or Value, Edit Caption, or Edit Value) determine when the Hyphenate option is available in the Paragraph palette (does not apply to text objects). That is, the Allow Hyphenation in Text and Field Captions option and the Allow Hyphenation in Text Field Values option are applied according to the command you select. For example, if you select the Allow Hyphenation in Text and Field Captions option, you must also select either the Edit Caption or Value command or the Edit Caption command to make the Hyphenate option available.

Use these commands to select the area (caption, value, or both) to hyphenate in each object that contains hyphenated text. For example, to hyphenate the text in text objects and the captions in text field objects, select the Allow Hyphenation in Text and Field Captions option and either the Edit Caption and Value menu command or the Edit Caption command. Alternatively, to hyphenate the default or user input text in the value area of text objects, select the Allow Hyphenation in Text Field Values option and the Edit Value menu command. The default command for text fields, decimal fields, numeric fields, and signature fields is Edit Caption or Value.

Expand to Fit options
It is recommended that you select the Expand to Fit (Height) option for each object that contains hyphenated text. When a word is too long to fit entirely on a line, hyphenation forces a line break and divides the word over two lines. As a result, the object that contains the text needs to expand in height to accommodate additional lines when needed. However, you can also manually enlarge objects to the correct size where hyphenation can occur. Text objects expand in width (as needed) when you enter text, even if you do not select the Expand To Fit (Width) option.

Allow Multiple Lines option
(Applies to the value area of text objects only) Select this option on the Field tab of the Object palette for each text object to hyphenate the text in, in the value area of text field objects. This option enables the text to break onto more than one line where the height of the field permits.

Setting and removing hyphenation in forms
You can hyphenate the text in, and remove hyphenation from, text objects, the caption area of objects such as text fields, decimal fields, numeric fields, and signature fields, and the value area of text field objects (default text and text the form filler enters).

To set hyphenation for new forms  Use the Formatting panel in the Options dialog box. The numbers you enter and the options you select on the Formatting panel are used as default settings in the Form Properties dialog box on the Formatting tab.

To set hyphenation for individual forms  Use the Formatting tab in the Form Properties dialog box. The numbers you enter and the hyphenation options you select on the Formatting tab apply to the current form only. You can change the hyphenation settings for the current form to suit your needs. The default settings that initially appear in the Form Properties dialog box are inherited from the Formatting panel in the Options dialog box, which automatically apply to all new forms. If you change the default settings in the Form Properties dialog box, the new settings override the settings in the Options dialog box for the current form. The hyphenation options in the Options dialog box do not change.

Note: Before you set hyphenation for forms, it is recommended that you become familiar with the different options you need to select for objects that contain hyphenated text.

To remove hyphenation from forms  Use the You can remove hyphenation from the text in text objects, in the caption area of objects such as text fields, decimal fields, numeric fields, and signature fields, and in the value area of text field objects.

To set hyphenation for a new form
1  Select Tools > Options.
2  Click Formatting, enter the values, and select the options to apply to new forms:
   •  In the Words With At Least <x> Letters box, enter the minimum number of letters that a word must contain to be hyphenated.
   •  In the After First <x> Letters box, enter the minimum number of letters in a word that must appear on a line before the hyphen.
In the Before Last \(<x>\) Letters box, enter the minimum number of letters in a word that must appear on the next line, after the hyphen.

To hyphenate words that begin with a capital (uppercase) letter, such as the first word of a sentence, select Hyphenate Capitalized Words.

To hyphenate words that are all capital letters, such as acronyms, select Hyphenate Words in ALL CAPS.

To hyphenate the text in text objects, and in the caption area of objects such as text fields, decimal fields, numeric fields, and signature fields, select Allow Hyphenation in Text and Field Captions.

To hyphenate the text in the value area of text field objects (default text and text entered by the person filling the form), select Allow Hyphenation in Text Field Values.

Select Hyphenate New Items to hyphenate the text in new objects added to the form.

Click OK.

To set hyphenation for an individual form

1. Select File > Form Properties.
2. Select Edit > Form Properties.
3. Click the Formatting tab, and enter the values and select the options to apply to the current form:
   - In the Words With At Least \(<x>\) Letters box, enter the minimum number of letters that a word must contain to be hyphenated.
   - In the After First \(<x>\) Letters box, enter the minimum number of letters in a word that must appear on a line before the hyphenation point.
   - In the Before Last \(<x>\) Letters box, enter the minimum number of letters in a word that must appear on the next line after the hyphenation point.
   - To hyphenate words that begin with a capital (uppercase) letter, such as the first word of a sentence, select Hyphenate Capitalized Words.
   - To hyphenate words that are all capital letters, such as acronyms, select Hyphenate Words in ALL CAPS.
   - To hyphenate the text in text objects, and in the caption area of objects such as text fields, decimal fields, numeric fields, and signature fields, select Allow Hyphenation in Text and Field Captions.
   - To hyphenate the text in the value area of text field objects (default text and text entered by the person filling the form), select Allow Hyphenation in Text Field Values.
4. Click Hyphenate All Text.

Click Edit > Undo Hyphenate All text to immediately undo hyphenation.

To remove hyphenation from a form

1. Select File > Form Properties.
2. Select Edit > Form Properties.
3. Click the Formatting tab and do one of the following actions:
   - To remove hyphenation from the text in text objects and in the caption area of objects, deselect Allow Hyphenation in Text and Field Captions.
   - To remove hyphenation from the text in the value area of text field objects, deselect Allow Hyphenation in Text Field Values.
   - To remove all hyphenation, click Remove All Hyphenation.
4. Click OK.

To immediately restore all hyphenation, click Edit > Undo Remove All Hyphenation.”
Page Break Controls

Designer ES2 provides ways to control the page breaks that are inserted when a form that has a flowable layout expands beyond the bottom edge of a rendered page:

- **Widow and Orphan Control**  Lets you prevent a single line of text from being separated from the rest of a paragraph when a page break is introduced. A *widow* is the last line of a paragraph that appears by itself at the top of the next page. An *orphan* is the first line of a paragraph that appears by itself at the bottom of a page.

- **Keep With Next**  Lets you keep a text object or text field object with the next object in the document when a page break is introduced.

- **Allow Page Breaks Within Content**  Allows page breaks within the content of a text object or text field object.

When both the Keep with Next and Allow Page Breaks Within Content options are selected, the Allow Page Breaks Within Content option is considered first followed by the Keep with Next option.

The Widow and Orphan, Keep With Next, and Allow Page Breaks Within Content options are not available for objects on a master page or in artwork.

You can set the default page break options for new forms, page break options for an individual form, and page break options for selected objects by using the various options in the Form Properties dialog box, Options dialog box, and Object palette.

**To control widow and orphan lines for new forms**

Use the Formatting page in the Options dialog box to control widow and orphan lines for new forms. The widow and orphan functionality can prevent a single line of text in all text and text field objects in the form from being separated from the rest of a paragraph when page breaks are introduced. The widow and orphan control ensures that a minimum of two lines from the paragraph are kept together.

**To control widow and orphan lines for new forms:**

1. Select Tools > Options.
2. Click Formatting and select Widow and Orphan Control.
3. Click OK.

**To control widow and orphan lines for a form**

Use the Formatting tab in the Form Properties dialog box to control window and orphan lines in all text and text field objects for individual forms.

The default setting that initially appears in the Form Properties dialog box is inherited from the Options dialog box on the Formatting page. The default setting automatically applies to all new forms. If you change the default setting in the Form Properties dialog box, the new setting overrides the setting in the Options dialog box for the current form. The Widow and Orphan Control option in the Options dialog box does not change.

**To control widow and orphan lines for an individual form:**

1. Select File > Form Properties, Select Edit > Form Properties.
2. Click the Formatting tab and select Widow and Orphan Control.
3. Click OK.

**To remove widow and orphan control from a form**

You can remove widow and orphan control from the text and text field objects.
To remove widow and orphan control from a form:
2. Click the Formatting tab and deselect Widow and Orphan Control.
3. Click OK.

Allowing page breaks within a text field
The Allow Page Break Within Content option is enabled by default for a text field if the parent subform allows page breaks.

Here are a few things to consider when allowing page breaks within a text field:
- The Allow Page Break Within Content option is available only if the parent object allows page breaks.
- The caption text of a text field object will not break between pages when Top or Bottom is selected for the position for the caption in the Field tab of the Object palette.
- Page breaks are not supported in rotated text field objects.
- A page break is not allowed when an object that can accept a page break is positioned beside an object that cannot.

To allow page breaks within a text field
In the Object palette, click the Field tab and select Allow Page Breaks Within Content.

To allow page breaks within a text object
The Allow Page Break Within Content option is enabled for a text object if the parent subform allows page breaks. This option is deselected by default.

Here are a few things to consider when allowing page breaks within a text object:
- The Allow Page Break Within Content option is enabled for a text object only when the parent object allows page breaks.
- Page breaks are not supported in rotated text objects.
- The Allow Page Breaks Within Content option is disabled for floating fields. Instead, select the Allow Page Break Within Content option for the text object referencing the floating field.
- A page break is not allowed when an object that can accept a page break is positioned beside an object that cannot.
- When the Make Fixed-Size Text Objects Auto-Fit While Editing option (in Tools > Options > Workspace) or the Expand to Fit options (in the Layout palette) are deselected, the boilerplate text does not expand at design time, which means that page breaks are prevented.

To allow page breaks within a text object
In the Object palette, click the Draw tab and select Allow Page Breaks Within Content.
Typography

Use leaders to guide the reader from one piece of information to another across a page, such as in a table of contents, price list, or invoice. You can add leaders to a line or block of text in text objects and in the caption area of objects such as text fields, decimal fields, and numeric fields. When the text object or caption area is in edit mode, the Leader command is available in the Insert menu and the Context menu.

You can define and edit leader properties such as the end position, pattern, line thickness or dot spacing, and alignment. You can also adjust the positioning, size, and appearance of leaders by using the options in the Font and Paragraph palettes.

When you add a leader, Designer ES2 draws a leader from the insertion point to the end position that you specify. The leader end position is based on ruler increments. The default end position extends the leader to the right edge of the caption or static text object. You can only define or edit one leader at a time. All changes are applied immediately.

You can define different leaders for each paragraph of text. However, it is important to understand that a new paragraph inherits the leader properties from the previous paragraph. In other words, when you create a new paragraph, any leaders defined in the previous paragraph apply to the new paragraph.

**Note:** LiveCycle Designer ES2 can not anticipate the content of run-time data. If you intend to populate a form with rich text that contains leaders, you must set the target version to Acrobat 9 when you design the form.

**To add a leader**

1. Place the insertion point where you want the leader to start within the text object or caption area, and then select Insert > Leader.
2. In the Leader End Position box, type the ruler position where you want the leader to end.
3. In the Leader Pattern list, select the pattern to fill the leader.
4. In the Leader Dot Spacing or Leader Underline Weight list, select the amount of space to appear between dots or dashes, or select the thickness of the line.
5. Set the alignment for the leader by doing one of these actions:
   - To align the left side of the text that follows the leader (immediately before the first letter) with the specified end position, click Left Align Leader.
   - To align the right side of the text that follows the leader (immediately after the last letter) as the end position for the leader, click Right Align Leader.
6. Click anywhere in the Layout Editor to close the dialog box.

**To edit a leader**

1. Select the leader and edit it.
2. Click anywhere in the Layout Editor to close the dialog box.

**To delete a leader**

The Delete Leader command removes the selected leader (tab), the associated tab stops, as well as any leader properties, such as pattern, spacing, and alignment.

**Note:** The Delete Leader command is available on the Edit menu after you select a leader. However, if the Delete Leader command remains unavailable after you select a leader, this means that the leader is not associated with a tab stop. When this is the case, you can use the Delete key to remove the leader.

1. Select the leader to delete.
2. Select Edit > Delete Leader.
To delete tab stops

If you use the Delete key on the keyboard to delete a selected leader, only the tab character is removed. Any associated tab stops or leader properties such as pattern, spacing, and alignment are not removed.

To delete unused tab stops, you must use the Delete Tab Stops dialog box.

**Note:** The Delete Tab Stops command is available on the Edit menu when there are one or more tab stops in the current paragraph. If there are two or more paragraphs within a selection, the dialog is unavailable.

1. Select Edit > Delete Tab Stops.
2. Complete one of the following actions:
   - To delete selected tab stops, select one or more tab stops from the list, and then click Clear.
   - To delete all tab stops, click Clear All.
3. Click OK.
Tabbing Order

Many users use the Tab key to move between fields and buttons in a form instead of using the mouse. Designer ES2 lets you set the tabbing order between objects in a form.

Tabbing order is important for interactive forms and forms that have a fixed layout. For interactive forms, the tabbing order affects the end user’s experience when filling the form.

For both interactive and non-interactive forms, tabbing order is critical if your forms need to be accessible to users with vision or mobility impairments. These users typically do not use a mouse to navigate through the form, so they depend on the keyboard keys and a good tabbing order sequence to ensure that they have full access to all the fields on the form.

Accessible forms require a tabbing order, whether the form is interactive or designed for print. Additionally, a screen reader will read the form in geographic order, which in Designer ES2 is set when you use the default tabbing order.

Designer ES2 automatically sets a default tabbing order for each form. This tabbing order can be easily changed to better reflect the logical flow of the form and to accommodate particular user requirements.

How the default tabbing order works

The default tabbing order for objects in a form is from left to right, top to bottom, starting from the upper-left corner. Tabbing order respects the existence of subforms, radio buttons, and content areas. For example, if two subforms exist side-by-side, and each subform contains a number of field objects, the tabbing sequence will go through the fields in the first subform before moving on to the next.

Note: Designer ES2 does not include circle, line, or rectangle objects in the tabbing order.

Tabbing order is also determined by the vertical position of objects on a page and its master page.

The tabbing order starts from the object with the smallest vertical coordinate and ends with the object with the largest vertical coordinate, regardless of whether the object is on the body or master page.

For objects that contain objects, such as content areas, all child objects are tabbed through before tabbing to the next higher-level object.

The following list provides an example of the tabbing order for objects on body and master pages:

- Image object on the master page with a vertical coordinate of 1.
- Content area object on the page with a vertical coordinate of 4. All objects in the content area are tabbed through before tabbing to the subform object on the master page.
- Subform object on the master page with a vertical coordinate of 10. All objects in the content area are tabbed through before tabbing to the text object on the master page.
- Text object on the master page with a vertical coordinate of 12.

Because the tabbing order is important, ensure that you position objects precisely on the form, relative to each other. For example, you can position and size an object using its coordinates and you can snap objects to points on a grid.

You can change the default tabbing order if you require a different sequence in your form. For example, you may want to change the tabbing order to move through objects in a column, from top to bottom, and then left to right.

Note: In Acrobat 6.0.2, tabbing to a group of radio buttons makes the upper-left radio button active. Use the Tab key to move through the radio buttons. In Acrobat 7.0.5 and later, the selected radio button becomes active. Use the arrow keys to move between the radio buttons in a group and the Tab key to move out of the group.
Viewing the default tabbing order

One of the final tasks to do when creating a form is to set the tabbing order. Before you do this task, make sure you have finished laying out the form design so that all the objects are in their correct position on the form. First, you should examine the default tabbing order, which is created automatically by Designer ES2.

When you design or open a form in Designer ES2, the tabbing order is not visible. To see the tabbing order for the form, select View > Show Tab Order or click Show Order on the Tab Order palette.

**Note:** If the Tab Order palette is not visible, select Window > Tab Order.

**Note:** If the Tab Order palette is not visible, select Palettes > Tab Order.

The tabbing order is displayed on the form as a series of consecutive numbers inside colored shapes:

- Numbers inside a gray circle indicates the default tabbing order for the objects in the content area.
- Numbers inside a green circle indicate the tabbing order for master page objects.
- Numbers inside a lavender square indicate the tabbing order for the objects inside a fragment.

![Form design with numbers inside colored shapes](image)

Examine the default tabbing order carefully to determine whether it suits your particular requirements or whether you need to change it. Even if you modify the tabbing order, you can quickly return to the default tabbing order by selecting the Automatic option in the Tab Order palette.

**Note:** While viewing the tabbing order, you cannot edit any parts of the form.

To hide the tabbing order and return to normal form-editing mode

Perform one of the following actions:

- Select View > Hide Tab Order
- Click Hide Order in the Tab Order palette
- Open a different palette, such as the Hierarchy palette.

Using the Tab Order palette

The Tab Order palette provides an alternative view of the tabbing order on the form. It shows all the objects on the form as a numbered list, where each number represents the position of the object in the tabbing flow. The palette is also where you modify the tab order, if required.

**To open the Tab Order palette, select Window > Tab Order.**

**To open the Tab Order palette, select Palettes > Tab Order.**
The Tab Order palette may show the following visual markers in the list:

- A gray bar marks each page of the form. The tabbing order on each page starts with the number 1.
- The letter M inside a green circle indicates master page objects (visible only when viewing the form on the Design View tab).
- A range of numbers indicates objects within a fragment.
- A yellow background indicates the currently selected item.
- A lock icon beside the first object on the page indicates that the object cannot be moved within the tabbing order (visible only when viewing the form on the Master Pages tab).

The list shows exactly the same tabbing order numbers as the numbers displayed on the form itself.

The difference is that the numbers displayed on the form are for information purpose only, whereas the numbers on the list can be changed to modify the tabbing order.

In the Tab Order palette, you can also show or hide the tabbing order on the form and switch between the default and custom tabbing order.

**Changing the tabbing order**

After examining the default tabbing order, you may decide that you need a different sequence for some of the objects on the form. For example, if you have two groups of address fields situated side-by-side, you may want to tab through the fields in the first group before moving to the second group.

You change the position of an object in the tabbing order by moving the object up or down in the Tab Order palette list. You can move a single object or a group of objects. When you move the object to a new place in the order, Designer ES2 reassigns the numbers to accommodate the object in its new place.

For example, you have four objects on the page, which are arranged in the following layout.
Therefore, their default tabbing order will be as shown in this illustration.

![Tabbing Order Illustration]

You may want to change this tabbing order to a more logical one, such as First Name, Last Name, Telephone, Email. All you have to do is move the Last Name object one position up in the list. The tabbing order numbers are reassigned to reflect this move.

Although the tabbing order for the objects on the master page and the objects inside a fragment are displayed on the form, you cannot change the order for these objects on the Design View tab. For the master page objects, click the Master Pages tab and customize the tabbing order. For the fragments, open the required fragment and change the order inside the fragment.

If your form has more than one page, the tabbing order for each page starts at the number 1, and you can change the order only inside each page.

Before you change the tabbing order, you should prepare your working area in the following way:

1. Select Window > Tab Order. The Tab Order palette appears on the left side of the working area.
2. Select Palettes > Tab Order. The Tab Order palette appears on the left side of the working area.
3. Click Show Order and then select Custom.

The tabbing order numbers displayed on the form are now inside blue squares. This square visually indicates that you are now in custom tabbing mode and can change the position of one or more objects in the tabbing sequence.

To change the tabbing order for a single object using the mouse
You change the tabbing order by selecting the objects and changing their positions in the Tab Order palette list.

To change the order using drag-and-drop
- Drag the selected object up or down the list and place it at the required location. A black handle marks your current position within the list before you place the object.

To change the order using the arrow buttons
- In the Tab Order palette, click the up or down arrow buttons until the selected object is placed in the correct position.

To change the order using the menu
- In the Tab Order palette menu, select Move Up or Move Down.

To change the order by editing the number
- In the Tab Order palette list, click the selected object to make the number listed beside the object name editable. Then, type the new number indicating the new position of the object in the tabbing order and press Enter.
To change the order using copy and paste

- Select Copy from the Tab Order palette menu and, in the list, select the object above which to place the object you are moving, and then select Paste from the menu.

To change the tabbing order for a single object using keyboard shortcuts

*Note: You must use the mouse pointer in conjunction with the keyboard to change the tabbing order by using keyboard shortcuts.*

You change the tabbing order by selecting the objects and changing their positions in the Tab Order palette list.

To change the order using the menu

- Press Ctrl+Up Arrow or Ctrl+Down Arrow to move the object one position up or down in the list.

To change the order by editing the number

- Press F2 to make the number listed beside the object name editable. Then type the new number that indicates the new position of the object in the tabbing order and press Enter.

To change the order using copy and paste

- Press Ctrl+C to copy the selected object to the clipboard. Then press the Up Arrow or Down Arrow repeatedly until you highlight the object above which you want to place the object, and press Ctrl+V to paste the object from the clipboard.

Changing the tabbing order for a group of objects

Instead of moving individual objects to their new location within the tabbing order, you can move a group of objects all at once. This method works only for the objects that are positioned in sequence in the list of objects.

If you have a selection of objects that are scattered throughout the list, you must make them sequential first and then move the group.

After you select a group of sequentially ordered objects, you move the group to a new location in the tabbing order by using the same methods you used for moving a single object.

To select a sequential group of objects using the mouse

1. In the Tab Order palette list, click the first object in the sequence.
2. Press Shift and click the last object in the sequence.

To select a sequential group of objects using the keyboard

1. In the Tab Order palette list, click the first object in the sequence.
2. Press Shift+Down Arrow or Shift+Up Arrow until all objects in the sequence are selected.

To change a group of individually selected objects to a sequential group

1. In the Tab Order palette list, click the first object and then press Ctrl and click each object that you want to select.
2. In the palette menu, select Make Sequential. The selected objects are moved together into a sequentially ordered group.

Changing the tabbing order on master pages

Although the tabbing order for the objects that are located on a master page is displayed on the Design View tab, you can change the order for these objects only on the Master Pages tab.
The objects on the master page are placed in two areas, one above the content area and one below the content area. The tabbing order is set for objects within each of these two areas. Consequently, you can only change the tabbing order within each area.

**Note:** The upper-left object on each master page is locked in the first tabbing position and cannot be moved.

When you change the tabbing order on a master page, any pages in your form that are using this master page are affected by the change.

---

**To change the tabbing order in fragments**

If you use fragment references in your form, the tabbing order inside a fragment is visible when viewing the order for the form. To change the tabbing order inside a fragment, you must open the fragment source file for editing, make the change, and save the file. Any forms that use this fragment are affected by this change.

1. Select View > Hide Tab Order.
2. Select the fragment reference.
4. Select View > Show Tab Order.
5. Change the tabbing order as required.
6. Save and close the fragment source file.

---

**To return to the default tabbing order**

If you decide that you do not want the customized tabbing order on your form, you can quickly return to the automatic (default) tabbing order. You will lose any changes made to the tabbing order.

1. On the Tab Order palette, select Automatic.
2. In the message box, click Yes to acknowledge that the custom tab order will be removed.

---

**Excluding text and image objects from the tabbing order**

You can exclude text and image objects from the tabbing order to improve usability and flow of the tabbing on the form.

For example, on the form below, the user will need to tab through the title of the form (Purchase Order) before reaching the P.O. Date field.

If the text and image objects are excluded from the tabbing order, the tabbing flow on this form will change to include only the fields that require user input or action.
Excluding text and image objects from the tabbing order is not recommended if one or more of the following is true for your form:

- Your form will be used with a screen reader.
- Your form contains hyperlinks inside the text objects.

You can exclude the text and image objects for both the default and the custom tabbing order.

**To exclude text and image objects**

- Do one of the following actions:
  - Select Tools > Options > Tab Order and, in the Tab Order panel, select Only Show Tab Order For Fields.
  - In the Tab Order palette menu, select Show Fields Only.

**To view the tabbing order using visual aids**

On a complex and busy form, it may be difficult to see how the tabbing flows from one object to the next. You can use visual aids to help you see the tabbing flow on the form.

With the visual aids turned on, when you hover the pointer over the object, blue arrows show the tabbing flow for the two preceding and two following objects in the tabbing order.

- Do one of the following actions:
  - Select Tools > Options > Tab Order and, in the Tab Order panel, select Display Additional Visual Aids For Tab Order.
  - In the Tab Order palette menu, select Show Visual Aids.
Using Text RTL

Text objects present read-only text that users cannot edit. You can use text objects to do these tasks:

- Label an area in the form
- Provide instructions for filling out the form
- Include a header and footer

After you add a text object to the form design, you can define these properties:

- Define the text as visible, invisible, or hidden.
- Specify a locale for the text.
- Type and style the text.
- Allow page breaks within the content of the field.
- Keep a text object with the next object in the document when a page break is introduced.
- Apply font properties to all of the text or selected portions of the text.

In addition, you can insert placeholders, such as floating text fields, and the values of run-time properties, such as page count, current page, and current date/time, into text objects.

The text object has the Make Fixed-Sized Text Objects Auto-fit While Editing option (Tools > Options > Workspace) selected by default. Therefore, the object automatically resizes at design time to accommodate the text you enter and the property settings you change, such as margins, borders, and font type and size.

To type characters into a text object

You can type characters into the object after you add the object to the form.

By default, the Make Fixed-Size Text Objects Auto-Fit While Editing option on the Workspace panel of the Tools > Options dialog is selected. This option allows the width and height of text objects to expand as you type characters. To expand only either the width or height of a text object, deselect Make Fixed-Size Text Objects Auto-Fit While Editing and then select the appropriate Expand To Fit option on the Layout palette.

1. Select Edit and ensure that the Lock Text and Lock Static Objects commands are not selected.
2. Double-click the text object and type the characters you want to add.

To insert a floating text field into a text object

You can embed a floating text field in a text object. Floating text fields support the merging of different text values within text objects. For example, you could embed a floating text field to generate a customer name in the salutation of a form letter.

A floating text field does not have a caption and can be edited as if it were a single character of text. The properties of the floating text field must be defined separately in addition to the properties of the text object.

*Note:* When merging floating fields with data, keep in mind that floating fields cannot span pages.

1. In the text object, place the cursor where you want the floating text field to be merged.
2. Select Insert > Floating Field.
3. Select the floating text field and edit its properties.

To insert the value of a run-time property into a text object

You can insert the values of run-time properties into a text object. When the form is viewed, the actual value is inserted automatically.

The inserted value can have a different font, font size, and font style than the text object.

1. In the text object, place the cursor where you want the value of the run-time property to be merged.
2 Select Insert > [run-time property value].
   • To insert the current page number of the finished form, select Current Page Number.
   • To insert the total number of pages making up the finished, select Number of Pages.
   • To insert the current date/time, select Current Date/Time.
   • To insert the value of the locale setting for the application processing the form, select Viewer Locale.
   • To insert the name of the application processing the form, select Viewer Name.
   • To insert the version number of the application processing the form, select Viewer Version.

3 Select the run-time property value and edit its properties.

To merge text objects
When you import a PDF file into Designer ES2, depending on the import options you choose, title and instruction text may be broken up into multiple text objects. This import behavior makes the text difficult to edit because individual words or individual letters may be broken into separate text objects. If you need to edit the text, you can merge text objects together to make the text easier to update.

1 Select the text objects that you want to merge.
   If you also select an object of a different type, it will not be included in the merged object.

2 Select Layout > Merge Selected Text Objects.
   All selected text objects combine into a single object.

To create a caption by merging a text object and a field object
You can create a caption for a field object that does not have a caption by merging it with a text object.

For example, if you have a text object above a text field whose caption is set to none, you can merge the two objects to create one field. The text from the text object becomes the caption for the text field.

Before Merging: A. Text object B. Text field

After Merging: A. Text object

Before you perform this task, you must ensure the following settings are in effect:
   • Ensure you have one text object.
   • Ensure you have one field object whose caption is set to none in the Layout palette or the caption is empty.

The field object can be one of the following objects:
   • Check Box
   • Date/Time Field
• Decimal Field
• Signature Field
• Drop-down List
• Image Field
• List Box
• Numeric Field
• Password Field
• Radio Button
• Text Field

You can merge only one text object to one field object.

1. Select the text object that you want to become the caption for the field object.
2. Ctrl+click or Shift+click to select the field object that you want to merge with the text object.
   The field object must have its caption set to none in the Layout palette or the caption must be empty.
3. Select Layout > Merge as Caption.
   The new object’s caption becomes the text from the text object. The new object’s size and position matches the total boundaries of the
   merged objects.
   You may have to edit the font of the caption and the value after you merge the objects.

To insert language-specific numbers in text or captions
Most regions of the world use a numbering system that is often called Roman or European (1, 2, 3, ...). However, the numbering system in
Arabic is different; it is derived from Hindi or Indic numbers.

Although Arabic letters are written from right-to-left, numbers in Arabic are written from left-to-right. This numbering system is often
called Arabic-Indic.

You can insert language-specific numbers in text or captions if your locale, such as Arabic (Egypt), supports it. The following locales support
this feature:
• Arabic (all except Algeria, Morocco, and Tunisia)
• Thai (Thailand Traditional)
The options described in this Help topic are available only if support for the appropriate language is enabled through Microsoft Office
Language Settings.
1. Select the text.
2. Right-click and ensure the language-specific Use Digits command is selected.
   If this menu item is not there, ensure the locale set for this object supports this feature.
3. Type the numbers.
   This example is of a caption that shows three types of language-specific numbers.

A. Arabic (Egypt) B. English (USA) C. Thai (Thailand Traditional)
To allow page breaks within a text object

The Allow Page Break Within Content option is enabled for a text object if the parent subform allows page breaks. This option is deselected by default.

Here are a few things to consider when allowing page breaks within a text object:

- The Allow Page Break Within Content option is enabled for a text object only when the parent object allows page breaks.
- Page breaks are not supported in rotated text objects.
- The Allow Page Breaks Within Content option is disabled for floating fields. Instead, select the Allow Page Break Within Content option for the text object referencing the floating field.
- A page break is not allowed when an object that can accept a page break is positioned beside an object that cannot.
- When the Make Fixed-Size Text Objects Auto-Fit While Editing option (in Tools > Options > Workspace) or the Expand to Fit options (in the Layout palette) are deselected, the boilerplate text does not expand at design time, which means that page breaks are prevented.

To allow page breaks within a text object

 arrivou em dieta, click the Draw tab and select Allow Page Breaks Within Content.

To keep a text object with the next object in the form

The Keep With Next option is enabled for a text field if the parent object allows page breaks and is a flowed container.

The Keep With Next option is disabled for floating fields. Instead, select the Keep With Next option for the text object referencing the floating field.

 arrivou em dieta, click the Draw tab and select Keep With Next.
Using Barcodes

Businesses use barcodes extensively, particularly for inventory control. Barcodes can be used to identify forms, but they are often printed on adhesive paper to create labels for inventory purposes.

Designer ES2 supports two types of barcodes:

- **Hardware barcodes** Only use when the form is being printed directly to the printer from the server. Because a printer is required to print them, Designer ES2 uses a placeholder to represent hardware barcodes in the form.

- **Software barcodes** Can be drawn by Designer ES2 and are visible in Acrobat and Adobe Reader. They can be printed on any general-purpose printer.

Some types of barcodes can hold arbitrary binary data. Others are limited to a particular set of characters or codes. It is the responsibility of the form author to ensure that the data is appropriate for the barcode, for example, by imposing a validation on the field.

Barcode formats

Designer ES2 provides a variety of barcode formats that you can work with. The supported barcode formats are listed in the Barcodes category of the Object Library palette. If you use a barcode that requires a specific type of printer, Designer ES2 represents the barcode as a shaded rectangle in the form.

**Note:** Interactive barcodes that can accept user input are only supported for PDF forms that are filled in Acrobat 7.0.5 or Adobe Reader 7.0.5.

Designer ES2 also supports the two-dimensional paper forms barcode.

Barcode position and appearance

The function of a barcode is to be read by a specialized piece of hardware called a barcode reader. Since the barcode is intended to be read by a machine, its appearance is usually strictly constrained. For example, for a particular type of barcode, the bars may have to be a particular height and distance apart. In addition it is common for a barcode to require a minimum amount of white space around it (the quiet zone) and a particular range of distances from a designated edge of the page.

Designer ES2 does nothing to express or enforce positioning or quiet zone requirements. It is up to the form creator to ensure that these requirements are met.

Barcode properties

After you add a barcode to the form design, you can manipulate the object's properties in the Field, Value, and Binding tabs of the Object palette. You can define these properties:

- Text position
- Length of the data
- Any additional properties supported by the barcode (for example, optional checksum capabilities, and text positioning and embedding)
- Presence of the barcode as visible, invisible, or hidden
- Binding method for storing and retrieving bound data

**To define barcode field properties**

To define the properties of a barcode, you must first select the barcode and set basic characteristics in the Field tab of the Object palette. The number and type of barcode properties vary from one barcode to another.

1. Select the barcode.
2. In the Object palette, click the Value tab and then type the barcode text in the Default box.
3. Click the Field tab and, in the Location box, select the placement of the text.

The PDF 417 format does not support text positioning, and the EAN8, EAN13, and UPC-A formats support the Below Embedded option only.
4 In the Value column, do one or more of the following actions:
   • In the Data Length box, type the length of the data. If you are defining an MSI barcode, the data length must be a value between 1 and 14. The MSI, UPC-E, UPC EAN2, UPC EAN5, US Postal Zip-5, US Postal DPBC, and US Postal Standard barcode formats have fixed data lengths that cannot be changed.
   • If a Checksum box is displayed, enable or disable the checksum. To enable the default checksum, select Auto. To disable the checksum, select None. If you are defining an MSI barcode, select one of the available checksum methods.
   • If an End Char or Start Char box is displayed, type the end character, start character, or both. If you are defining a Codabar barcode, the valid end and start characters are A, B, C, D, a, b, c, d, *, N, T, E, n, t, and e. If you are defining a Code 49 barcode, the valid start characters are A, 1, 2, 3, 4, and 5.
   • If a Wide/Narrow Ratio box is displayed, type a wide-narrow-ratio value. For Codabar, Code 2 of 5 (Industrial, Interleaved, and Matrix), and Code 3 of 9, the wide-narrow-ratio value must be a value from 2.2 to 3.0. For Code 11, Logmars, MSI, and Plessey barcodes, the value must be a value from 2.0 to 3.0.

To set the default value for the barcode
   ✷ In the Object palette, click the Value tab and then type a default value in the Default box.

To control how a barcode obtains data
   To define the properties of a barcode, you must first select the barcode. Set properties that control how the barcode obtains data in the Value tab of the Object palette.

You can dynamically populate a validation pattern or script message with a value from a data source. This allows you to ensure users enter the correct value in the field.

1 In the Object palette, click the Value tab and then select one of these options from the Type list:
   • To allow users to choose to enter data or not, select User Entered - Optional.
   • To prompt users to enter data and make the field recommended, select User Entered - Recommended and type a custom message in the Empty Message box.
   • To prompt users to enter data and make the field required, select User Entered - Required and type a custom message in the Empty Message box.
   • To make the field read only and display a data value that is calculated and displayed through an attached script, select Calculated - Read Only. Users cannot edit the calculated value.
   • To make the field editable and display a data value that is calculated and displayed through an attached script, select Calculated - User Can Override. Users can edit the value if the calculation script has been written to accept the input. If a user does edit the calculated value, the custom message you specify in theOverride Message box appears.
   • To make the field read only and display a data value that is merged or calculated and displayed at run time, select Read Only. Users cannot edit the value.

2 If the value is recommended or required, type a prompt in the Empty Message box.
3 If the value will be calculated, attach the calculation script to the object by using the Script Editor.
4 (Optional) If a calculated value can be overridden, type a message into the Override Message box.

To define custom data-binding properties for a barcode
   Binding options enable you to build a form that captures data for enterprise infrastructures and/or use an external data source to populate a form at run time. Set data-binding properties in the Binding tab of the Object palette.

1 Select the barcode.
2 Enable the form to connect to the data source when the form is opened.
3 Bind the barcode to its corresponding data node.
Encoding non-printing characters in barcode data

You can encode hidden, non-printing characters between the data in Code 128 barcodes. For example, you can encode the characters that represent the prefix and the delimiter as part of the Code 128 barcode standard. The application that reads the barcode then decodes the meaning of these characters.

Some characters are reserved for use as non-printing characters in Code 128 barcodes. To encode these characters in a barcode, use the following mechanism in Designer ES2.

<table>
<thead>
<tr>
<th>Code 128 non-printing characters</th>
<th>Use these values in Designer ES2</th>
</tr>
</thead>
<tbody>
<tr>
<td>FNC1</td>
<td>[F1]</td>
</tr>
<tr>
<td>FNC2</td>
<td>[F2]</td>
</tr>
<tr>
<td>FNC3</td>
<td>[F3]</td>
</tr>
<tr>
<td>SHIFT</td>
<td>[SH]</td>
</tr>
<tr>
<td>Change to Subset A</td>
<td>[CA]</td>
</tr>
<tr>
<td>Change to Subset B</td>
<td>[CB]</td>
</tr>
<tr>
<td>Change to Subset C</td>
<td>[CC]</td>
</tr>
<tr>
<td>Start in Subset A</td>
<td>[SA]</td>
</tr>
<tr>
<td>Start in Subset B</td>
<td>[SB]</td>
</tr>
<tr>
<td>Start in Subset C</td>
<td>[SC]</td>
</tr>
</tbody>
</table>

For example, to encode FNC1 in a Code 128 barcode, insert [F1] as follows: 00[F1]12345[F1]67890.
**Strict Scoping**

Although FormCalc and JavaScript are intended for two different types of users, there is some overlap between the types of built-in functions they offer. The following table lists all available FormCalc functions and whether a comparable function exists within JavaScript.

<table>
<thead>
<tr>
<th>FormCalc function</th>
<th>Description</th>
<th>JavaScript method equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs(n1)</td>
<td>Returns the absolute value of a numeric value or expression.</td>
<td>Math.abs(n1)</td>
</tr>
<tr>
<td>Apr(n1, n2, n3)</td>
<td>Returns the annual percentage rate for a loan.</td>
<td>None</td>
</tr>
<tr>
<td>At(s1, s2)</td>
<td>Locates the starting character position of a string within another string.</td>
<td>String.search(s1)</td>
</tr>
<tr>
<td>Avg[n1 [, n2... ] }</td>
<td>Evaluates a set of number values and/or expressions and returns the average of the non-null elements contained within that set.</td>
<td>None</td>
</tr>
<tr>
<td>Ceil(n1)</td>
<td>Returns the whole number greater than or equal to a given number.</td>
<td>Math.ceil(n1)</td>
</tr>
<tr>
<td>Choose(n1, s1 [, s2... ] }</td>
<td>Selects a value from a given set of parameters.</td>
<td>None</td>
</tr>
<tr>
<td>Concat(s1 [, s2... ] )</td>
<td>Returns the concatenation of two or more strings.</td>
<td>String.concat(s1, s2 [, s3 ... ])}</td>
</tr>
<tr>
<td>Count[n1 [, n2...] ]</td>
<td>Evaluates a set of values and/or expressions and returns the number of non-null elements contained within the set.</td>
<td>None</td>
</tr>
<tr>
<td>CTerm(n1, n2, n3)</td>
<td>Returns the number of periods needed for an investment earning a fixed, but compounded, interest rate to grow to a future value.</td>
<td>None</td>
</tr>
<tr>
<td>Date()</td>
<td>Returns the current system date as the number of days since the epoch.</td>
<td>Date.getDate()</td>
</tr>
<tr>
<td>Date2Num(d1 [, f1 [, k1 ] ] )</td>
<td>Returns the number of days since the epoch, given a date string.</td>
<td>The JavaScript Date object does not use the epoch as a reference point.</td>
</tr>
<tr>
<td>DateFmt([ n1 [, k1 ] ] )</td>
<td>Returns a date format string, given a date format style.</td>
<td>None</td>
</tr>
<tr>
<td>Decode(s1 [, s2 ] )</td>
<td>Returns the decoded version of a given string.</td>
<td>Partial support</td>
</tr>
<tr>
<td>Encode(s1 [, s2 ] )</td>
<td>Returns the encoded version of a given string.</td>
<td>Partial support</td>
</tr>
<tr>
<td>Eval()</td>
<td>Returns the value of a given form calculation.</td>
<td>eval(s1)</td>
</tr>
<tr>
<td>Exists(v1)</td>
<td>Determines whether the given parameter is a valid reference syntax to an existing object.</td>
<td>None</td>
</tr>
<tr>
<td>Floor(n1)</td>
<td>Returns the largest whole number that is less than or equal to the given value.</td>
<td>Math.floor(n1)</td>
</tr>
<tr>
<td>Format(s1, s2)</td>
<td>Formats the given data according to the specified picture format string.</td>
<td>None</td>
</tr>
<tr>
<td>FV(n1, n2, n3)</td>
<td>Returns the future value of consistent payment amounts made at regular intervals at a constant interest rate.</td>
<td>None</td>
</tr>
<tr>
<td>FormCalc function</td>
<td>Description</td>
<td>JavaScript method equivalent</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Get(s1)</td>
<td>Downloads the contents of the given URL.</td>
<td>None</td>
</tr>
<tr>
<td>HasValue(v1)</td>
<td>Determines whether the given parameter is a valid reference syntax with a non-null, non-empty, or non-blank value.</td>
<td>None</td>
</tr>
<tr>
<td>IPmt(n1, n2, n3, n4, n5)</td>
<td>Returns the amount of interest paid on a loan over a set time.</td>
<td>None</td>
</tr>
<tr>
<td>IsoDate2Num(d1)</td>
<td>Returns the number of days since the epoch, given a valid date string.</td>
<td>None</td>
</tr>
<tr>
<td>IsoTime2Num(d1)</td>
<td>Returns the number of milliseconds since the epoch, given a valid time string.</td>
<td>None</td>
</tr>
<tr>
<td>Left(s1, n1)</td>
<td>Extracts a specified number of characters from a string, starting with the first character on the left.</td>
<td>String.substring(n1, n2)</td>
</tr>
<tr>
<td>Len(s1)</td>
<td>Returns the number of characters in a given string.</td>
<td>String.length</td>
</tr>
<tr>
<td>LocalDateFmt[n1 [, k1 ] ]</td>
<td>Returns a localized date format string, given a date format style.</td>
<td>None</td>
</tr>
<tr>
<td>LocalTimeFmt[n1 [, k1 ] ]</td>
<td>Returns a localized time format string, given a time format style.</td>
<td>None</td>
</tr>
<tr>
<td>Lower[s1 [, k1 ] ]</td>
<td>Converts all uppercase characters within a specified string to lowercase characters.</td>
<td>String.toLowerCase(s1)</td>
</tr>
<tr>
<td>Ltrim[s1]</td>
<td>Returns a string with all leading white space characters removed.</td>
<td>None</td>
</tr>
<tr>
<td>Max[n1 [, n2... ] ]</td>
<td>Returns the maximum value of the non-null elements in the given set of numbers.</td>
<td>Math.max(n1, n2)</td>
</tr>
<tr>
<td>Min[n1 [, n2... ] ]</td>
<td>Returns the minimum value of the non-null elements of the given set of numbers.</td>
<td>Math.min(n1, n2)</td>
</tr>
<tr>
<td>Mod[n1, n2]</td>
<td>Returns the modulus of one number divided by another.</td>
<td>Use the modulo (%) operator.</td>
</tr>
<tr>
<td>NPV[n1, n2 [, ... ] ]</td>
<td>Returns the net present value of an investment based on a discount rate and a series of periodic future cash flows.</td>
<td>None</td>
</tr>
<tr>
<td>Num2Date[n1[ , f1 [, k1 ] ] ]</td>
<td>Returns a date string given a number of days since the epoch.</td>
<td>None</td>
</tr>
<tr>
<td>Num2GMTTime[n1 [, f1 [, k1 ] ] ]</td>
<td>Returns a GMT time string given a number of milliseconds from the epoch.</td>
<td>None</td>
</tr>
<tr>
<td>Num2Time[n1 [, f1 [, k1 ] ] ]</td>
<td>Returns a time string given a number of milliseconds from the epoch.</td>
<td>None</td>
</tr>
<tr>
<td>Oneof[s1, s2 [, s3... ] ]</td>
<td>Returns true (1) if a value is in a given set and false (0) if it is not.</td>
<td>None</td>
</tr>
<tr>
<td>Parse[s1, s2]</td>
<td>Analyzes the given data according to the given picture format.</td>
<td>None</td>
</tr>
<tr>
<td>Pmt[n1, n2, n3]</td>
<td>Returns the payment for a loan based on constant payments and a constant interest rate.</td>
<td>None</td>
</tr>
<tr>
<td>FormCalc function</td>
<td>Description</td>
<td>JavaScript method equivalent</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Post(s1, s2 [, s3 [, s4 [, s5 ] ] ] )</td>
<td>Posts the given data to the specified URL.</td>
<td>None</td>
</tr>
<tr>
<td>P_fmt(n1, n2, n3, n4, n5)</td>
<td>Returns the amount of principal paid on a loan over a period of time.</td>
<td>None</td>
</tr>
<tr>
<td>Put(s1, s2 [, s3 ] )</td>
<td>Uploads the given data to the specified URL.</td>
<td>None</td>
</tr>
<tr>
<td>PV(n1, n2, n3)</td>
<td>Returns the present value of an investment of periodic constant payments at a constant interest rate.</td>
<td>None</td>
</tr>
<tr>
<td>Rate(n1, n2, n3)</td>
<td>Returns the compound interest rate per period required for an investment to grow from present to future value in a given period.</td>
<td>None</td>
</tr>
<tr>
<td>Ref()</td>
<td>Returns a reference to an existing object.</td>
<td>None</td>
</tr>
<tr>
<td>Replace(s1, s2 [, s3 ] )</td>
<td>Replaces all occurrences of one string with another within a specified string.</td>
<td>String.replace(s1, s2)</td>
</tr>
<tr>
<td>Right(s1, n1)</td>
<td>Extracts several characters from a given string, beginning with the last character on the right.</td>
<td>String.substring(n1, n2)</td>
</tr>
<tr>
<td>Round(n1 [, n2 ] )</td>
<td>Evaluates a given numeric value or expression and returns a number rounded to the given number of decimal places.</td>
<td>Math.round(n1)</td>
</tr>
<tr>
<td>Rtrim(s1)</td>
<td>Returns a string with all trailing white space characters removed.</td>
<td>None</td>
</tr>
<tr>
<td>You can use JavaScript regular expressions to perform this operation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Space(n1)</td>
<td>Returns a string consisting of a given number of blank spaces.</td>
<td>None</td>
</tr>
<tr>
<td>Str(n1 [, n2 [, n3 ] ] )</td>
<td>Converts a number to a character string. FormCalc formats the result to the specified width and rounds to the specified number of decimal places.</td>
<td>String(n1) or Number.toString(radix)</td>
</tr>
<tr>
<td>Stuff(s1, n1, s2 [, n2 ] )</td>
<td>Inserts a string into another string.</td>
<td>None</td>
</tr>
<tr>
<td>Substr(s1, n1, n2)</td>
<td>Extracts a portion of a given string.</td>
<td>String.substring(n1, n2)</td>
</tr>
<tr>
<td>Sum(n1 [, n2... ] )</td>
<td>Returns the sum of the non-null elements of a given set of numbers.</td>
<td>None</td>
</tr>
<tr>
<td>Term(n1, n2, n3)</td>
<td>Returns the number of periods required to reach a given future value from periodic constant payments into an interest-bearing account.</td>
<td>None</td>
</tr>
<tr>
<td>Time()</td>
<td>Returns the current system time as the number of milliseconds since the epoch.</td>
<td>Date.getTime()</td>
</tr>
<tr>
<td>The JavaScript Date object does not use the epoch as a reference point.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time2Num(d1 [, f1 [, k1 ] ] )</td>
<td>Returns the number of milliseconds since the epoch, given a time string.</td>
<td>None</td>
</tr>
<tr>
<td>TimeFmt([n1 [, k1 ] ] )</td>
<td>Returns a time format, given a time format style.</td>
<td>None</td>
</tr>
<tr>
<td>UnitType(s1)</td>
<td>Returns the units of a unitspan. A unitspan is a string consisting of a number followed by a unit name.</td>
<td>None</td>
</tr>
<tr>
<td>UnitValue(s2 [, s1 ] )</td>
<td>Returns the numeric value of a measurement with its associated unitspan after an optional unit conversion.</td>
<td>None</td>
</tr>
</tbody>
</table>
Using FormCalc

FormCalc is a simple yet powerful calculation language modeled on common spreadsheet software. Its purpose is to facilitate fast and efficient form design without requiring a knowledge of traditional scripting techniques or languages. Using several of the built-in functions, inexperienced FormCalc users can quickly create forms that save users from performing time-consuming calculations, validations, and other verifications. This way, you can create a basic set of rules for the form design that allows the resulting form to react according to the data it comes into contact with.

Within Designer ES2, FormCalc is the default scripting language in all scripting locations with JavaScript as the alternative.

**Caution:** If you are developing forms for use with a server-based process (for example, using Forms ES2), with the intent of rendering your forms in HTML, you should develop your calculations and scripts in JavaScript. FormCalc calculations are not valid in HTML browsers, and are removed prior to the form being rendered in HTML.

FormCalc treats each new line in the Script Editor as a new expression to evaluate.

### Using built-in functions

The built-in functions that comprise FormCalc cover a wide range of areas, including mathematics, dates and times, strings, finance, logic, and the web. These areas represent the types of functionality that usually occur in forms. The purpose of the functions is to provide quick and easy manipulation of form data in a useful way.

At the most basic level, a calculation can consist of only a single FormCalc function. However, a single FormCalc function can use other FormCalc functions as parameters.

### To attach a FormCalc function to an object

You can add a FormCalc function to any form design object that allows calculations and scripts, except for the script object.

1. Make sure that you have the multiline version of the Script Editor displayed in the Designer ES2 workspace.
2. Select a field on your form.
3. In the Show list, select the calculate event.
4. Click the Functions icon or F10 to display a list of FormCalc functions.
5. Select the appropriate function and press Enter.
6. Replace the default function syntax notation with your own set of values.
7. Click the Preview PDF tab to test the form.
Built-in function syntax

Each FormCalc function uses a specific syntax notation that you must follow in order for the function to execute correctly. This table describes, generally, the pieces of syntax notation.

<table>
<thead>
<tr>
<th>Syntax notation</th>
<th>Replacement values</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>A valid date string (for example, 03/15/1996)</td>
</tr>
<tr>
<td>f</td>
<td>A valid date format string (for example, MM/DD/YYYY)</td>
</tr>
<tr>
<td>k</td>
<td>A valid locale identifier (for example, fr_FR)</td>
</tr>
<tr>
<td>n</td>
<td>A valid numeric value. Notice that the range of valid values varies from function to function.</td>
</tr>
<tr>
<td>s</td>
<td>A valid unit of measurement (for example, &quot;in&quot; for inches).</td>
</tr>
<tr>
<td>v</td>
<td>A valid reference syntax.</td>
</tr>
<tr>
<td>n1, n2, n3</td>
<td>All values are required.</td>
</tr>
<tr>
<td>[ [ n , [ , k ] ] ]</td>
<td>No values are required, but you can choose to specify just n, or both n and k.</td>
</tr>
<tr>
<td>n1 [ , n2 ... ]</td>
<td>n1 is required, but you can choose to specify an unlimited number of additional values.</td>
</tr>
<tr>
<td>d [ , f [ , k ] ]</td>
<td>d is required, but you can choose to also specify f or both f and k.</td>
</tr>
</tbody>
</table>

Creating basic calculations

About basic calculations

Simple expressions are the most basic instances of scripting. These expressions do not involve using FormCalc built-in functions and are never more than a single line. Add simple expressions to the calculate event of a particular field or object so that the expression value can output onto your form.

Examples of basic calculations

These examples are all of simple expressions:

\[
\begin{align*}
&2 \\
&"abc" \\
&2 - 3 \ast 10 / 2 + 7 \\
&\text{(10 + 2) \ast (5 + 4)} \\
&0 \text{ and } 1 \text{ or } 2 > 1 \\
&2 < 3 \text{ not } 1 == 1
\end{align*}
\]

Each simple expression evaluates to a single value by following a traditional order of operations, even if the order is not always obvious from the expression syntax. For example, the following sets of expressions produce equivalent results.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Equivalent to</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;abc&quot;</td>
<td>&quot;abc&quot;</td>
<td>abc</td>
</tr>
<tr>
<td>2 - 3 \ast 10 / 2 + 7</td>
<td>2 - (3 \ast 10 / 2) + 7</td>
<td>-6</td>
</tr>
<tr>
<td>(10 + 2) \ast (5 + 4)</td>
<td>(10 + 2) \ast (5 + 4)</td>
<td>108</td>
</tr>
<tr>
<td>0 \text{ and } 1 \text{ or } 2 &gt; 1</td>
<td>(0 \text{ and } 1) \text{ or } (2 &gt; 1)</td>
<td>1 (true)</td>
</tr>
<tr>
<td>2 &lt; 3 \text{ not } 1 == 1</td>
<td>(2 &lt; 3) \text{ not } (1 == 1)</td>
<td>0 (false)</td>
</tr>
</tbody>
</table>
As implied in the previous table, all FormCalc operators carry a certain precedence when they appear within expressions. The following table illustrates this operator hierarchy.

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest</td>
<td>=</td>
</tr>
<tr>
<td></td>
<td>(Unary) -, +, not</td>
</tr>
<tr>
<td></td>
<td>* , /</td>
</tr>
<tr>
<td></td>
<td>+ , -</td>
</tr>
<tr>
<td></td>
<td>&lt; , &lt;= , &gt; , &gt;= , lt , le , gt , ge</td>
</tr>
<tr>
<td></td>
<td>== , &lt;&gt; , eq , ne</td>
</tr>
<tr>
<td></td>
<td>&amp; , and</td>
</tr>
<tr>
<td>Lowest</td>
<td></td>
</tr>
</tbody>
</table>

All the previous examples are valid, simple expressions that you can add to a form field or object that accepts calculations and scripts. For example, if you create a form in Designer ES2 with a single numeric field, add the following calculation to the calculate event in the Script Editor.
Then, when you click the Preview PDF tab to view the completed form, the value of the simple expression appears in the text field.

If the value does not appear in the preview, ensure that your simple expression appears in the calculate event of the form design object. Also, ensure that you installed Designer ES2 and Acrobat correctly.

**Using JavaScript**

To allow form designers more flexibility and scripting power, Designer ES2 supports the use of JavaScript version 1.6 or earlier in all situations that support scripting.

Form developers who are familiar with JavaScript can apply their existing expertise directly to Designer ES2. Designer ES2 provides several properties and methods that enhance JavaScript so that you to access field and object values. These properties and methods combined with the Designer ES2 reference syntax provide you with easy manipulation of form values and data.

*Note: The Script Editor does not provide syntax error checking for scripts created using JavaScript. In addition, statement completion options do not appear for standard JavaScript objects or methods.*

**Creating scripts using JavaScript**

Creating scripts in Designer ES2 using JavaScript is similar to creating JavaScript in other applications. You can take advantage of previous knowledge of JavaScript concepts, reuse JavaScript functions using the Designer ES2 script object, and take advantage of JavaScript language functionality.

However, notice that although previous JavaScript knowledge is transferable, to effectively use JavaScript on your form design, you must understand how to construct Designer ES2 reference syntax. Specifically, you must know how to correctly use the XML Form Object Model reference syntax to access objects on your form design.
Enforcing strict scoping rules in JavaScript

When working with JavaScript in forms, it is important to declare objects and variables within the scope they are intended. Globally declaring objects or variables unnecessarily can cause performance problems. Strict scoping was introduced in Designer 8.1 to improve the run time and memory usage of a form.

What is scope in JavaScript?
Scope works outwardly so that everything within curly brackets ({} can see outside them. However anything outside the curly brackets cannot access anything inside them.

In the following example, the first curly bracket opens the function scope and the second one closes it. Everything between the curly brackets is within the scope of foo ()

```javascript
nOutsideVar is outside
the scope of foo() -> var nOutsideVar = 2;
function foo()
This opens the function scope -> {
    // Everything between the two
    // curly braces is within the
    // scope of foo()
    var nFooVar = 4;
This closes the function scope -> }
```

The scope in the following example is valid because var nFooVar = nOutsideVar inside the curly brackets can see var nOutsideVar = 2 outside the curly brackets.

```javascript
var nOutsideVar = 2;
function foo()
{
    var nFooVar = nOutsideVar; // This is correct;
    // anything inside the
    // braces can see stuff
    // outside
}
```
In contrast, the following example shows an invalid scope because `var nOutsideVar = nFooVar` cannot access `var nFooVar = 4` within the curly brackets.

```javascript
function foo()
{
    var nFooVar = 4;
}

var nOutsideVar = nFooVar; // This is incorrect,
    // nOutsideVar cannot access
    // things declared within
    // foo()'s scope
```

Scope in scripting describes pieces of scripts that can access pieces. The pieces of script can be variables or functions.

**What is scope XML?**
Scope in a form design is about hierarchy. For example, to access the subform `inside` in the following XML source, you must type `outside.inside`.

```xml
<subform name="outside">
    <subform name="inside">
    ...
</subform>
</subform>
```

You do not type `inside.outside` because you must access the outermost subform first and drill inwards.

**SOM expressions and scope**
In forms that are targeted for Acrobat or Adobe Reader 8.1, SOM expressions are properly scoped as shown in this example:

```xml
<subform name="a">
    <subform name="b"/>
</subform>
```

In forms targeted for Acrobat or Adobe Reader 8.0, the SOM expression `a.b.a` returns the subform `a`. In forms targeted for Acrobat or Adobe Reader 8.1, the SOM expression `a.b.a` returns `null` because subform `b` does not have a child named `a`. In Acrobat or Adobe Reader 9.0 or later, the expression returns an error because `a` is not a valid child of `b`.

In Acrobat or Adobe Reader 8.1, functions and variable within a node's script do not become global (script objects being the exception) as shown in this example:

```xml
<field name="field1">
    event activity="initialize">
        <script contentType="application/x-javascript">
            // Function bar() is scoped to field1.initialize; nothing outside <event activity="initialize">
            scope can see inside here (in 8.1)
function bar()
{
  return "bar";
}
</script>
</event>
</field>

field name="field2">
  <event activity="click">
    <script contentType="application/x-javascript">
      field1.bar();
    </script>
  </event>
</field>

When you click field 2 in a form targeting Acrobat or Adobe Reader 8.0, the function bar() executes.

When you click field 2 in a form targeting Acrobat or Adobe Reader 8.0, the function bar() does not execute. The reason is because function bar() is available only from within the initialized script of field1.

**Scoping and script objects**

Script objects have global scope; therefore, anyone can access them from anywhere. If you have a method that you want both field1.initialize and field2.click to access, place it in a script object. Strict scoping means that you cannot call bar() from anywhere in a form. You also get a run-time error indicating that the function bar() could not be resolved. The script engine looked for bar() within the scope that you have access to and did not find it.

**Scoping and target version**

If you use strict scoping, remember that you get performance improvements in forms targeted for Acrobat or Adobe Reader 8.1 and later. Avoid using strict scoping in forms targeted for older versions of Acrobat or Adobe Reader. Otherwise, the scripts in the forms can work differently. For existing forms, back up the form before you enable strict scoping and always verify the script afterwards. When you enable strict scoping and then change the target version to earlier than Acrobat or Adobe Reader 8.1, warning messages appear.

**When to use scoping**

When a form is targeted for Acrobat or Adobe Reader 8.1 and strict scoping is on, declared JavaScript variables are released after each script executes. When a form is targeted for Acrobat or Adobe Reader 9.0 and later, strict scoping does not release all the JavaScript variables. The excepting is when you remerge or import new data.

The performance enhancements with strict scoping rules apply to forms targeted for Acrobat or Adobe Reader 8.1 and later. Do not apply strict scoping rules to forms that are targeted for versions of Acrobat or Adobe Reader earlier than 8. Otherwise, the scripts can behave differently or not work.
To enable strict scoping
1. Select File > Form Properties and click the Defaults tab.
2. Select Enforce Strict Scoping Rules In JavaScript and then click OK.

To attach a JavaScript script to an object
You can add a JavaScript script to any form design object that allows calculations and scripts, including the script object.
1. Make sure that you have the multiline version of the Script Editor displayed in the Designer ES2 workspace.
2. Select a field on your form. For example, add a new text field to your form design.
3. In the Show list, select a valid event. For example, using the new text field, select the `docReady` event.
4. In the Run At list, select where you want the script to execute. For example, for the new text field, select Client.
5. Click the Functions icon or F10 to display a list of JavaScript functions.
6. Select the desired function, and press Enter.
7. Replace the default function syntax notation with your own set of values. Alternatively, you can create your own script manually in the Script Source field of the Script Editor. For example, in the new text field, add the following JavaScript to the Script Source field:
   ```javascript
   this.border.fill.color.value = "255,0,0";
   ```
8. Click the Preview PDF tab to test the form. The text appears red for the new button object when the form is displayed in the Preview PDF tab.
Secure Form
A very important consideration for designing interactive PDF forms is ensuring that your forms and the data you gather is secure. The following functions and features in Designer ES2 provide security options for your forms:

**Conditional Text - PDF Security settings: Workbench & Standalone**

**PDF Security settings**  Let you define the access capabilities users have to the PDF forms. In the Form Properties dialog box, you can specify a password that the users must enter to open and fill the form. You can also restrict various levels of access to the form. For example, you may want to allow users to fill fields and signatures but not add comments or pages to the form.

**Digital signatures**  Let you authenticate a signing party and ensure that a form is not altered after it is signed. To add a digital signature field to a form design, you can use a signature field or a data signature associated with a submit button. Use a signature field to protect the appearance of the form objects and the values they hold. Use a data signature to protect the integrity of the form data.

**Password fields**  Let you control access to a form. You can set character masking as an option to the field so that the data is not displayed on the screen. You can define whether the password is optional, recommended, or required, and you can set up messages to prompt users appropriately. All user input may be validated through scripting. You can also bind password fields to data sources.

**Secure WSDL data connections**  Let you use HTTP/HTTPS and message-level (SOAP) client authentication to access WSDL documents and to connect to secure web services by using a security token (embedded in the SOAP message header).

**Importing digital IDs**

When you create a secure WSDL data connection with HTTP/HTTPS client certificate authentication, you need to use a digital ID to access the WSDL document on the secure web server. The digital ID is the credential that the secure web server requires to carry out the client authentication.

A digital ID is a document that usually contains the owner's name and public key, the expiration date of the public key, the serial number of the ID, and the name and digital signature of the organization that issued the ID. The digital ID binds together the owner's name and a pair of electronic keys (a public key and a private key) that can be used to encrypt and sign documents.

Before you can create a secure WSDL data connection with HTTP/HTTPS client certificate authentication, you must have a digital ID available in the Windows Certificate Store or in the Designer ES2 Digital ID Files store, ready to select when you create the connection. Client certificate authentication uses a digital ID for HTTP/HTTPS authentication, where you are prompted with the Select Digital ID dialog box. This dialog box contains a list of the digital IDs that are available in the Windows Certificate Store or in the Designer ES2 Digital ID Files store. You must select the correct digital ID to log in to the secure server. The digital ID is embedded in HTTP/HTTPS protocol header, and acts as a credential for the HTTP/HTTPS client authentication.

**Note:** Client certificate authentication may fail on Windows Vista. By default, Microsoft® Windows® Vista™ enables increased HTTPS Security in Internet Explorer 7 through certificate revocation checking. Windows Vista checks for certificate revocation to determine whether the presented certificate remains valid. The increased security may cause the HTTPS client certificate authentication certificate for LiveCycle Designer ES to fail. To disable the certificate revocation checking, go to the Advanced tab of the Internet Control Panel and, in the Security section, deselect the Check For Server Certificate Revocation option.

When the HTTPS connection is first established, the HTTPS server sends its certificate to the client for verification. At this point, you must have the certificate that is needed to identify the HTTPS server's certificate installed on the client computer. If the HTTPS server certificate cannot be identified (trusted), Designer ES2 displays a warning dialog box that lets you indicate whether you want to continue and establish the connection.

You can use the two tabs in the Digital IDs dialog box (Tools > Options > Security > Digital IDs) to import digital IDs into the Windows Certificate Store or Digital ID Files store or both. The digital IDs in the Windows Certificate Store are available to all Windows applications. The digital IDs in the Digital ID Files store are available in Designer ES2 only. The Digital ID Files tab lists the digital IDs that are installed in the current user's private certificate store.
You can also add digital IDs to the Designer ES2 Digital ID Files store by using the Select Digital ID dialog box when you create a secure WSDL data connection with client certificate authentication.

Digital IDs are stored in a password-protected PKCS#12 file with the file name extension *.p12 or *.pfx.

**To import a digital ID into the Windows Certificate Store**

2. Click Digital IDs.
3. Click the Windows Certificate Store tab and click Import Digital ID.
4. Click Next.
5. In the File Name box, type the path to the digital ID file to import, or click Browse to locate and select the digital ID file, and then click Next. Be sure to select the Personal Information Exchange (*.pfx* .p12) option in the File Of Type list.
6. In the Password box, type the password that is required to access the digital ID file.
7. (Optional) Select Enable Strong Private Key Protection.
8. (Optional) Select Mark This Key As Exportable.
9. Click Next.
10. Select Automatically Select The Certificate Store Based On The Type Of Certificate.
11. Click Next, and then click Finish. The name of the digital ID appears on the list of IDs that is available in the Windows Certificate Store.

**To import a digital ID into the Digital ID Files store**

2. Click Digital IDs.
3. Click the Digital ID Files tab and click Add ID File.
4. In the File Name box, type the path to the digital ID file, or click Browse to locate and select the digital ID file.
5. In the Password box, type the password that is required to access the digital ID file, and then click OK. The name of the digital ID appears on the list of IDs available in the Digital ID Files store.
6. Click OK.

**Using digital signatures**

You can protect the integrity of forms by allowing people to use certificates to digitally sign forms. After the form is signed, the signed portion cannot be altered without invalidating the signature. The digital signature invokes a third-party signature handler that provides the required digital signature functionality.

Verifying the signature guarantees that no one tampered with the data after it was submitted. When someone signs a form, a message digest of the data to be signed is created, and a mathematical computation combines the person's private key with the specified form data and encrypts them together. The output is a digital signature. This digital signature contains the signed data and the certificate information associated with the person who signed the form.

When the signature is verified, the individual’s public key is used to decrypt the signed data and to obtain the digest value. The new digest value is calculated against the received document by using the same algorithm of the signing process. If the two digest values do not match, this means that the data has been tampered with since the form was signed, and the verification fails.

Digital signatures also bind certificates to the signed data. The certificate included in the signature can be authenticated to validate the identity of the person who signed the data.

The individual must have a digital certificate from an appropriate certificate issuer to sign the form. Adobe signatures support the Public Key Cryptography Standard (PKCS) #7, using the RSA MD5, RSA SHA-1 or DSA SHA-1 hash algorithm.
Designer ES2 contains the following two types of digital signatures.

**Document signatures**
Document signatures protect the appearance of form objects and the values they hold. To create a document signature, add a signature field to the form design. You can specify whether a document signature applies to an entire form or to a collection of objects on a form. By default, it applies to the entire form. If you want the document signature to apply to a collection of form objects, the signing party must use Acrobat or Adobe Reader 8.0 or later.

If the document signature applies to a collection of objects, it ignores static objects, such as circles and rectangles. It applies only to buttons, check boxes, date/time fields, decimal fields, drop-down lists, image fields, list boxes, numeric fields, paper forms barcodes, password fields, radio buttons, other signature fields, and text fields.

To sign the form, the user clicks the signature field. If the document signature applies to a collection of form objects, the objects specified in the signature are locked and set to read-only. Locking the objects prevents form recipients from changing the object values after the document is signed.

**Data signatures**
Data signatures secure the form data and guarantee the data integrity during transmission. To create a data signature, add a submit button, email submit button, or HTTP submit button, and select the Sign Submission option in the Object palette. To sign the data, the signing party must use Acrobat or Adobe Reader 8.0 or later.

Data signatures can apply to the form data only or to the entire submission, including any attachments such as annotations, document signatures, or PDF documents.

To sign the form data, the user clicks the submit button. When a user clicks the button, they must provide their signing certificate so that their digital signature is applied to the form data before it is submitted.

**To add a signature field**
Designer ES2 provides a built-in signature field that you can add to a form so that users can sign the data they enter. By default, the signature applies to all the objects on the form. If you want the signature to apply to a collection of objects on the form, you need to create the collection and then specify the collection in the Signature tab of the Object palette.

If the signature applies to a collection of objects, signing the document also locks the fields in the collection. If you add more than one signature field to cover different parts of the form, the first collection of objects is locked after the first signature field is signed and only the remaining fields can be filled in the form.

When a signature applies to a collection of objects, it protects the value in the objects. The signature can also protect the appearance of the objects in the collection so that their original appearance is restored when the form is opened. The ability to protect the appearance of the objects depends on the Preserve Scripting Changes To Form When Saved option in the Defaults tab of the Form Properties dialog box. If this option is set to the default, Automatically, the appearance of the objects is automatically protected. However, if the option is set to Manually, you will need to write a script to restore the appearance of the objects.

A signature field should not be repeatable and should not be inserted in a repeatable subform object. A form with repeatable signature fields is not valid and causes problems in Acrobat where the signature field may not appear at all.

If signature covers a collection of objects, the objects in the collection should not be repeatable, because the number of objects can change and invalidate the signature. You can either remove the repeatable objects from the collection or use a data signature instead.

To sign the form, the user clicks the signature field. To sign a collection of form objects, the signing party must use Acrobat or Adobe Reader 8.0 or later.

1. In the Object Library palette, click the Standard category and drag the Signature Field onto the form design.
2 (Optional) To apply the signature to a collection of fields on the form, click the Signature tab in the Object palette and select Lock Fields After Signing.

3 (Optional) Create and select the collection:
   • Select New/Manage Collection.
   • In the Collection List dialog box, click New, type a collection name, press Enter, and click Modify.
   • In the Collection Editor dialog box, select the objects that you want to include in the collection and click OK.
   • In the Collection List dialog box, click Close.
   • Select the collection from the list.

4 (Optional) Apply the signature to the fields inside or outside the collection:
   • To apply the signature to the fields inside the collection, select All Fields In Collection.
   • To apply the signature to the fields outside the collection, select All Fields Not In Collection.

5 (Optional) Click Settings to define optional security settings, such as a signature handler or certificate revocation list.

6 Use the palettes to format the appearance of the signature field.

7 Save the form design.

**Considerations for adding multiple signature field**

When you place a signature field on a form design, the Lock All Fields After Signing option is selected by default. This means all that all form objects will be locked after the field is signed.

If you place more than one signature on the form design, and the Lock All Fields After Signing option is selected for at least one of the signature fields, all form objects, including all signature fields, will be locked after the form is signed.

To ensure that all signature fields can be signed, create a collection of form objects that includes only one of the signature field objects, and make sure that the Lock All Fields After Signing option is selected. After this signature field is signed, all of the objects specified in the collection will be locked. Keep any remaining signature field objects outside of the collection, and deselect the Lock All Fields After Signing option for each one, so that these signature fields can still be signed.

**To test a signatures field**

Use the Preview PDF tab to verify that the signature field behaves as expected. Before testing the form design, ensure that, in the Form Properties dialog box, you have set the default options for previewing an interactive form.

To sign a collection of form objects, the signing party must use Acrobat or Adobe Reader 8.0 or later.

8 Click the Preview PDF tab to view the form design as an interactive PDF form.

9 Click in the signature field.

10 In the Alert dialog box, select Continue Signing.

11 In the Data Exchange File dialog box, select a digital signature or click Add Digital ID to create a new ID.

**To add a data signatures to a submit button**

You can configure a submit button to create a data signature before submitting data so that users can sign the form data when they submit the form. Data signatures provide data integrity and authenticate the signing party during transmission. For example, you can use data signatures to secure the form data during automated transactions.

To sign the data, the signing party must use Acrobat or Adobe Reader 8.0 or later.
12 In the Object Library palette, click the Standard category and drag the Button, Email Submit Button, or HTTP Submit Button onto the form design.

13 In the Object palette, associate a data signature with the button:
   • If you are using a standard button, in the Control Type area of the Field tab, select Submit, and then select Sign Submission in the Submit tab.
   • If you are using an email or HTTP submit button, select Sign Submission in the Field tab.

14 (Optional) Click Settings to define optional security settings, such as whether the signature applies to the data only or to the entire submission.

15 Use the palettes to format the appearance of the submit button.

16 Save the form design.

To use digital signatures in dynamic PDF forms

If you add a digital signature to a dynamic PDF form, you need to preserve the state of the form when it is signed. Preserving the form state ensures that scripting changes in a form are preserved when a form is saved. For example, if some fields are hidden when a form is signed, those fields should remain hidden when the form is saved and reopened.

The form state can be preserved automatically or manually. If it is automatically preserved, scripting changes are automatically preserved when the form is saved. If it is manually preserved, scripting changes are preserved through scripting when the form is saved. Use the delta script object and the preSave and initialize events to manually preserve the form state.

If the form is not certified, you can preserve the form state automatically or manually. If the form is certified, you must manually preserve the form state. Automatically preserving scripting changes and restoring the form state invalidates the digital signature in a certified form.

Note: If the form is certified, you can use data binding instead of scripts to determine the form state.

You specify how the form state is preserved on the Form Properties dialog box.

17 To display the Form Properties dialog box, select File > Form Properties and click Defaults.

18 To display the Form Properties dialog box, select Edit > Form Properties and click Defaults.

19 Select a Preserve Scripting Changes To Form When Saved option:
   • If the form is not certified, select Automatically or Manually.
   • If the form is certified, select Manually.

Using password fields

You can control access to a form using a password field and a script. The password field provides a number of options:

• A character-masking option that enables you to specify a character that masks the data the user enters. The password itself does not display in the password field
• An edit pattern, which describes the syntax for entering data into the password field at run time
• A validation pattern, which is used to validate user input at run time
• A setting that specifies the password field as required, recommended, or optional, and includes a message for prompting users to enter a recommended or required value
Note: The capture of information and the display of characters in the field is for visual purposes only. No attempt is made to hide the value of the field from the programming interfaces or when submitting data. If you want to secure your information, you must use a secure link (such as SSL) to encrypt the data as it is transmitted.

To add a password field in an interactive form
1 In the Object Library palette, click the Standard category and drag a Password Field object onto the form design.
2 Using the Layout, Font, Paragraph, and Border palettes, set the properties you want for the password field.
3 Add a validation pattern that will be used to validate the password and the message that will appear if the password is not a valid format.
4 (Optional) Change the default masking character.
5 Add any binding information as required.
6 Save and test the form design.

To change the password display character
The password display character that appears by default when a user enters a password is the asterisk symbol (*). You can change this symbol in the Password Display Character option of the Object palette Field tab.

To specify an input format and validation message
You can include a validation pattern that is used to validate user input at run time and a message to prompt the user to re-enter the password if the format is not a valid pattern. When a user enters a new password, it is compared it to the validation pattern. If the pattern does not match, a message appears.

You can dynamically populate a validation pattern message with a value from a data source. This option allows you to ensure that users enter the correct value in the field.
1 In the Object palette, click the Value tab, click Validation Pattern, and do the following actions:
   • In the Select Type list, select a validation pattern or, in the Pattern box, type a custom pattern.
   • In the Validation Pattern Message box, type a message that will prompt users to enter the correct value.
2 To generate an error at run time, select Error.

Using PDF security options
When distributing interactive PDF forms, you may want to restrict some of the capabilities that are normally available to PDF forms. For example, in a PDF form, you can normally add comments, insert and delete pages, sign the form, copy text, and save copies of the form.

You can select the Form Properties dialog box from the File menu and set the PDF security options in the PDF Security tab to restrict access to certain capabilities. You can disable the following options:

• Printing of forms
• Ability to open a form without a password
• Insertion, extraction, or rotation of pages
• Ability to add comments
• Ability to add signatures
• Copying of text and images
• Plain text metadata
When you restrict access to any of the PDF form features, you should be aware that it may also affect the ability of screen readers to read the tagged PDF elements in your file. A *screen reader* is assistive software technology that allows users with vision impairment or disabilities to interact with computer software. To ensure that the form is available to these users, you need to enable text access for screen reader devices.

**To set a password to open a PDF form**

You can set a password that a user must enter to open a PDF form. This is a two-step process; first, you specify that a password is required and then you specify the password that the user must enter when opening the form.

1. Select File > Form Properties.
2. Click the PDF Security tab.
3. Select Require A Password To Open The Document, and click OK.
4. Save the form as an Adobe PDF Form (*.pdf).
5. In the Set Document Open Password dialog box, do these tasks:
   - In the Document Open Password box, type a password.
   - In the Confirm box, retype the password.
   - Click OK.
6. Close the form design.

**To restrict printing in a PDF form**

You can prevent users from print a PDF form. You can specify whether the form can be printed, printed with minimal resolution, or printed with a high resolution output.

1. Select File > Form Properties.
2. Click the PDF Security tab.
3. In the Permissions area, select Use A Password To Restrict Printing And Editing Of The Document And Its Security Settings.
4. In the Printing Allowed list, select one of the following options:
   - None
   - Low Resolution (150 dpi)
   - High Resolution

**To restrict editing in a PDF form**

You can disable certain editing capabilities for PDF forms:

- Inserting, extracting, and rotating pages
- Adding comments
- Adding signatures
- Copying of text and images
- Enabling plain text metadata

1. Select File > Form Properties.
2. Click the PDF Security tab.
3. In the Permissions area, select Use A Password To Restrict Printing And Editing Of The Document And Its Security Settings.
4. In the Changes Allowed list, select one of the following options:
   - Inserting, deleting, and rotating pages
   - Filling in form fields and signing
   - Commenting, filling in form fields, and signing
Select one or both of the following options:

- Enable copying of text, images, and other content
- Enable plaintext metadata

**To enable text access for screen readers in secure forms**

If you choose to set options that restrict access to PDF functions, you should be aware that these options can also affect the ability of screen readers to read the tagged PDF material in your form.

To ensure that your forms are still accessible to screen reader devices, you must ensure that the Text Access For Screen Readers option is selected.

1. Select File > Form Properties.
2. Click the PDF Security tab and, in the Permissions area, do these tasks:
   - Select Use A Password To Restrict Printing And Editing Of The Document And Its Security Settings.
   - Select Enable Text Access For Screen Reader Devices For The Visually Impaired.
Validating Data

Depending on the requirements of your situation, you can specify one or more of the following patterns to control how field values, such as text fields, numeric fields, and date/time fields are formatted at run time:

- A display pattern, which describes how data will be displayed in the form. If you define an initial default value, it is formatted according to the display pattern. The display pattern is also responsible for formatting user input and any bound values retrieved at run time.
- An edit pattern, which describes the syntax for entering data into a date/time field, numeric field, text field, or password field at run time.
- A validation pattern, which is used to validate user input at run time.
- A data pattern, which describes the syntax of bound or saved data.

The formatting options that you choose will depend on the purpose of your form. For example, if you are designing an interactive form, for each field you should define an edit pattern to process user input and a validation pattern to validate the input. You would only define a data pattern if the fields are bound to a data source.

When to use patterns

Use patterns to control how field values are processed at run time. For example, users can enter letters and numbers into a text field and any special punctuation or spacing can be applied automatically according to a predefined pattern before the value is displayed.

Capturing and displaying user input

If you are creating a form to capture data, you can specify how data should be formatted. You specify how the data should appear using a display pattern. If you do not specify a display pattern, the data appears according to Designer ES2 defaults.

If users will be entering data that does not match the Designer ES2 defaults, you must specify an edit pattern. The edit pattern describes the syntax of the user input. Given the pattern, the run-time application converts the user input into a raw value and then formats the value according to the display pattern.

If you are designing an interactive form, consider what user input must be validated. For example, a text field may or may not require validation depending on usage. A multiple line text field allowing the form filler to enter a comment does not need to be validated. Similarly, a numeric field will automatically prevent the form-filler from entering any non-numeric data. However, if the data has to be restricted to a specific range of numbers, you will want to validate the user input. You can choose to display a custom message to prompt users for a correct value at run time. If you do not specify a custom message, the system generates one automatically.

Remember that by using the options on the Form Validations tab in the Form Properties dialog box, you can configure how Acrobat displays validations messages, highlights failed or mandatory fields that contain invalid data or no data, and sets the focus on the first field that fails to validate.

Note: User input can be processed through FormCalc formulas and JavaScript scripts (for example, a script can request the raw value of a field). Because formulas and scripts operate on raw and formatted values, it is important to validate those fields where input is restricted. Because formulas and scripts operate on raw values, it is important to validate those fields where input is restricted.

One example of how an edit and validation pattern may be used together is a credit card or social security number entry. You could define a text field with the following edit patterns:

text[9999-9999-9999-9999]|text[9999 9999 9999 9999] for credit cards

or

text[999-99-9999]|text[999 99 9999] for a US social security number

In both cases, the user may enter the number with hyphen(-), space ( ), or just the 16 or 9 digit number. The canonical, or simplest form of the number is the 16 or 9 digit number.

You may also choose to add the following validation pattern:

text[9999999999999999]
or text{9999999999}

In this case, only the number is stored and the validation checks for the correct number of digits. However, in this case, it might be more useful to specify a validation script rather than a pattern. There are algorithms that will checksum a credit card number to ensure that it looks like a valid credit card number and not just a random 16 digit number. An example is the Luhn Algorithm for credit cards.

The result is a form that has a text field where the edit pattern allows user entry in one of three typical ways for typing a credit number, and the validation runs a script that validates that the number looks like a valid credit card number.

**Retrieving and displaying bound data**

If bound data will be merged with a form, you can specify how the data should be formatted for display using a display pattern. If you do not specify a display pattern, the data is displayed according to Designer ES2 defaults.

If the bound data does not match Designer ES2 defaults, you must specify a data pattern. The data pattern describes the syntax of the bound data. Given the pattern, the run-time application converts the retrieved data into raw values and then formats them for display.

**Defaults for value formatting**

Default values must conform to the following rules, depending on the type of field.

<table>
<thead>
<tr>
<th>Field</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date/Time Field</td>
<td>A default date/time value must conform to the short format for the locale specified for the date/time field. However, by default, Designer ES2 displays the default value in the medium locale format at both design time and run time. For example, consider a form with a Date/Time Field set to use the German (Germany) locale. You enter the default value for a date in the short format DD.MM.YY. After you change the focus to another field, the value specified in the field on the page is displayed in the medium format DD.MM.YYYY. The formatted value also appears in the medium format if you view the form in the Preview PDF tab. <strong>Note:</strong> At run time, by default, form fillers must edit the value of date/time fields using the short format for the locale specified for the field. If you specify an Edit Pattern on the Edit tab in the Patterns dialog box (Field tab &gt; Patterns), that pattern overrides the short format, and users must enter data that conforms to the Edit Pattern.</td>
</tr>
<tr>
<td>Numeric Field or Decimal Field</td>
<td>A default numeric value can be any integer or any decimal number that contains a single radix point. The radix character can be either a “.” (period) or “,” (comma) depending on the locale selected. Thousands separators (or grouping symbols) and currency symbols are not valid as part of the default value. For example, if a numeric field is set to the locale English (USA), and you specify the default value $1,234.56, both the currency symbol “$” (dollar sign) and the thousands separator “,” (comma) are not valid.</td>
</tr>
<tr>
<td>Text Field</td>
<td>A default text value (including passwords) can be any alphanumeric text string, including spaces. <strong>Note:</strong> Only those fields listed in the table have default values that must conform to locale-specific formatting.</td>
</tr>
</tbody>
</table>

**To specify a default value**

Date/time fields, numeric fields, and text fields can display an initial (default) value when the form is opened. The value can be derived from a run-time property, or you can specify the value explicitly in Designer ES2. The value can also be derived from an external data source through binding. At run time, Designer ES2 formats field default values according to the locale specified for each field.

1. Select a date/time field, decimal field, numeric field, or text field.
2. In the Object palette, click the Field tab. Select a locale from the Locale list.
3. In the Object palette, click the Value tab. Type the value into the Default box.
The default value must be specified in locale-sensitive format.

Note: If the data is bound and a data pattern has been specified, the value must match the data pattern specified in the Binding tab.

To specify a display pattern
At run time, Designer ES2 displays date, time, and numeric field values in locale-sensitive format. If you want to display a field value in a format other than the default, you can specify the custom pattern by clicking the Patterns button on the Field tab.

Note: Drop-down lists support custom user entries, but a display pattern for custom user entries cannot be specified. You can write a script to format the user input if required.

Because the display pattern describes how data will be displayed in the form, all default values, user-entered values, and values retrieved from a database are converted to the format described by the display pattern.

Note: Dates earlier than January 1, 1900 are not formatted by the display pattern.

1. Select the date/time field, numeric field, or text field.
2. In the Object palette, click the Field tab.
3. Click Patterns and either select one of the predefined display patterns from the Select Type list or type a custom pattern in the Pattern box.

To prompt users to enter data
Prompts are useful for situations where users are expected to enter data or make a selection. You can write a message to prompt users to enter a value into a date/time field, numeric field, text field, password field, or drop-down list, or prompt users to select an option from a drop-down list, list box, or radio button group.

Recommending that users enter data
You can recommend that users enter data in a field but still let them submit the form if they do not. If a user enters data in the field, leaves the field and then clears it, a message box appears. A custom message appears if one written in the Empty Message box. A standard empty field message appears if you do not type a custom message. A message only appears if there was data in the field, the value was deleted, and the user exited the field without re-entering data. If the user never attempts to enter data in the field and tries to submit the form, a field is required message appears. The user can choose to ignore the message and submit the form. Choose User Entered - Recommended to recommend that users enter data in a field.

Requiring that users enter data
You can make it mandatory for users to enter data in a field before they can submit a form. If a user enters data in the field, tabs out, and then returns to clear it, a message box appears. A custom message appears if one written in the Empty Message box. A standard empty field message appears if you do not type a custom message. A message only appears if there was data in the field, the value was deleted, and the user exited the field without re-entering data. If the user never attempts to enter data in the field and tries to submit the form, a field is required message appears. Choose User Entered - Required to make it mandatory that users enter data in a field.

Remember that by using the options on the Form Validations tab in the Form Properties dialog box, you can configure how Acrobat displays validations messages, highlights failed or mandatory fields that contain invalid data or no data, and sets the focus on the first field that fails to validate.

Note: If users do not enter a value into the field and try to submit the form, the error message field is required appears. However, users can save and close a PDF form without providing recommended or required values. In this case, no messages appear to prompt users for input.

1. Select the field, drop-down list, list box, or radio button group.
2. In the Object palette, click the Value tab. From the Type list, select one of these options:
To specify an edit pattern

At run time, Designer ES2 displays date, time, and numeric field values in locale-sensitive format. If you want to permit form fillers to edit field values in a format other than the locale-sensitive default, you can specify the Edit Pattern on the Field tab. If the user's input does not conform to the edit pattern, the data is input as-is.

The edit pattern can be different than the display pattern. For example, because it is easier for users to enter short dates and read long dates, you could consider specifying a short date for a date/time field’s edit pattern and a long date for its display pattern. When the display and edit patterns are different, the value is formatted to match the display pattern as soon as the user exits the field.

Note: This option is not available when the Type option in the Value tab of the Object palette is set to Protected, Calculated - Read Only or Read Only.

1. Select the date/time field, numeric field, text field, or password field.
2. In the Object palette, click the Field tab.
3. Click Patterns, click the Edit tab, and either select one of the predefined display patterns from the Select Type list or type a custom pattern in the Pattern box.

To validate user input

Three separate validations are possible for any field. The order of initiation of these validations is as follows:

- Test the field for null content.
- Verify the format of the field value against a specific field pattern.
- Invoke a validation script.

You can define a validation pattern to validate user input for date/time fields, numeric fields, text fields, and password fields. By default, null entries are not accepted when a value is required. Raw values are compared to the validation pattern directly and, if the raw value matches the validation pattern, it is formatted for display.

If the user-entered value does not match the validation pattern, a programming error or warning appears. The error/warning is trapped by Acrobat, Adobe Reader, or Forms ES2, which returns a message to the user automatically. If an edit pattern has not been specified and the user input does not match Designer ES2 defaults, validation fails.

A validation message appears if objects that require values contain null values and the user attempts to submit data to Forms ES2.

A validation message displays if objects that require values contain null values and the user attempts to submit data to Adobe document services.

Note: Users can save and close a PDF form without providing required values. In this case, no validation is performed.

If needed, you can write a custom validation pattern message to replace the default error or warning message.

In addition to a validation pattern, or in cases where a validation pattern is not supported (for example, for radio button groups and check boxes), you can validate user input by using a validation script. Validating input through a script ensures that the data is acceptable for your application. A custom message and run-time error or warning is also supported in this case.
Remember that by using the options on the Form Validations tab in the Form Properties dialog box, you can configure how Acrobat displays validations messages, highlights failed or mandatory fields that contain invalid data or no data, and sets the focus on the first field that fails to validate.

You can dynamically populate a validation pattern message with a value from a data source. This option allows you to ensure that users enter a valid value in the field.

To define a validation pattern and custom message
1. Select the date/time field, numeric field, text field, password field, drop-down list, or list box.
2. In the Object palette, click the Value tab.
3. Click Validation Pattern and either select one of the predefined validation patterns from the Select Type list or type a custom pattern in the Pattern box.
4. In the Validation Pattern Message box, type a message to prompt users to enter the correct value. The message should specify the required input format. To start a new line in the message, press Ctrl+Enter.
5. To have a programming error to appear instead of a warning, select the Error option.

To display a message when an attached script detects unacceptable input
1. Select the date/time field, numeric field, text field, password field, drop-down list, list box, check box, or radio button group.
2. In the Object palette, click the Value tab. In the Validation Script Message box, type the message.
3. To have a programming error appear instead of a warning, select the Error option.

To specify a data pattern
Data binding options enable you to build a form that captures data for enterprise infrastructures and/or use an external data source to populate a form at run time. For example, given appropriate binding information and access to the data source, Acrobat and Adobe Reader can import and display data from an OLEDB database when the form is opened. Objects can also be bound to an XML schema, an XML file, or a WSDL data source.

Acrobat, Adobe Reader, and Forms ES2 interpret the data-binding properties to store captured data and parse retrieved data. Acrobat, Adobe Reader, and Adobe document services interpret the data-binding properties to store captured data and parse retrieved data. By default, an object’s data is stored and merged according to Adobe data-merging rules. When a form opens in Acrobat or Adobe Reader, or is rendered by Forms ES2, the field values are populated from the data source. When a form opens in Acrobat or Adobe Reader, or is rendered by Adobe document services, the field values are populated from the data source. Any changes to a field’s value by the user are committed to the associated data source when the form is saved in Acrobat or Adobe Reader or the data is submitted to Forms ES2. Any changes to a field’s value by the user are committed to the associated data source when the form is saved in Acrobat or Adobe Reader or the data is submitted to Adobe document services.

If the data is not bound to a data source (for example, if the form data will be returned by email), the data pattern specifies the format that the data is saved in. If you do not create a data pattern, the data will be saved in canonical format. If a form may be filled by end users in a variety of locales or if the data may be returned to more than one locale, having the data in canonical format helps ensure that it is interpreted the same way by all users.

You can specify data patterns for date/time fields, numeric fields, text fields, and password fields. If the data pattern prevents Acrobat or Adobe Reader, or Forms ES2 from parsing a retrieved value, the value appears in the form unchanged (it is not formatted for display). If the data pattern prevents Acrobat or Adobe Reader, or Adobe document services from parsing a retrieved value, the value appears in the form unchanged (it is not formatted for display).

1. Select the date/time field, numeric field, text field, or password field.
2. In the Object palette, click the Field tab.
3 Click Patterns, click the Data tab, and either select one of the predefined data-binding patterns from the Select Type list or type a custom pattern in the Pattern box.

Simple patterns
Simple patterns can be used to format the values of date/time fields, numeric fields, text fields, and password fields. They each have their own rules governing the valid formation of patterns. There is a limited set of characters that you can use in a pattern, and the syntax of a valid pattern differs among date/time fields, numeric fields, text fields, and password fields.

For information about the valid characters that you can use in a pattern and examples of valid constructs, see one of the sections listed below.

Locales
A locale is a standard term used when developing international standards to identify a particular nation (language, country or region). For the purposes of FormCalc, a locale defines the format of dates, times, numeric, and currency values relevant to a specific nation or region so that users can use the formats they are accustomed to.

Each locale is comprised of a unique string of characters called a locale identifier. The composition of these strings is controlled by the international standards organization (ISO) Internet Engineering Task Force (IETF), a working group of the Internet Society (www.isoc.org).

Locale identifiers consist of a language part, a country or region part, or both. The following table lists valid locales for this release of Designer ES2.

<table>
<thead>
<tr>
<th>Language</th>
<th>Country or Region</th>
<th>ISO Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arabic</td>
<td>Algeria</td>
<td>ar_DZ</td>
</tr>
<tr>
<td>Arabic</td>
<td>Bahrain</td>
<td>ar_BH</td>
</tr>
<tr>
<td>Arabic</td>
<td>Egypt</td>
<td>ar_EG</td>
</tr>
<tr>
<td>Arabic</td>
<td>Iraq</td>
<td>ar_IQ</td>
</tr>
<tr>
<td>Arabic</td>
<td>Jordan</td>
<td>ar_JO</td>
</tr>
<tr>
<td>Arabic</td>
<td>Kuwait</td>
<td>ar_KW</td>
</tr>
<tr>
<td>Arabic</td>
<td>Lebanon</td>
<td>ar_LB</td>
</tr>
<tr>
<td>Arabic</td>
<td>Libya</td>
<td>ar_LY</td>
</tr>
<tr>
<td>Arabic</td>
<td>Morocco</td>
<td>ar_MA</td>
</tr>
<tr>
<td>Arabic</td>
<td>Oman</td>
<td>ar_OM</td>
</tr>
<tr>
<td>Arabic</td>
<td>Qatar</td>
<td>ar_QA</td>
</tr>
<tr>
<td>Arabic</td>
<td>Saudi Arabia</td>
<td>ar_SA</td>
</tr>
<tr>
<td>Arabic</td>
<td>Sudan</td>
<td>ar_SD</td>
</tr>
<tr>
<td>Arabic</td>
<td>Syria</td>
<td>ar_SY</td>
</tr>
<tr>
<td>Arabic</td>
<td>Tunisia</td>
<td>ar_TN</td>
</tr>
<tr>
<td>Arabic</td>
<td>United Arab Emirates</td>
<td>ar_AE</td>
</tr>
<tr>
<td>Armenian</td>
<td>Armenia</td>
<td>hy_AM</td>
</tr>
<tr>
<td>Language</td>
<td>Country or Region</td>
<td>ISO Code</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>Azerbaijani-Cyrilic</td>
<td>Azerbaijan</td>
<td>az_Cyril_AZ</td>
</tr>
<tr>
<td>Azerbaijani-Latin</td>
<td>Azerbaijan</td>
<td>az_Latn_AZ</td>
</tr>
<tr>
<td>Basque</td>
<td>Spain</td>
<td>eu_ES</td>
</tr>
<tr>
<td>Bosnain</td>
<td>Bosnia and Herzegovina</td>
<td>bs_BA</td>
</tr>
<tr>
<td>Bulgarian</td>
<td>Bulgaria</td>
<td>bg_BG</td>
</tr>
<tr>
<td>Catalan</td>
<td>Spain</td>
<td>ca_ES</td>
</tr>
<tr>
<td>Chinese</td>
<td>People's Republic of China (Simplified)</td>
<td>zh_CN</td>
</tr>
<tr>
<td>Chinese</td>
<td>Hong Kong S.A.R., China</td>
<td>zh_HK</td>
</tr>
<tr>
<td>Chinese</td>
<td>Taiwan (Traditional)</td>
<td>zh_TW</td>
</tr>
<tr>
<td>Croatian</td>
<td>Croatia</td>
<td>hr_HR</td>
</tr>
<tr>
<td>Czech</td>
<td>Czech Republic</td>
<td>cs_CZ</td>
</tr>
<tr>
<td>Danish</td>
<td>Denmark</td>
<td>da_DK</td>
</tr>
<tr>
<td>Dutch</td>
<td>Belgium</td>
<td>nl_BE</td>
</tr>
<tr>
<td>Dutch</td>
<td>Netherlands</td>
<td>nl_NL</td>
</tr>
<tr>
<td>English</td>
<td>Australia</td>
<td>en_AU</td>
</tr>
<tr>
<td>English</td>
<td>Belgium</td>
<td>en_BE</td>
</tr>
<tr>
<td>English</td>
<td>Canada</td>
<td>en_CA</td>
</tr>
<tr>
<td>English</td>
<td>Hong Kong S.A.R., China</td>
<td>en_HK</td>
</tr>
<tr>
<td>English</td>
<td>India</td>
<td>en_IN</td>
</tr>
<tr>
<td>English</td>
<td>Ireland</td>
<td>en_IE</td>
</tr>
<tr>
<td>English</td>
<td>New Zealand</td>
<td>en_NZ</td>
</tr>
<tr>
<td>English</td>
<td>Philippines</td>
<td>en_PH</td>
</tr>
<tr>
<td>English</td>
<td>Singapore</td>
<td>en_SG</td>
</tr>
<tr>
<td>English</td>
<td>South Africa</td>
<td>en_ZA</td>
</tr>
<tr>
<td>English</td>
<td>United Kingdom</td>
<td>en_GB</td>
</tr>
<tr>
<td>English</td>
<td>United Kingdom Euro</td>
<td>en_GB_EURO</td>
</tr>
<tr>
<td>English</td>
<td>United States of America</td>
<td>en_US</td>
</tr>
<tr>
<td>English</td>
<td>U.S. Virgin Islands</td>
<td>en_VI</td>
</tr>
<tr>
<td>Estonian</td>
<td>Estonia</td>
<td>et_EE</td>
</tr>
<tr>
<td>Finnish</td>
<td>Finland</td>
<td>fi_FI</td>
</tr>
<tr>
<td>French</td>
<td>Belgium</td>
<td>fr_BE</td>
</tr>
<tr>
<td>French</td>
<td>Canada</td>
<td>fr_CA</td>
</tr>
<tr>
<td>French</td>
<td>France</td>
<td>fr_FR</td>
</tr>
<tr>
<td>Language</td>
<td>Country or Region</td>
<td>ISO Code</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>French</td>
<td>Luxembourg</td>
<td>fr_LU</td>
</tr>
<tr>
<td>French</td>
<td>Switzerland</td>
<td>fr_CH</td>
</tr>
<tr>
<td>German</td>
<td>Austria</td>
<td>de_AT</td>
</tr>
<tr>
<td>German</td>
<td>Germany</td>
<td>de_DE</td>
</tr>
<tr>
<td>German</td>
<td>Luxembourg</td>
<td>de_LU</td>
</tr>
<tr>
<td>German</td>
<td>Switzerland</td>
<td>de_CH</td>
</tr>
<tr>
<td>Greek</td>
<td>Greece</td>
<td>el_GR</td>
</tr>
<tr>
<td>Hebrew</td>
<td>Israel</td>
<td>he_IL</td>
</tr>
<tr>
<td>Hungarian</td>
<td>Hungary</td>
<td>hu_HU</td>
</tr>
<tr>
<td>Indonesian</td>
<td>Indonesia</td>
<td>id_ID</td>
</tr>
<tr>
<td>Italian</td>
<td>Italy</td>
<td>it_IT</td>
</tr>
<tr>
<td>Italian</td>
<td>Switzerland</td>
<td>it_CH</td>
</tr>
<tr>
<td>Japanese</td>
<td>Japan</td>
<td>ja_JP</td>
</tr>
<tr>
<td>Kazakh</td>
<td>Kazakhstan</td>
<td>kk_KZ</td>
</tr>
<tr>
<td>Khmer</td>
<td>Cambodia</td>
<td>km_KH</td>
</tr>
<tr>
<td>Korean</td>
<td>Korea</td>
<td>ko_KR</td>
</tr>
<tr>
<td>Korean</td>
<td>Korea Hanja</td>
<td>ko_KR_HANI</td>
</tr>
<tr>
<td>Lao</td>
<td>Laos</td>
<td>lo_LA</td>
</tr>
<tr>
<td>Latvian</td>
<td>Latvia</td>
<td>lv_LV</td>
</tr>
<tr>
<td>Lithuanian</td>
<td>Lithuania</td>
<td>lt_LT</td>
</tr>
<tr>
<td>Malay</td>
<td>Malaysia</td>
<td>ms_MY</td>
</tr>
<tr>
<td>Norwegian - Bokmal</td>
<td>Norway</td>
<td>nb_NO</td>
</tr>
<tr>
<td>Norwegian - Nynorsk</td>
<td>Norway</td>
<td>nn_NO</td>
</tr>
<tr>
<td>Persian</td>
<td>Iran</td>
<td>fa_IR</td>
</tr>
<tr>
<td>Polish</td>
<td>Poland</td>
<td>pl_PL</td>
</tr>
<tr>
<td>Portuguese</td>
<td>Brazil</td>
<td>pt_BR</td>
</tr>
<tr>
<td>Portuguese</td>
<td>Portugal</td>
<td>pt_PT</td>
</tr>
<tr>
<td>Romanian</td>
<td>Romania</td>
<td>ro_RO</td>
</tr>
<tr>
<td>Russian</td>
<td>Russia</td>
<td>ru_RU</td>
</tr>
<tr>
<td>Serbian-Cyrillic</td>
<td>Serbia and Montenegro</td>
<td>sr_Cyrl_CS</td>
</tr>
<tr>
<td>Serbian-Latin</td>
<td>Serbia and Montenegro</td>
<td>sr_Latn_CS</td>
</tr>
<tr>
<td>Slovak</td>
<td>Slovakia</td>
<td>sk_SK</td>
</tr>
<tr>
<td>Slovenian</td>
<td>Slovenia</td>
<td>sl_SI</td>
</tr>
</tbody>
</table>
Usually, both elements of a locale are important. For example, the names of weekdays and months, in English, for Canada and Great Britain are formatted identically, but dates are formatted differently. Therefore, specifying an English language locale is insufficient. Also, specifying only a country as the locale is insufficient. For example, Canada has different date formats for English and French.

In general, every application operates in an environment where a locale is present. This locale is known as the ambient locale. In some circumstances, an application might operate on a system, or within an environment, where a locale is not present. In these rare cases, the ambient locale is set to a default of English United States (en_US). This locale is known as a default locale.
See also

Epoch
Date values and time values have an associated origin or epoch, which is a moment in time from which time begins. Any date value and any time value prior to its epoch is invalid.

The unit of value for all date functions is the number of days since the epoch. The unit of value for all time functions is the number of milliseconds since the epoch.

Designer ES2 defines day one for the epoch for all date functions as Jan 1, 1900, and millisecond one for the epoch for all time functions is midnight, 00:00:00, Greenwich Mean Time (GMT). This definition means that negative time values can be returned to users in time zones east of GMT.

See also

Date formats
A date format is a shorthand specification of how a date appears. It consists of various punctuation marks and symbols that represent the formatting that the date must use. The following table lists examples of date formats.

<table>
<thead>
<tr>
<th>Date format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MM/DD/YY</td>
<td>11/11/78</td>
</tr>
<tr>
<td>DD/MM/YY</td>
<td>25/07/85</td>
</tr>
<tr>
<td>MMMM DD, YYYY</td>
<td>March 10, 1964</td>
</tr>
</tbody>
</table>

The format of dates is governed by an ISO standard. Each country or region specifies its own date formats. The four general categories of date formats are short, medium, long, and full. The following table contains examples of different date formats from different locales for each of the categories.

<table>
<thead>
<tr>
<th>Locale identifier and description</th>
<th>Date format (Category)</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>en_GB</td>
<td>DD/MM/YY (Short)</td>
<td>08/12/92, 08/04/05</td>
</tr>
<tr>
<td>English (United Kingdom)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr_CA</td>
<td>YY-MM-DD (Medium)</td>
<td>92-08-18</td>
</tr>
<tr>
<td>French (Canada)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>de_DE</td>
<td>D. MMMM YYYY (Long)</td>
<td>17. Juni 1989</td>
</tr>
<tr>
<td>German (Germany)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr_FR</td>
<td>EEEE, ‘le’ D MMMM YYYY (Full)</td>
<td>Lundi, le 29 Octobre, 1990</td>
</tr>
<tr>
<td>French (France)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also
Time formats

A time format is a shorthand specification to format a time. It consists of punctuations, literals, and pattern symbols. The following table lists examples of time formats.

<table>
<thead>
<tr>
<th>Time format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>h:MM A</td>
<td>7:15 PM</td>
</tr>
<tr>
<td>HH:MM:SS</td>
<td>21:35:26</td>
</tr>
<tr>
<td>HH:MM:SS 'o’clock' A Z</td>
<td>14:20:10 o’clock PM EDT</td>
</tr>
</tbody>
</table>

Time formats are governed by an ISO standard. Each nation specifies the form of its default, short, medium, long, and full-time formats. The locale identifies the format of times that conform to the standards of that nation.

The following table contains some examples of different date formats from different locales for each of the categories.

<table>
<thead>
<tr>
<th>Locale identifier and description</th>
<th>Time format (Category)</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>en_GB</td>
<td>HH:MM (Short)</td>
<td>14:13</td>
</tr>
<tr>
<td>English (United Kingdom)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr_CA</td>
<td>HH:MM:SS (Medium)</td>
<td>12:15:50</td>
</tr>
<tr>
<td>French (Canada)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>de_DE</td>
<td>HH:MM:SS z (Long)</td>
<td>14:13:13 -0400</td>
</tr>
<tr>
<td>German (Germany)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fr_FR</td>
<td>HH ‘h’ MM Z (Full)</td>
<td>14 h 13 GMT-04:00</td>
</tr>
<tr>
<td>French (France)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also

Date and time patterns

The following symbols must be used to create date and time patterns for date/time fields. Certain date symbols are only used in Chinese, Japanese, and Korean locales. These symbols are also specified below.

**Note:** The comma (,), dash (-), colon (:), slash (/), period (.), and space ( ) are treated as literal values and can be included anywhere in a pattern. To include a phrase in a pattern, delimit the text string with single quotation marks (’). For example, ‘Your payment is due no later than’ MM-DD-YY can be specified as the display pattern.

<table>
<thead>
<tr>
<th>Date symbol</th>
<th>Description</th>
<th>Formatted value for English (USA) locale where the locale-sensitive input value is 1/1/08 (which is January 1, 2008)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>1 or 2 digit (1-31) day of the month</td>
<td>1</td>
</tr>
<tr>
<td>DD</td>
<td>Zero-padded 2 digit (01-31) day of the month</td>
<td>01</td>
</tr>
<tr>
<td>J</td>
<td>1, 2, or 3 digit (1-366) day of the year</td>
<td>1</td>
</tr>
<tr>
<td>JJJJ</td>
<td>Zero-padded, three-digit (001-366) day of the year</td>
<td>001</td>
</tr>
<tr>
<td>M</td>
<td>One- or two-digit (1-12) month of the year</td>
<td>1</td>
</tr>
<tr>
<td>MM</td>
<td>Zero-padded, two-digit (01-12) month of the year</td>
<td>01</td>
</tr>
</tbody>
</table>
Several additional date patterns are available for specifying date patterns in Chinese, Japanese, and Korean locales.

Japanese eras can be represented by several different symbols. The final four era symbols provide alternative symbols to represent Japanese eras.

<table>
<thead>
<tr>
<th>Date symbol</th>
<th>Description</th>
<th>Formatted value for English (USA) locale where the locale-sensitive input value is 1/1/08 (which is January 1, 2008)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMM</td>
<td>Abbreviated month name</td>
<td>Jan</td>
</tr>
<tr>
<td>MMMMM</td>
<td>Full month name</td>
<td>January</td>
</tr>
<tr>
<td>E</td>
<td>One-digit (1-7) day of the week, where (1=Sunday)</td>
<td>3 (because January 1, 2008 is a Tuesday)</td>
</tr>
<tr>
<td>EEE</td>
<td>Abbreviated weekday name</td>
<td>Tue (because January 1, 2008 is a Tuesday)</td>
</tr>
<tr>
<td>EEEE</td>
<td>Full weekday name</td>
<td>Tuesday (because January 1, 2008 is a Tuesday)</td>
</tr>
<tr>
<td>YY</td>
<td>Two-digit year, where numbers less than 30 are considered to fall after the year 2000 and numbers 30 and higher are considered to occur before 2000. For example, 00=2000, 29=2029, 30=1930, and 99=1999</td>
<td>08</td>
</tr>
<tr>
<td>YYYY</td>
<td>Four-digit year</td>
<td>2008</td>
</tr>
<tr>
<td>G</td>
<td>Era name (BC or AD)</td>
<td>AD</td>
</tr>
<tr>
<td>w</td>
<td>One-digit (0-5) week of the month, where week 1 is the earliest set of four contiguous days ending on a Saturday</td>
<td>1</td>
</tr>
<tr>
<td>WW</td>
<td>Two-digit (01-53) ISO-8601 week of the year, where week 1 is the week containing January 4</td>
<td>01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CJK date symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDD</td>
<td>The locale's ideographic numeric valued day of the month</td>
</tr>
<tr>
<td>DDDD</td>
<td>The locale's tens rule ideographic numeric valued day of the month</td>
</tr>
<tr>
<td>YYYY</td>
<td>The locale's ideographic numeric valued year</td>
</tr>
<tr>
<td>YYYYY</td>
<td>The locale's tens rule ideographic numeric valued year</td>
</tr>
<tr>
<td>g</td>
<td>The locale's alternate era name. For the current Japanese era, Heisei, this pattern displays the ASCII letter H (U+48)</td>
</tr>
<tr>
<td>gg</td>
<td>The locale's alternate era name. For the current Japanese era, this pattern displays the ideograph that is represented by the Unicode symbol (U+5E73)</td>
</tr>
<tr>
<td>ggg</td>
<td>The locale's alternate era name. For the current Japanese era, this pattern displays the ideographs that are represented by the Unicode symbols (U+5E73 U+6210)</td>
</tr>
<tr>
<td>g</td>
<td>The locale's alternate era name. For the current Japanese era, this pattern displays the full width letter H (U+FF28)</td>
</tr>
<tr>
<td>gg</td>
<td>The locale's alternate era name. For the current Japanese era, this pattern displays the ideograph that is represented by the Unicode symbol (U+337B)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time symbol</th>
<th>Description</th>
<th>Locale-sensitive input value</th>
<th>Formatted value for English (USA) locale</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>One- or two-digit (1-12) hour of the day (AM/PM)</td>
<td>12:08 AM or 2:08 PM</td>
<td>12 or 2</td>
</tr>
<tr>
<td>hh</td>
<td>Zero-padded 2 digit (01-12) hour of the day (AM/PM)</td>
<td>12:08 AM or 2:08 PM</td>
<td>12 or 02</td>
</tr>
</tbody>
</table>
Reserved symbols
The following symbols have special meanings and cannot be used as literal text.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>When submitted, the symbol matches any one character. When merged for display, it becomes a space.</td>
</tr>
<tr>
<td>*</td>
<td>When submitted, the symbol matches 0 or Unicode white space characters. When merged for display, it becomes a space.</td>
</tr>
<tr>
<td>+</td>
<td>When submitted, the symbol matches one or more Unicode white space characters. When merged for display, it becomes a space.</td>
</tr>
</tbody>
</table>
Complex field patterns
In addition to defining simple patterns for date/time fields, numeric fields, and text fields, you can define a locale-specific pattern or handle variable patterns.

Locale-specific patterns
If you want to force a locale on a pattern, regardless of the locale that has already been assigned to an object, you can define a locale-specific pattern. The syntax of a locale-specific pattern is defined as follows:

\[ \text{category\_name(locale\_name)}\{\text{pattern}\} \]

where
- \text{category\_name} can be date, time, num, or text.
- \text{locale\_name} is identified by a language and/or country or region code, as defined in RFC 1766 (Tags for the Identification of Languages, 1995).
- \text{pattern} is the simple pattern for processing values.

For example, to force a date/time field to translate a date into the French language according to France's country code, you would define the pattern as follows:

\[ \text{date(fr\_FR)}\{DD MMMM, YYYY\} \]

Variable patterns
In cases where the user input or bound data is available in more than one format (for example, telephone numbers may or may not have a three-digit area code), you can define a pattern that accounts for the differences. The syntax for defining a number of acceptable patterns is as follows:

\[ \text{category\_name(pattern)}|\text{category\_name(pattern)}|\text{category\_name(pattern)} \]

where each pattern is separated by a vertical bar (|). You can specify an unlimited number of patterns. For example, the following construct handles two different text patterns:

\[ \text{text(999*9999)}|\text{text(999*999*9999)} \]
Form performance

General design considerations for printing

When a form is sent to print, both the form and the data are loaded into memory and the resulting layout also builds in memory. This means that there is a limit to the size of the final output.

The amount of data and the number of pages in a form are the most significant factors that affect memory usage; however, there are other factors in a form that can also affect memory usage. Keep the following factors in mind when designing forms for print.

- Avoid using choice subforms.
- Use a caption in a text field to label a field instead of static text.
- Avoid using rich text.
- Avoid overuse of subforms and nested subforms.
- Avoid the overuse of rectangle and line objects.

Optimizing performance for images

Images require additional processing time when the form is rendered. If you choose to use images, use compact file formats and the smallest possible image dimensions to minimize the file size. Size the images to the intended dimensions before adding them to the form. Avoid resizing images after adding them to the form.

Ensure that corners are square and lines are exactly horizontal and vertical to optimize performance when the form is rendered. Lines that appear to be exactly vertical or horizontal may have a slight slope.

You can check the slope of horizontal and vertical lines in the Layout palette. The Height option represents the slope of a horizontal line and the Width option represents the slope of a vertical line. In the following example, a vertical line has a small slope of approximately 0.5 millimeters. A width of 0mm makes the line exactly vertical.
Optimizing performance for discrete option choices

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

Optimizing performance for subforms

LiveCycle Designer ES 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 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 use 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 ES creates as many instances of the subform as necessary to represent all the data groups. When rendering the form, Adobe document services create 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 ES does not apply page breaks. Allowing page breaks in subforms causes additional processing, even if Adobe document services do 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 subforms. By default, page breaks are allowed.

In addition, subforms are useful for grouping and organizing objects without binding the subform to a data element. To prevent LiveCycle Forms ES 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. To prevent Adobe document services 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.

Optimizing performance for text fields

By default, entries in text field objects are formatted as plain text. You can change this option to allow rich text formatting.

However, for plain text entries, 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, the extra attributes decrease performance.

Data binding options for performance

For forms that are bound to data sources, 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 ES can process the data binding. However, the more obvious the relationship between the form object and its data element, the more efficiently Adobe document services 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. The binding is ambiguous because multiple instances of the field objects and their data
exist. More processing is required to match the nth occurrence of each object to the nth occurrence of the corresponding data when rendering the form.

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. The binding is evaluated for the first group and does not need to be re-evaluated for subsequent repetitions.

Tip: You can use subforms to group and organize objects without binding the subform to a data element. To prevent LiveCycle Forms ES 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. To prevent Adobe document services 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.

Evaluating data structure for performance

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.

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 exists.

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.

**Performance of objects on master pages versus pages**

Objects on master pages require additional processing time when LiveCycle Forms ES renders the form. If possible, place objects on pages.
Tips for reducing the number of objects

The file size of a form is affected by the number of objects on the form. Consider these general tips to minimize the number of objects on forms:

- When creating a box, 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 text object.
- Combine multiple text objects into a single text object. This strategy is particularly useful after importing forms into LiveCycle Designer ES.

Tips for reducing the complexity of objects

The file size of a form is affected by the complexity of the objects on the form. Simple objects can be processed more efficiently than complex objects. Consider these general tips to minimize the complexity of objects:

- Keep the number of fonts, styles, and sizes to a minimum to take advantage of cached font information.
- Use fixed objects instead of dynamic objects that shrink and grow, if possible.
- Avoid building objects on top of each other, especially more than three layers deep.