

6.5. Creating scripts using JavaScript

Creating scripts in Designer 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 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 reference syntax. Specifically, you must know how to correctly use the XML Form Object Model reference syntax to access objects on your form design.

This table outlines the key concepts for developing scripts in JavaScript for Designer. The table also provides the location for more information on each concept within the *Designer Help*.

Key concept	For more information see...
Creating references to object properties and values, including using the <code>resolveNode</code> method.	Referencing object properties and values <code>resolveNode</code> To use statement completion to create calculations and scripts
Using the host and event models to test and debug your form.	Testing and debugging calculations and scripts Referencing object properties and values
Using a script object to reuse your existing JavaScript functions.	Creating and Reusing JavaScript Functions

In addition to the resources available in the *Designer Help*, the [Developer Center](#) contains extensive scripting resources and documentation.

RELATED LINKS:

Enforcing strict scoping rules in JavaScript

To attach a JavaScript script to an object

Using JavaScript

6.6. 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. Strict scoping is enabled, by default, in Designer, for new forms. For old forms the option to enable strict scoping is available.

6.6.1. 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 ().

```

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 nInner = 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.

```

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.

```

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.

6.6.2. 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*.

```

<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.

6.6.3. 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"/>
```

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:

```
<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 8.1, the function `bar()` does not execute. The reason is because function `bar()` is available only from within the initialized script of `field1`.

6.6.4. 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.

6.6.5. 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.

6.6.6. 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 exception 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.

6.6.7. To enable strict scoping

- 1) Select File > Form Properties and click the Run-time tab.
- 2) Select Enforce Strict Scoping Rules In JavaScript , if the option is available, and then click OK.

NOTE: If the option to enforce strict scoping rules is not available in the Run-time tab, then strict scoping is already enabled.


RELATED LINKS:

Using JavaScript

To attach a JavaScript script to an object

6.7. 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 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.

RELATED LINKS:

Using JavaScript

7. Variables

You can define form variables in Designer 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 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 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.


7.1. 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`.

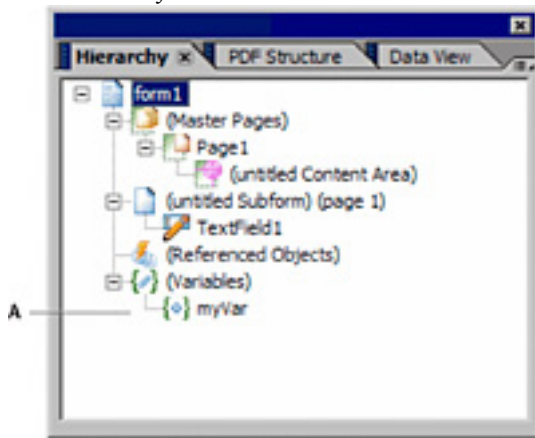
NOTE: Variable names are case-sensitive and should not contain spaces.

7.2. 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.

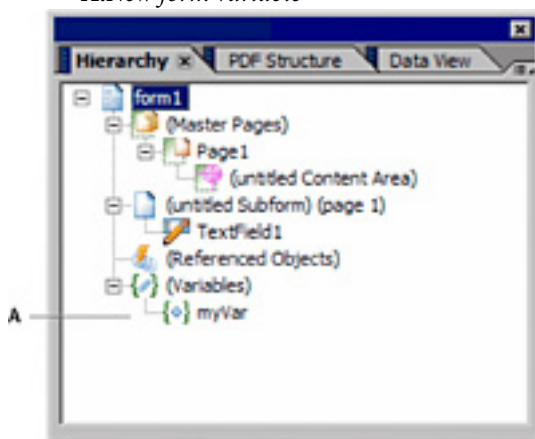
A.New form variable



7.3. To define a text variable

- 1) Select Edit > 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.

A.New form variable




7.4. To view a text variable definition

- 1) Select File > 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.


7.5. 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.

7.6. To delete a text variable

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

7.7. To delete a text variable

- 1) Select Edit > Form Properties.
- 2) In the Variables tab, select the variable and click Delete (Delete) .

7.8. 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.

IMPORTANT: 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 [Scripting Reference](#).

For example, create the following form variable definitions.

Variable name	Value
firstName	Tony
lastName	Blue
age	32

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;
```

NOTE: Using and modifying form variables with scripting in XFA forms can cause the document message bar in Acrobat and Adobe Reader to display a signature validation status warning indicating that the signature validity is unknown due to subsequent changes to the document.

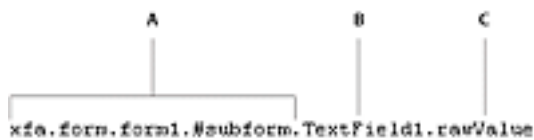
8. Referencing Objects in Calculations and Scripts

Although both FormCalc calculations and JavaScript scripts have rules for structuring code, both rely on the same reference syntax when accessing form object properties and values. The XML Form Object Model provides a structured way to access object properties and values through a compound naming convention with each object, property, and method separated by dot (.) characters.

In general, each reference syntax has a similar structure divided into the following sections:

- The names of the parent objects in the form hierarchy that is used to navigate to a specific field or object. You can use the Hierarchy palette and Data View palette to determine the location of an object relative to other objects in the form and in any associated data.
- The name of the object you want to reference.
- The the name of the property or method you want to access. This section may also include XML Form Object Model objects that precede the property or method in the structure but that do not appear as objects in the Hierarchy palette.

For example, the following illustration shows the reference syntax for accessing the value of a text field on a form design that uses the default object-naming conventions:



A.

Form hierarchy objects

B.

Object name

C.

Property or method name

NOTE: By default, the subform object that represents the first page of a new form is unnamed. In the reference syntax above, the unnamed subform is represented by `#subform`.

The reference syntax notation structure varies slightly, depending on the specific situation. For example, a fully qualified reference syntax works in any situation; however, in some cases, you can use a shortened reference syntax or a reference syntax shortcut to reduce the size of the syntax.

8.1. Referencing object properties and values

The reference syntax you use to access or modify object properties and values takes one of the following forms:

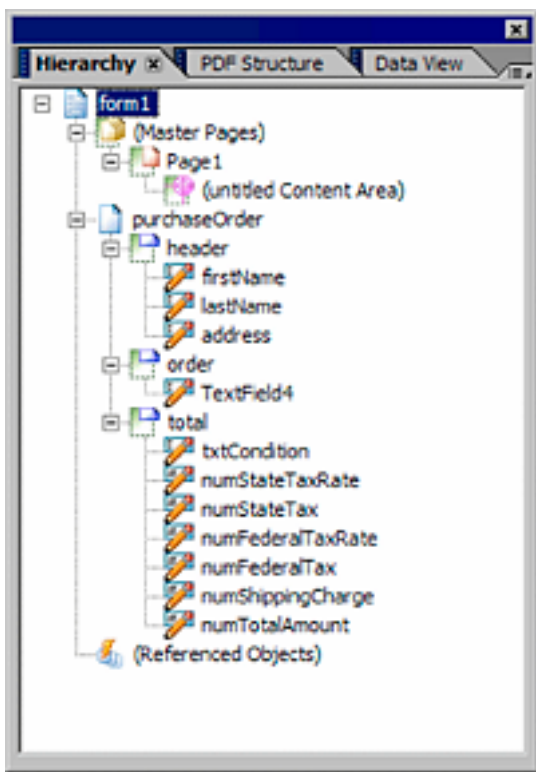
Fully qualified

Reference syntax includes the full object hierarchy, beginning with the `xfa` root node. The fully qualified syntax accurately accesses the property or value of an object regardless of where the calculation or script that contains the reference syntax is located.

Abbreviated

The reference syntax is shortened either because of the relative positioning of the calculation or script that contains the reference syntax and the object syntax references, or because shortcuts are used. In general, although an abbreviated reference syntax is faster to create, the disadvantage is that it works only as long as the objects remain in the same positions relative to each other.

For example, this illustration shows the hierarchy of a sample purchase order form.



This illustration shows a fully qualified reference syntax, for both FormCalc and JavaScript, to access the value of the `txtCondition` field. This reference syntax could be used as part of a calculation or script on any object on the form.

```

A B C D E F G
| | | | | | |
xfa.form.form1.purchaseOrder.total.txtCondition.rawValue

```

- A.
Root Node
- B.
Model
- C.
Form design root node
- D.
Page object
- E.
Subform name
- F.
Object name
- G.
Property or method name

NOTE: Even though the reference syntax is common to both FormCalc and JavaScript, you must observe the conventions for each scripting language. For example, the reference syntax in the example above works as is for FormCalc; however, you would need to include a trailing semicolon (;) character for JavaScript.

If two objects exist in the same container, such as a subform, they are referred to as sharing the same context. Where objects exist in the same context, you can use an abbreviated reference syntax that includes only the name of the object followed by the property or method you want to access. For example, using the example from above, the following abbreviated reference syntax accesses the value of the `txtCondition` field from any of the fields in the `total` subform:

```
txtCondition.rawValue
```

If two objects exist in different containers, they do not share the same context. In this case, you can use an abbreviated reference syntax; however, the syntax must begin with the name of the highest level container object that the two objects do not have in common. For example, using the hierarchy above, the following abbreviated reference syntax accesses the value of the `address` field from the `txtCondition` field:

```
header.address.rawValue
```

Due to the way the XML Form Object Model is structured, some object properties and methods exist on child objects of the objects on the form. These child objects exist only as part

of the XML Form Object Model and do not appear in the Hierarchy and Data View palettes. To access these properties and methods, you must include the child objects in the reference syntax. For example, the following reference syntax sets the tool tip text for the `txtCondition` field:

```
txtCondition.assist.toolTip.value = "Conditions of purchase." // FormCalc  
txtCondition.assist.toolTip.value = "Conditions of purchase."; // JavaScript
```

For more information about the XML Form Object model objects and their structure, see [Scripting Reference](#).

RELATED LINKS:

Referencing Objects in Calculations and Scripts

Referencing unnamed and repeated objects

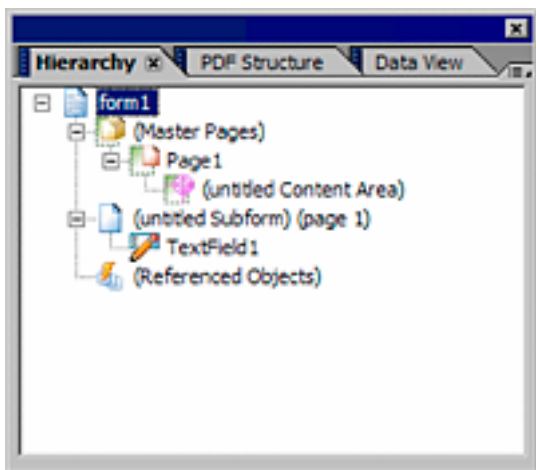
Referencing the current object

FormCalc reference syntax shortcuts

8.2. Referencing unnamed and repeated objects

Designer supports the capability to create both unnamed objects and multiple objects with the same name. You can still create calculations and scripts to access and modify properties and values of unnamed objects by using the number sign (#) notation and object occurrence values using the square bracket ([]) notation. FormCalc correctly interprets the number sign (#) and square bracket ([]) characters; however, JavaScript does not. To access the value of a text field in a situation where the number sign (#) or square brackets ([]) occur, using JavaScript, you must use the `resolveNode` method in conjunction with either a fully qualified reference syntax or an abbreviated reference syntax.

For example, when you create a new blank form, by default, the name of the subform that represents the page of the form is an unnamed subform with an occurrence value of 0. The following illustration shows the form hierarchy on a new form with default object naming.



The untitled subform that represents the first page of the form has an occurrence number of 0. In this situation, both of the following reference syntaxes access the value of the text field in the form hierarchy above on a new form that uses default naming conditions:

```
xfa.form.form1.#subform.TextField1.rawValue  
xfa.form.form1.#subform[0].TextField1.rawValue
```

NOTE: By default, if you do not specify an occurrence value for an object, the reference syntax accesses the first occurrence of that object.

FormCalc recognizes the fully qualified reference syntax above and interprets it directly. To access the same value by using JavaScript, you must use one of these forms of the `resolveNode` scripting method:

```
xfa.resolveNode("xfa.form.form1.#subform.TextField1").rawValue;  
xfa.resolveNode("xfa.form.form1.#subform[0].TextField1").rawValue;
```

If you add a new page to your form, by default, the name of the subform that represents the new page is unnamed; however, the occurrence value for the new subform is set to 1. You can specify the new unnamed subform by using a similar reference syntax as above:

```
xfa.form.form1.#subform[1].TextField1.rawValue // FormCalc  
xfa.resolveNode("xfa.form.form1.#subform[1].TextField1").rawValue; //  
JavaScript
```

NOTE: The statement completion options available in the Script Editor include unnamed objects at the beginning of the list. Objects that have multiple occurrence values appear only once in the list, representing the first occurrence of the object. If you want to access an occurrence value other than the first occurrence, you must manually add the occurrence value to the reference syntax.

You can use the `resolveNode` method to reference objects within other reference syntax statements. This can help to reduce the amount of scripting you need to reference a particular object, property, or method. For example, you could simplify the reference syntax that points to a text field on the second page of your form to the following statement:

```
    xfa.form.form1.resolveNode("#subform[1].TextField1").rawValue  
; // JavaScript
```

For more information on the `resolveNode` method, see [resolveNode](#).

RELATED LINKS:

- Referencing Objects in Calculations and Scripts
- Referencing object properties and values
- Referencing the current object
- FormCalc reference syntax shortcuts

8.3. Referencing the current object

If you want to change properties or values of the current object using calculations or scripts attached to the object itself, both FormCalc and JavaScript use unique shortcuts to reduce the size of the reference syntax. FormCalc uses the number sign (\$) character to denote the current object, and JavaScript uses the keyword `this`.

For example, the following reference syntax returns the value of the current object:

```
$ // FormCalc  
this.rawValue // JavaScript
```

Similarly, you can use the dollar sign (\$) shortcut and the keyword `this` to replace the name of the current object when accessing object properties in calculations and scripts. For example, the following reference syntax changes the tool tip text associated with the current object:

```
$.assist.toolTip.value = "This is some tool tip text." // FormCalc  
this.assist.toolTip.value = "This is some tool tip text."; // JavaScript
```

RELATED LINKS:

[Referencing Objects in Calculations and Scripts](#)

[Referencing object properties and values](#)

[Referencing unnamed and repeated objects](#)

[FormCalc reference syntax shortcuts](#)

8.4. FormCalc reference syntax shortcuts

To make accessing object properties and values easier, FormCalc includes shortcuts to reduce the effort required to create references. This section describes the reference syntax shortcuts for FormCalc.

8.4.1. Current field or object

Refers to the current field or object

Notation

\$

Example

```
$ = "Tony Blue"
```

The above example sets the value of the current field or object to Tony Blue.

8.4.2. Data model root of `xfa.datasets.data`

Represents the root of the data model `xfa.datasets.data`.

Notation

`$data`

Example

`$data.purchaseOrder.total`

is equivalent to

`xfa.datasets.data.purchaseOrder.total`

8.4.3. Form object event

Represents the current form object event.

Notation

`$event`

Example

`$event.name`

is equivalent to

`xfa.event.name`

For more information see [Working with the Event Model](#).

8.4.4. Form model root

Represents the root of the form model `xfa.form`.

Notation

`$form`

Example

`$form.purchaseOrder.tax`
is equivalent to stating
`xfa.form.purchaseOrder.tax`

8.4.5. Host object

Represents the host object.

Notation

`$host`

Example

`$host.messageBox("Hello world")`
is equivalent to
`xfa.host.messageBox("Hello world")`
For more information, see [Working with a Host Application](#).

8.4.6. Layout model root

Represents the root of the layout model `xfa.layout`.

Notation

`$layout`

Example

`$layout.ready`
is equivalent to stating

```
xfa.layout.ready
```

8.4.7. Collection of data record

Represents the current record of a collection of data, such as from an XML file.

Notation

```
$record
```

Example

```
$record.header.txtOrderedByCity
```

references the `txtOrderedByCity` node within the header node of the current XML data.

8.4.8. Template model root

Represents the root of the template model `xfa.template`.

Notation

```
$template
```

Example

```
$template.purchaseOrder.item
```

is equivalent to

```
xfa.template.purchaseOrder.item
```

8.4.9. Data model root of `xfa.datasets`

Represents the root of the data model `xfa.datasets`.

Notation

```
!
```

Example

```
!data  
is equivalent to  
xfa.datasets.data
```

8.4.10. Select all form objects

Selects all form objects within a given container, such as a subform, regardless of name, or selects all objects that have a similar name.

You can use the ‘*’ (asterisk) syntax with JavaScript if it used with the [resolveNode](#) method.

Notation

*

Example

For example, the following expression selects all objects named `item` on a form:

```
xfa.form.form1.item[*]
```

8.4.11. Search for objects that are part of a subcontainer

You can use two dots at any point in your reference syntax to search for objects that are a part of any subcontainer of the current container object, such as a subform.

You can use the ‘..’ (double period) syntax with JavaScript if it used with the [resolveNode](#) method.

Notation

..

Example

The expression `Subform_Page..Subform2` means locate the node `Subform_Page` (as usual) and find a descendant of `Subform_Page` called `Subform2`.



Using the example tree above,

`Subform_Page..TextField2`

is equivalent to

`Subform_Page.Subform1[0].Subform3.TextField2[0]`

because `TextField2[0]` is in the first `Subform1` node that `FormCalc` encounters on its search. As a second example,

`Subform_Page..Subform3[*]`

returns all four `TextField2` objects.

8.4.12. Denote an unnamed object or specify a property

The number sign (#) notation is used to denote one of the following items in a reference syntax:

- An unnamed object
- Specify a property in a reference syntax if a property and an object have the same name

You can use the '#' (number sign) syntax with JavaScript if it used with the [resolveNode](#) method.

Notation

#

Example

The following reference syntax accesses an unnamed subform:

`xfa.form.form1.#subform`

The following reference syntax accesses the `name` property of a subform if the subform also contains a field named `name`:

`xfa.form.form1.#subform.#name`

8.4.13. Occurrence value of an object

The square bracket ([]) notation denotes the occurrence value of an object.

In language-specific forms for Arabic, Hebrew, Thai, and Vietnamese, the reference syntax is always on the right (even for right-to-left languages).

Notation

[]

Example

To construct an occurrence value reference, place square brackets ([]) after an object name, and enclose within the brackets one of the following values:

- [n], where *n* is an absolute occurrence index number beginning at 0. An occurrence number that is out of range does not return a value. For example,

```
xfa.form.form1.#subform.Quantity[3]
```

refers to the fourth occurrence of the Quantity object.

- [+/- n], where *n* indicates an occurrence relative to the occurrence of the object making the reference. Positive values yield higher occurrence numbers, and negative values yield lower occurrence numbers. For example,

```
xfa.form.form1.#subform.Quantity[+2]
```

This reference yields the occurrence of Quantity whose occurrence number is two more than the occurrence number of the container making the reference. For example, if this reference was attached to the Quantity[2] object, the reference would be the same as

```
xfa.template.Quantity[4]
```

If the computed index number is out of range, the reference returns an error.

The most common use of this syntax is for locating the previous or next occurrence of a particular object. For example, every occurrence of the Quantity object (except the first) might use Quantity[-1] to get the value of the previous Quantity object.

- [*] indicates multiple occurrences of an object. The first named object is found, and objects of the same name that are siblings to the first are returned. Note that using this notation returns a collection of objects. For example,

```
xfa.form.form1.#subform.Quantity[*]
```

- This expression refers to all objects with a name of Quantity that are siblings to the first occurrence of Quantity found by the reference.



Using the tree for reference, these expressions return the following objects:

- `Subform_Page.Subform1[*]` returns both `Subform1` objects.
- `Subform_Page.Subform1.Subform3.TextField2[*]` returns two `TextField2` objects. `Subform_Page.Subform1` resolves to the first `Subform1` object on the left, and `TextField2[*]` evaluates relative to the `Subform3` object.
- `Subform_Page.Subform1[*].Textfield1` returns both of the `Textfield1` instances. `Subform_Page.Subform1[*]` resolves to both `Subform1` objects, and `Textfield1` evaluates relative to the `Subform1` objects.
- `Subform_Page.Subform1[*].Subform3.TextField2[1]` returns the second and fourth `TextField2` objects from the left. `Subform_Page.Subform1[*]` resolves to both `Subform1` objects, and `TextField2[1]` evaluates relative to the `Subform3` objects.
- `Subform_Page.Subform1[*].Subform3[*]` returns both instances of the `Subform3` object.
- `Subform_Page.*` returns both `Subform1` objects and the `Subform2` object.
- `Subform_Page.Subform2.*` returns the two instances of the `NumericField2` object.
- You can use the ‘`[]`’ (square bracket) syntax with JavaScript if it used with the [resolveNode](#) method.
-

RELATED LINKS:

[Referencing Objects in Calculations and Scripts](#)

9. 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 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.

9.1. 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.

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.

9.2. 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.

9.2.1. Example

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

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

9.3. 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. For more information about referencing objects in scripting, see [Referencing object properties and values](#).

9.3.1. Example

For example, using the script object example from [To add script to a script object](#), place the following JavaScript script on the `exit` event for a text field. Test the form using the Preview PDF tab.

10. 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 create a script fragment from the Hierarchy palette.

You edit script fragments the same way as other fragments.

10.1. Script fragment properties

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

10.1.1. Source File

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

10.1.2. 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.

10.2. 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**.

10.3. 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.*

10.3.1. 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.

10.3.2. 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

RELATED LINKS:

[Creating and Reusing JavaScript Functions](#)[To create a script object](#)[To add script to a script object](#)

11. Debugging Calculations and Scripts

Designer 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.

11.1. Designer 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 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.

For more information about using the Report palette, see [Using the workspace to debug calculations and scripts](#).

11.2. 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 JavaS-

cript 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.

11.3. FormCalc

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

11.4. JavaScript

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

IMPORTANT: *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.*

For more informatin about using the `messageBox`, see `messageBox`

11.5. 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..

11.6. 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, yu can use the `alert` method from the Acrobat JavaScript Object Model to debug JavaScript.

11.6.1. JavaScript Debugger in Acrobat Professional

The JavaScript Debugger in Acrobat Professional lets you test JavaScripts 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 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 Acrobat Reader DC extensions installed. To enable the JavaScript Debugger in Adobe Reader, you must get the `debugger.js` file and then edit the Microsoft Windows Registry. For more information about enabling the JavaScript Debugger in Adobe Reader, see [Developing Acrobat Applications Using JavaScript](#) (English only).

11.6.2. To enable the JavaScript Debugger for Designer

- 1) Start Designer.
- 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.
- 6) Under JavaScript Debugger, select Enable JavaScript Debugger After Acrobat Is Restarted.
- 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, click the Preview PDF tab.
- 12) Press Ctrl+J to open the JavaScript Debugger.

11.6.3. To prevent the JavaScript Debugger from disappearing in Designer

If the JavaScript Debugger from Acrobat is active and it disappears when you click components in the Designer 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.

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

11.6.4. Evaluating code using the JavaScript Console

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

11.6.5. To evaluate a portion of a line of code

- 1) Highlight the portion in the console window and press either Enter on the numeric keypad or Ctrl+Enter on the regular keyboard.

11.6.6. To evaluate a single line of code

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

11.6.7. To evaluate multiple lines of code

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

11.6.8. To delete content that appear in the JavaScript Console

- 1) 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.

11.6.9. 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).

11.6.10. 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).

RELATED LINKS:

[Using the workspace to debug calculations and scripts](#)

11.7. Debugging tips

Remember the following tips when debugging calculations and scripts.

11.7.1. 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.

11.7.2. Master pages

To debug master pages, drop a different object on each master page to find out which one is specified.

11.7.3. First page in a form

Designer 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.

11.7.4. 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.

11.7.5. 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.

11.7.6. Script error messages

In Designer, 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.

11.7.7. 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.

11.7.8. 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 cannot resolve the script object. The same error can occur when a syntax issue occurs in the script object.

11.7.9. 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.

11.7.10. 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.

11.7.11. 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)`

11.7.12. 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:

```
<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>
```

11.7.13. 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.

12. Working with a Host Application

A host application is the application in which a form exists at any given time.

For example, if you are using Forms to render a form in HTML format, then during the pre-rendering process the host application is Forms.

Once you render a form and view it in a client application such as Acrobat, Adobe Reader, or an HTML browser, then the client application becomes the host application.

Designer includes a scripting model that provides scripting properties and methods for directly interfacing with a hosting application. For example, you can use the properties and methods in the host scripting model to provide PDF page navigation actions in Acrobat or Adobe Reader, or you can use the `importData` method to load data into your form.

You can reference the host script model syntax on any valid scripting event for form design objects using the following syntax for both FormCalc and JavaScript:

```
xfa.host.property_or_method
```

12.1. Host scripting model properties and methods

Using the host scripting model properties and methods, you can retrieve information and execute actions that are not otherwise accessible through calculations and scripts. For example, you can retrieve the name of the host application (such as Acrobat), or advance the current page on an interactive form. The following table lists the properties and methods that are available for the host scripting model.

Properties	Methods
<code>appType</code>	<code>beep</code>
<code>calculationsEnabled</code>	<code>exportData</code>
<code>currentPage</code>	<code>gotoURL</code>
<code>language</code>	<code>importData</code>
<code>name</code>	<code>messageBox</code>
<code>numPages</code>	<code>pageDown</code>
<code>platform</code>	<code>pageUp</code>
<code>title</code>	<code>print</code>

Properties	Methods
validationsEnabled	resetData
variation	response
version	setFocus

For more information about the host scripting model properties and methods, see the [Developer Center](#).

12.2. Comparing the host scripting model functionality

This table lists the Designer host scripting model properties and methods, and compares them to the equivalent expressions in the JavaScript Object Model in Acrobat.

For more information about the host scripting model properties and methods, see *Designer Help*, or see the [Scripting Reference](#).

Host scripting model properties and methods	JavaScript Object Model from Acrobat equivalent
<code>xfa.host.appType</code>	<code>app.viewerType</code>
<code>xfa.host.beep([INTEGER param])</code>	<code>app.beep([nType])</code>
<code>xfa.host.currentPage</code>	<code>doc.pageNum</code>
<code>xfa.host.exportData([STRING param1 [, BOOLEAN param2]])</code>	<code>doc.exportXFADData(cPath [, bXDP])</code>
<code>xfa.host.gotoURL(STRING param1)</code>	<code>doc.getURL(cURL, [bAppend])</code> or <code>app.launchURL(URL);</code>
<code>xfa.host.importData([STRING param])</code>	<code>doc.importXFADData(cPath)</code>
<code>xfa.host.language</code>	<code>app.language</code>
<code>xfa.host.messageBox(STRING param1 [, STRING param2 [, INTEGER param3 [, INTEGER param4]]])</code>	<code>app.alert(cMsg [, nIcon [, nType [, cTitle]]])</code>
<code>xfa.host.name</code>	<code>none</code>
<code>xfa.host.numPages</code>	<code>doc.numPages</code>

Host scripting model properties and methods	JavaScript Object Model from Acrobat equivalent
<code>xfa.host.pageDown()</code>	<code>doc.pageNum++</code>
<code>xfa.host.pageUp()</code>	<code>doc.pageNum--</code>
<code>xfa.host.platform</code>	<code>app.platform</code>
<code>xfa.host.print(BOOLEAN param1, INTEGER param2, INTEGER param3, BOOLEAN param4, BOOLEAN param5, BOOLEAN param6, BOOLEAN param7, BOOLEAN param8)</code>	<code>doc.print([bUI [, nStart [, nEnd [, bSilent [, bShrinkToFit [, bPrintAsImage [, bReverse [, bAnnotations]]]]]]])</code>
<code>xfa.host.resetData([STRING param])</code>	<code>doc.resetForm([aFields])</code>
<code>xfa.host.response(STRING param1 [, STRING param2 [, STRING param3 [, BOOLEAN param4]]])</code>	<code>app.response(cQuestion [, cTitle [, cDefault [, bPassword]]])</code>
<code>xfa.host.setFocus(STRING param)</code>	<code>field.setFocus() (Deprecated)</code>
<code>xfa.host.title</code>	<code>doc.title</code>
<code>xfa.host.variation</code>	<code>app.viewerVariation</code>
<code>xfa.host.version</code>	<code>app.viewerVersion</code>

RELATED LINKS:

[Referencing Objects in Calculations and Scripts](#)

13. Working with the Event Model

The event model stores object event properties. These properties are useful if you want to access values that are otherwise out of the scope of the events listed in the Show list within the Script Editor.

The event model controls the changes in a form that occur before, during, and after actions take place. These actions include dynamic form events, such as the point when the data and form design are merged but before pagination is applied, and also interactive form events, such as when a user updates the value of a field.

13.1. Event model properties and methods

Using the event object properties and methods, you can retrieve information and execute actions that otherwise are not accessible through calculations and scripts. For example, you can retrieve the full value of a field that otherwise would have part of the data stripped out because it is too long or otherwise invalid. Retrieving the full value of a field is useful in situations where you have to do extensive error checking.

Properties	Methods
<code>change</code>	<code>emit</code>
<code>className</code>	<code>reset</code>
<code>commitKey</code>	
<code>fullText</code>	
<code>keyDown</code>	
<code>modifier</code>	
<code>name</code>	
<code>newContentType</code>	
<code>newText</code>	
<code>prevContentType</code>	
<code>prevText</code>	
<code>reenter</code>	

Properties	Methods
selEnd	
selStart	
shift	
soapFaultCode	
soapFaultString	
target	

For more information about the event scripting model properties and methods, see the [Developer Center](#).

14. Moving from Scripting in Acrobat to Designer

Designer includes extensive scripting capabilities, including support for the most common JavaScript objects from Acrobat. When you convert an Acrobat form to Designer, most JavaScript scripts continue to work without requiring changes. However, you will need to manually convert some JavaScript scripts from Acrobat to maintain the behavior of your Acrobat form.

When converting scripts on your Acrobat form, note that Designer scripting differs from scripting in Acrobat in several ways:

Designer workspace

In the Designer workspace, you can change object properties and behaviors on your form without requiring you to create scripts.

Scripting languages

Designer includes support for JavaScript as well as for FormCalc, which is a simple calculation language. FormCalc includes built-in functions that perform many useful operations that would otherwise require extensive scripting.

Referencing objects, properties, and methods

Designer forms are highly structured; therefore, to reference specific objects, properties, or methods, you must include the appropriate reference syntax in your script. You can use the statement completion options in the Script Editor to assist you in creating reference syntaxes.

It is possible to continue to use JavaScript objects, properties, and methods from Acrobat in Designer. However, you should consider JavaScript from Acrobat only for tasks that you cannot perform using the XML Form Object Model in Designer. For example, you can use JavaScript from Acrobat to add attachments, bookmarks, and annotations; search or spell check the form; create reports; or access and manipulate metadata. You cannot use JavaScript from Acrobat to perform actions such as setting field values, adding new fields to a form, or deleting pages from a form.

***NOTE:** You cannot use Acrobat to add JavaScript scripts to a Designer form, including Acrobat forms that you have converted using Designer. When you view a Designer form in Acrobat, all JavaScript tools are unavailable.*

For more information about converting Acrobat scripting to Designer, see the article [Converting Acrobat JavaScript for Use in Designer Forms](#) in the Developer Center.

14.1. Converting Acrobat forms that contain scripts

One of the first steps in converting a form from Acrobat to Designer is to determine how much of the Acrobat scripting is supported by Designer and how much you must convert.

In general, you should convert all Acrobat scripting to an equivalent in Designer. Designer scripting takes full advantage of the highly structured nature of Designer forms, as well as useful forms-specific functionality, to make designing and implementing your forms solution faster and easier.

The Acrobat scripting you should retain include those that deal with the form's environment and peripheral operations, such as adding attachments or multimedia, performing searches, or creating reports and handling document metadata.

For more information about converting Acrobat scripting to Designer, see the article [Converting Acrobat JavaScript for Use in Designer Forms](#) in the Developer Center.

RELATED LINKS:

Moving from Scripting in Acrobat to Designer

Using JavaScript objects from Acrobat in Designer

JavaScript objects from Acrobat supported in Designer

14.2. Using JavaScript objects from Acrobat in Designer

In Designer, you can script against certain JavaScript objects in Acrobat by using the Acrobat scripting syntax. As a result, you can use the properties and methods of those objects on your form. For example, to display a message in the JavaScript Console from Acrobat, you can add the following script to the event of a form design object in Designer:

```
console.println("This message appears in the JavaScript  
Console.");
```

You can also have the form send itself by email by adding the following script to the `click` event of a button:

```
var myDoc = event.target;  
myDoc.mailDoc(true);
```

NOTE: In Designer, you must ensure that the scripting language for the event is set to JavaScript so that the script will execute correctly at run time.

You can also use references to the JavaScript objects in Acrobat in your reference syntax. For example, the following script gets the signed state of a signature field and takes an action based on the state:

```
// Proceed if the current field is not signed.  
var oState =
```

```
event.target.getField("form1[0].#subform[0].SignatureField1[0]")
.signatureValidate(); //Get the field's signed state.

if (oState == 0) {
...
}
```

NOTE: This example uses a fully qualified reference syntax to reference the text. For more information about referencing form design objects, see *Referencing object properties and values*.

When working with JavaScript from Acrobat in Designer, remember these points:

- In Designer, use `event.target` to access the `Doc` JavaScript object from Acrobat. In Acrobat, the `this` object is used to reference the `Doc` object; however, in Designer, the `this` object refers to the form design object to which the script is attached.
- The Script Editor has no statement completion for JavaScript objects from Acrobat. See the [JavaScript for Acrobat API Reference](#).
- The `Doc` method `event.target.importTextData("file.txt")` is not supported for dynamic XFA forms that have been certified.

For more information about converting Acrobat scripting to Designer, see the article [Converting Acrobat JavaScript for Use in Designer Forms](#) in the Developer Center.

RELATED LINKS:

- Moving from Scripting in Acrobat to Designer
- Converting Acrobat forms that contain scripts
- JavaScript objects from Acrobat supported in Designer

14.3. JavaScript objects from Acrobat supported in Designer

The following table lists the availability of the most commonly used Acrobat objects, properties, and methods in Designer, and provides information on any equivalent functionality in Designer. Although the table contains the most commonly used Acrobat objects, properties and methods, some are not listed, such as multimedia objects, because they are rarely used for forms.

In cases where no equivalent Designer functionality is listed, no direct Designer property or method can reproduce the Acrobat behavior. However, you can still create custom functions or scripts to replicate the Acrobat capability.

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
Annotobject properties and methods			

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
All properties and methods	Yes	None	Only forms with a fixed layout support the annotation layer.
appobject properties			
calculate	No	None	Designer includes the <code>execCalculate</code> method which initiates the <code>calculate</code> event. <code>execCalculate</code>
language	Yes	<code>xfa.host.language</code>	See the <code>language</code> property. <code>language</code>
monitors	Yes	None	
platform	Yes	<code>xfa.host.platform</code>	See the <code>platform</code> property. <code>platform</code>
plugins	Yes	None	
toolbar	Yes	None	
viewerType	Yes	<code>xfa.host.appType</code>	See the <code>appType</code> property. <code>appType</code>
viewerVariation	Yes	<code>xfa.host.variation</code>	See the <code>variation</code> property. <code>variation</code>
viewerVersion	Yes	<code>xfa.host.version</code>	See the <code>version</code> property. <code>version</code>
appobject methods			
addItem	Yes	None	
addSubMenu	Yes	None	
addToolButton	Yes	None	
alert	Yes	<code>xfa.host.messageBox()</code>	See the <code>messageBox</code> method. <code>messageBox</code>

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
beep	Yes	<code>xfa.host.beep()</code>	See the beep method. beep
browseForDoc	Yes	None	
clearInterval	Yes	None	
clearTimeOut	Yes	None	
execDialog	Yes	None	
execMenuItem	Yes	None	Executes the specified menu command. Use this method in Designer for File menu commands.
getNthPluginName	Yes	None	
getPath	Yes	None	
goBack	Yes	None	
goForward	Yes	None	
hideMenuItem	Yes	None	
hideToolBarButton	Yes	None	
launchURL	Yes	None	Designer includes the gotoURL method that loads a specified URL into the client application, such as Acrobat or Adobe Reader. gotoURL
listMenuItems	Yes	None	
listToolBarButtons	Yes	None	
mailGetAddrs	Yes	None	
mailMsg	Yes	None	
newDoc	Yes	None	This method can only be executed during batch, console, or menu events.
newFDF	No	None	

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
openDoc	Yes	None	
openFDF	No	None	
popUpMenuEx	Yes	None	
popUpMenu	Yes	None	
removeToolButton	Yes	None	
response	Yes	xfa.host.response()	See the responseMethod. response
setInterval	Yes	None	
setTimeout	Yes	None	
trustedFunction	Yes	None	
trustPropagatorFunction	Yes	None	This method is only available during batch, console, and application initialization.
Bookmark object properties and methods			
All properties and methods	Yes	None	
docobject properties			
author	Yes	None	
baseURL	Yes	None	
bookmarkRoot	Yes	None	
calculate	No	None	
dataObjects	Yes	None	
delay	No	None	

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
dirty	Yes	None	This JavaScript script for Designer saves a copy of a form and tests whether the form has changed: <pre>var sOrigXML = xfa.data.saveXML; if (sOrigXML != xfa.data.saveXML) {...}</pre>
disclosed	Yes	None	
documentFileName	Yes	None	
dynamicXFAForm	Yes	None	
external	Yes	None	
filesize	Yes	None	
hidden	Yes	None	
icons	Yes	None	
keywords	Yes	None	
layout	Yes	None	
media	Yes	None	
metadata	Yes	<code>xfa.form.desc</code>	See the descobject. desc
modDate	Yes	None	
mouseX mouseY	Yes	None	
noautocomplete	Yes	None	
nocache	Yes	None	

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
numFields	Yes	<code>xfa.layout.pageContent()</code>	The <code>pageContent</code> method returns a list of all objects of a particular type. However, you must execute the method for design views and master pages to scan the entire form. <code>pageContent</code>
numPages	Yes	<code>xfa.host.numPages</code> or <code>xfa.layout.absPageCount()</code> <code>xfa.layout.pageCount()</code>	The <code>numPages</code> property returns the page count for the rendered form in the client. See also the <code>absPageCount</code> and <code>pageCount</code> methods. <code>numPages</code> <code>absPageCount</code> <code>pageCount</code>
pageNum	Yes	<code>xfa.host.currentPage</code>	See the <code>currentPage</code> property. <code>currentPage</code>
pageNum--	Yes	<code>xfa.host.currentPage--</code> or <code>xfa.host.pageUp()</code>	See the <code>currentPage</code> property or the <code>pageUp</code> method. <code>currentPage</code> <code>pageUp</code>
pageNum++	Yes	<code>xfa.host.currentPage++</code> or <code>xfa.host.pageDown()</code>	See the <code>currentPage</code> property or the <code>pageDown</code> method. <code>currentPage</code> <code>pageDown</code>
path	Yes	None	
securityHandler	Yes	None	
templates	No	None	Use subform objects in Designer, and use properties and methods to add, remove, move, and set subform instances. Add and remove subform instances using scripting
title	Yes	<code>xfa.host.title</code>	See <code>title</code> .

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
document methods			
addAnnot	Yes	None	
addField	No	None	You must use forms that have a fixed layout in Designer, and then use the <code>instanceManager</code> object to add, remove, and set the number of instances of a particular object. <code>instanceManager</code> For more information, see Add and remove subform instances using scripting .
addIcon	Yes	None	
addLink	No	None	
addRecipientListCryptFilter	Yes	None	
addScript	Yes	None	
addThumbnails	No	None	
addWatermarkFromFile	Yes	None	
addWatermarkFromText	Yes	None	
addWeblinks	Yes	None	
appRightsSign	Yes	None	
appRightsValidate	Yes	None	
bringToFront	Yes	None	

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
calculateNow	No	<code>xfa.form.recalculate(1);</code> or <code>execCalculate()</code>	<code>recalculate</code> The <code>recalculate</code> method forces a specific set of scripts on calculate events to initiate. Boolean value indicates whether <code>True</code> (default) - all calculation scripts initiate; or <code>False</code> - only pending calculation scripts initiate. Designer <code>calculate</code> object controls whether a form filler can override a field's calculated value. <code>execCalculate</code> Alternatively, you can use the <code>execCalculate</code> method for each object for which you want to force a recalculation.
closeDoc	Yes	None	
createDataObject	Yes	None	
createTemplate	No	None	Designer forms do not have an equivalent to the concept of an Acrobat template. You must use subform objects in Designer.
deletePages	No	None	<code>instanceManager</code> In Designer, you can use the <code>instanceManager</code> object to remove the subform object that represents a page of your form. For more information, see Add and remove subform instances using scripting .
embedDocAsDataObject	Yes	None	
encryptForRecipients	Yes	None	

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
<code>encryptUsingPolicy</code>	Yes	None	
<code>exportAsText</code>	Yes	None	This method is only available in the JavaScript Console of the JavaScript Debugger in Acrobat or during batch processing.
<code>exportAsFDF</code>	No	<code>xfa.host.exportData()</code>	<code>exportData</code> The <code>exportData</code> method exports an XML or XDP file instead of an FDF file.
<code>exportAsXFDF</code>	No	<code>xfa.host.exportData()</code>	<code>exportData</code> The <code>exportData</code> method exports an XML or XDP file instead of an FDF file.
<code>exportDataObject</code>	Yes	None	
<code>exportXFADData</code>	No	<code>xfa.host.exportData()</code>	<code>exportData</code> The <code>exportData</code> method exports an XML or XDP file instead of an FDF file.
<code>extractPages</code>	No	None	
<code>flattenPages</code>	No	None	
<code>getAnnot</code>	Yes	None	
<code>getAnnots</code>	Yes	None	
<code>getDataObjectContents</code>	Yes	None	
<code>getField("FieldName")</code>	Yes	<code>xfa.resolveNode("FieldName")</code>	<code>resolveNode</code> The <code>resolveNode</code> method accesses the specified object in the source XML of the form.
<code>getLegalWarnings</code>	Yes	None	
<code>getLinks</code>	No	None	

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
getNthFieldName	Yes	You must loop through all objects with a similar class name until you reach the nth occurrence.	className See the className property.
getNthTemplate	No	None	
getOCGs	Yes	None	
getOCGOrder	Yes	None	
getPageBox	Yes	None	
getPageLabel	Yes	None	
getPageNthWord	Yes	None	
getPageNthWordQuads	Yes	None	
getPageNumWords	Yes	None	
getPageRotation	Yes	None	
getPrintParams	Yes	None	
getTemplate	No	None	
getURL	Yes	xfa.host.gotoURL ("http://www.adobe.com");	See the gotoURL method. gotoURL
gotoNamedDest	No	None	
importAnFDF	No	None	
importAnXFDF	Yes	None	
importDataObject	Yes	None	
importIcon	Yes	None	
importTextData	Yes	None	
importXFADData	No	xfa.host.importData ("filename.xdp");	See the importData method. importData

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
insertPages	No	None	
mailDoc	Yes	None	
mailForm	No	None	
movePage	No	None	
newPage	No	None	
openDataObject	Yes	None	
print	Yes	<code>xfa.host.print () ;</code>	See the <code>print</code> method. <code>print</code>
removeDataObject	Yes	None	
removeField	No	None	
removeIcon	Yes	None	
removeLinks	No	None	
removeScript	Yes	None	
removeTemplate	No	None	
removeThumbnails	No	None	
removeWeblinks	Yes	None	
replacePages	No	None	
resetForm	No	<code>xfa.host.resetData ()</code> or <code>xfa.event.reset ()</code>	The <code>resetData</code> method resets all field values on a form to the default values. The <code>reset</code> method resets all properties within the event model. <code>resetData</code> <code>reset</code>

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
saveAs	Yes	None	In Designer, the file must be saved at the application level. These scripts are examples of saving at the application level: <code>app.executeMenuItem ("SaveAs");</code> or <code>var myDoc = event.target;</code> <code>myDoc.saveAs();</code>
spawnPageFromTemplate	No	None	
setAction	No	None	
setPageLabel	Yes	None	
setPageRotation	No	None	
setPageTabOrder	No	None	In Designer, select Edit > Tab Order to set the tab order.
setScript	No	None	
submitForm	Yes	Use one of the submit button objects in Designer.	
eventobject properties			
change	Yes	<code>xfa.event.change</code>	change See the change property.
targetName	Yes	<code>xfa.event.target</code>	target See the target property.
fieldobject properties			
comb	No	None	

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
<code>charLimit</code>	No	<code>this.value.#text.maxChars</code>	In forms that have a fixed layout, character limit can be set in the Designer workspace. You can set fields on forms whose layout expands to accommodate all data. <code>maxChars</code>
<code>display = display.noView</code>	No	See Changing the presence of a form design object.	presence You can also set the <code>presence</code> property in the Designer workspace. You cannot use the <code>prePrint</code> event to change the presence of an object prior to printing.
<code>display = display.noPrint</code>	No	See Changing the presence of a form design object.	presence You can also set the <code>presence</code> property in the Designer workspace. You cannot use the <code>prePrint</code> event to change the presence of an object prior to printing.
<code>defaultValue</code>	No	None	Set the default field value in the Designer workspace.
<code>exportValues</code>	No	None	Set the export value in the Designer workspace.
<code>fillColor</code>	No	<code>xfa.form.Form1.NumericField1.fillColor</code>	<code>fillColor</code> See the <code>fillColor</code> property.
<code>hidden</code>	No	<code>this.presence = "invisible"</code> <code>this.presence = "visible"</code>	presence You can also set the <code>presence</code> property in the Designer workspace.
<code>multiline</code>	No	<code>this.ui.textEdit1.multiLine = "1";</code>	<code>multiLine</code> See the <code>multiLine</code> property.

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
password	No	None	Designer contains a Password Field that you can use for passwords on a form.
page	No	None	Not applicable for Designer forms.
print	No	<code>this.relevant = "-print";</code>	relevant See the <code>relevant</code> property.
radiosInUnison	No	None	Grouped radio buttons in Designer are mutually exclusive by default.
rect	Yes	You can get the height and width of a Designer form field by using the following reference syntax: <code>this.h; this.w;</code> Alternatively, you can retrieve the x and y coordinates of an object using the following reference syntax: <code>this.x; this.y;</code>	h, x, y See the <code>h</code> , <code>w</code> , <code>x</code> , and <code>y</code> properties.
required	No	<code>this.mandatory = "error";</code> or <code>this.validate.nullTest = "error";</code>	mandatory, <code>nullTest</code> See the <code>mandatory</code> and <code>nullTest</code> properties.
textColor	No	<code>this.fontColor</code>	fontColor See the <code>fontColor</code> property.
textSize	No	<code>this.font.size</code>	size See the <code>size</code> property.
textFont	No	<code>this.font.typeface</code>	typeface See the <code>typeface</code> property.

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
value	No	this.rawValue	rawValue See the rawValueproperty. value Designer fields have a valueproperty; it is not the equivalent of the Acrobat value property.
fieldobject methods			
clearItems	No	DropDownList1.clearItems();	clearItems The clearItemsmethod only applies to Drop-down List and List Box objects in Designer.
deleteItemAt	No	None	
getItemAt	No	None	
insertItemAt	No	DropDownList1.addItem (.....)	addItem See the addItemmethod.
isBoxChecked	No	if (CheckBox1.rawValue == 1)....	rawValue See the rawValueproperty.
isDefaultChecked	No	None	
setAction	No	None	Not applicable for Designer forms.
setFocus	Yes	xfa.host.setFocus ("TextField1.sourceExpression")	setFocus The setFocusmethod requires that the specified object have a unique name with respect to other objects on your form.
setItems	No	None	
setLock	Yes	None	
signatureGetModifications	Yes	None	
signatureGetSeedValue	Yes	None	

JavaScript in Acrobat	Designer support	JavaScript-equivalent in Designer	Comments
signatureInfo	Yes	None	
signatureSetSeedValue	Yes	None	
signatureSign	Yes	None	
signatureValidate	Yes	None	
searchobject method			
search.query("<your text>");	Yes	None	The “.” (double period) FormCalc shortcut syntax allows you to search for objects within the XML Form Object Model. For more information, see FormCalc referencesyntax shortcuts.
SOAPobject method			
All properties and methods	Yes	None	

RELATED LINKS:

[Working with the Event Model](#)

15. Examples of Common Scripting Tasks

The scripting examples demonstrate quick and simple techniques that you can apply to your own work.

For more examples and ideas, visit the [DeveloperCenter](#).

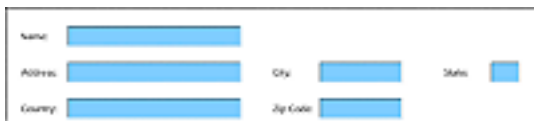
15.1. Changing the background colors of fields, fillable areas, and subforms

This example demonstrates how to change the background color of subforms, fields, and fillable areas on a form in response to form filler interaction at run time.

In this example, clicking a button changes the background color of an associated object.



A screenshot of a form with the following fields: Name, Address, City, State, Country, and Zip Code. Below the fields are four buttons: Subform, Fields, Fillable Areas, and Clear all. The form has a white background.



The same form as above, but the background color is now light blue. The buttons are still visible.



The same form as above, but the background color is now a medium blue. The buttons are still visible.



The same form as above, but the background color is now a dark blue. The buttons are still visible.

NOTE: To manipulate the background color of objects at run time, you must save your form as an Acrobat Dynamic XML Form file.

To see this scripting example and others, visit the [DeveloperCenter](#).

15.1.1. Scripting the subform and text field background colors

You set the subform and the text field background colors by using the `fillColor` method. For example, the following line is the script for the subform:

```
Subform1.fillColor = "17,136,255";
```

The following lines make up the script for the background color of the text fields:

```
Subform1.Name.fillColor = "102,179,255";
Subform1.Address.fillColor = "102,179,255";
Subform1.City.fillColor = "102,179,255";
Subform1.State.fillColor = "102,179,255";
Subform1.ZipCode.fillColor = "102,179,255";
Subform1.Country.fillColor = "102,179,255";
```

15.1.2. Scripting the fillable area background color

When setting the background color or the fillable area for each text field, your scripts must access properties that require a reference syntax expression that includes the number sign (#). Because JavaScript does not interpret the number sign (#) properly in reference syntax expressions, the script uses the `resolveNode` method to resolve the expression.

```
xfa.resolveNode("Subform1.Name.ui.#textEdit.border.fill.color").value =
"153,204,255";
xfa.resolveNode("Subform1.Address.ui.#textEdit.border.fill.color").value =
"153,204,255";
xfa.resolveNode("Subform1.City.ui.#textEdit.border.fill.color").value =
"153,204,255";
xfa.resolveNode("Subform1.State.ui.#textEdit.border.fill.color").value =
"153,204,255";
xfa.resolveNode("Subform1.ZipCode.ui.#textEdit.border.fill.color").value =
"153,204,255";
xfa.resolveNode("Subform1.Country.ui.#textEdit.border.fill.color").value =
"153,204,255";
```

15.1.3. Scripting the Clear All button

The script for the Clear All button uses the `remerge` method to remerge the form design and form data. In this case, the method effectively restores the fields, fillable areas, and subforms to their original state.

```
xfa.form.remerge();
```

15.2. Hiding and showing objects

This example demonstrates how to hide buttons when printing a form, as well as how to hide and show objects by changing the presence values at run time.

NOTE: You can also use the Action Builder dialog box on the Tools menu to hide and show objects in forms that have a flowable layout, without writing scripts. For information, see *Building actions in forms*

In this example, all form objects are showing in the form.

Form Objects

Name:	<input type="text" value="Jim Stall"/>
Address:	<input type="text" value="84 Brownstone St."/>
City:	<input type="text" value="Sunnydale"/>
State:	<input type="text"/>
Zip Code:	<input type="text"/>
Country:	<input type="text"/>
<input type="button" value="Submit"/>	
<input type="button" value="Submit by Email"/>	
<input type="button" value="Print Form"/>	

Presence Values

Subform:	Values:
<input type="text" value="Subform1"/>	<input type="text" value="Visible"/>
Text Fields:	Values:
<input type="text"/>	<input type="text" value="Visible"/>
Buttons:	Values:
<input type="text"/>	<input type="text" value="Visible"/>
<input type="button" value="Reset"/>	

The form filler can use the drop-down lists in the Presence Values area to show or hide objects. In the following diagram, the Address field is hidden and the form layout has adjusted accordingly. The Print Form button is also invisible.

Form Objects

A screenshot of a web form titled "Form Objects". It contains the following elements:

- Name:** A text input field containing "Jim Stall".
- City:** A text input field containing "Sunnydale".
- State:** An empty text input field.
- Zip Code:** An empty text input field.
- Country:** An empty text input field.
- Buttons:** Two buttons at the bottom: "Submit" and "Submit by Email".

Presence Values

A screenshot of a web form titled "Presence Values". It is organized into three rows, each with a category dropdown and a corresponding "Values" dropdown:

- Row 1:** "Subform:" dropdown set to "Subform1"; "Values:" dropdown set to "Visible".
- Row 2:** "Text Fields:" dropdown set to "Address"; "Values:" dropdown set to "Hidden (Exclude from Layout)".
- Row 3:** "Buttons:" dropdown set to "Print Form"; "Values:" dropdown set to "Invisible".

 A "Reset" button is located at the bottom center of the form.

NOTE: To hide and show objects at run time, you must save your form as an Acrobat Dynamic PDF Form file.

To see this scripting example and others, visit visit the [DeveloperCenter](#).

15.2.1. Scripting the presence values for subforms

The script for the subform presence values uses a switch statement to handle the three presence options that a form filler can apply to the subform object:

```
switch(xfa.event.newText) {
case 'Invisible':
Subform1.presence = "invisible";
break;
case 'Hidden (Exclude from Layout)':
Subform1.presence = "hidden";
break;
default:
Subform1.presence = "visible";
break;
}
```

15.2.2. Scripting the presence values for text fields

The script for the text fields presence values requires two variables. The first variable stores the number of objects contained in Subform1:

```
var nSubLength = Subform1.nodes.length;
```

The second variable stores the name of the text field that the form filler selects in the Text Fields drop-down list:

```
var sSelectField = fieldList.rawValue;
```

The following script uses the `replace` method to remove all of the spaces from the name of the field stored in the `sSelectField` variable, which allows the value of the drop-down list to match the name of the object in the Hierarchy palette:

```
sSelectField = sSelectField.replace(' ', '');
```

This script uses a `For` loop to cycle through all of the objects contained in Subform1:

```
for (var nCount = 0; nCount < nSubLength; nCount++) {
```

If the current object in Subform1 is of type `field` and the current object has the same name as the object that the form filler selected, the following switch cases are performed:

```
    if ((Subform1.nodes.item(nCount).className == "field") &  
        (Subform1.nodes.item(nCount).name == sSelectField)) {
```

The following script uses a switch statement to handle the three presence values that a form filler can apply to text field objects:

```
switch(xfa.event.newText) {  
    case 'Invisible':  
        Subform1.nodes.item(nCount).presence = "invisible";  
        break;  
    case 'Hidden (Exclude from Layout)':  
        Subform1.nodes.item(nCount).presence = "hidden";  
        break;  
    default:  
        Subform1.nodes.item(nCount).presence = "visible";  
        break;  
}  
}  
}
```

15.2.3. Scripting the presence values for buttons

The script for the buttons presence values requires two variables. This variable stores the number of objects contained in Subform1:

```
var nSubLength = Subform1.nodes.length;
```


This variable stores the name of the button that the form filler selects in the Buttons drop-down list:

```
var sSelectButton = buttonList.rawValue;
```

The following script uses the `replace` method to remove all of the spaces from the name of the button stored in the `sSelectField` variable, which allows the value of the drop-down list to match the name of the object in the Hierarchy palette:

```
sSelectButton = sSelectButton.replace(/\s/g, '');
```

This script uses a `For` loop to cycle through all of the objects contained in `Subform1`:

```
for (var nCount = 0; nCount < nSubLength; nCount++) {
```

If the current object in `Subform1` is of type `field` and the current object has the same name as the object that the form filler selected, perform the following switch cases:

```
if ((Subform1.nodes.item(nCount).className == "field") &  
Subform1.nodes.item(nCount).name == sSelectButton)) {
```

This script uses a `switch` statement to handle the five presence values that a form filler can apply to button objects.

NOTE: *The relevant property indicates whether an object should appear when the form is printed.*

```
switch(xfa.event.newText) {  
case 'Invisible':  
Subform1.nodes.item(nCount).presence = "invisible";  
break;  
case 'Hidden (Exclude from Layout)':  
Subform1.nodes.item(nCount).presence = "hidden";  
break;  
case 'Visible (but Don\'t Print)':  
Subform1.nodes.item(nCount).presence = "visible";  
Subform1.nodes.item(nCount).relevant = "-print";  
break;  
case 'Invisible (but Print Anyway)':  
Subform1.nodes.item(nCount).presence = "invisible";  
Subform1.nodes.item(nCount).relevant = "+print";  
break;  
default:  
Subform1.nodes.item(nCount).presence = "visible";  
break;  
}  
}  
}
```

15.2.4. Scripting for resetting the drop-down lists

Use the `resetData` method to reset all of the drop-down lists to their default values:

```
xfa.host.resetData();
```

Use the `remerge` method to remerge the form design and form data. In this case, the method effectively returns the objects in the Form Objects area to their original states:

```
xfa.form.remerge();
```

15.3. Excluding an object from the tabbing order

This example demonstrates how to exclude an object from the default tabbing sequence. In this example, a user would begin in `TextField1` and use the Tab button to navigate to `TextField2` and then to `TextField3`. However, the drop-down list object, `DropDownList1`, is configured to display when the user's cursor enters `TextField2`.



In this case, by default, the user's experience would be to move sequentially in the following order:



To exclude `DropDownList1` from the tabbing sequence, you would add the following scripts to the `TextField2` object:

Event	Script
enter	// This conditional statement displays DropDownList3 to the user // and sets the focus of the client application to TextField2. if (DropDownList3.presence != "visible") { DropDownList3.presence = "visible"; xfa.host.setFocus(this); }
exit	// This conditional statement tests to see if the user is // pressing the Shift key while pressing the Tab key. If Shift is // held down, then the client application focus returns to // TextField1, otherwise the focus is set to TextField3. The // experience for the user is that DropDownList3 is not a // part of the tabbing order. var isShiftDown = xfa.event.shift; if (isShiftDown) { xfa.host.setFocus(TextField1); } else { xfa.host.setFocus(textField3); }

15.4. Changing the visual properties of an object on the client

The example demonstrates how to manipulate the visual properties of an object; in this case, a text field. For example, selecting the Make the Field Wider check box expands the fillable area of the text field to four inches.



NOTE: To alter the visual properties of objects on the client, you must save your form as an Acrobat Dynamic PDF Form file.

In this example, the check boxes do not have unique object names; therefore, Designer assigns an instance value to reference the object. The check box script uses an `if-else` statement to give the effect of selecting and deselecting.

To see this scripting example and others, visit the [DeveloperCenter](#).

15.4.1. Scripting for the Move the Field check box

When the check box is selected, the field is moved according to the x and y settings. When the check box is deselected, the field is returned to its original location.

```
if (CheckBox1.rawValue == true) {
  TextField.x = "3.0in";
  TextField.y = "3.5in";
}
else {
  TextField.x = "1in";
  TextField.y = "3in";
}
```

15.4.2. Scripting for the Make the Field Wider check box

When the check box is selected, the field changes to 4 inches. When the check box is deselected, the field width changes to 2.5 inches.

```
if (CheckBox2.rawValue == true)
  TextField.w = "4in";
```

```
else  
TextField.w = "2.5in";
```

15.4.3. Scripting for the Make the Field Taller check box

When the check box is selected, the field height changes to 1.5 inches. When the check box is deselected, the field height changes to .5 inches.

```
if (CheckBox3.rawValue == true)  
TextField.h = "1.5in";  
else  
TextField.h = "0.5in";
```

15.4.4. Scripting for the Change the Border Color of the Object check box

When the check box is selected, the field border changes to red. When the check box is deselected, the field border changes to white.

```
if (CheckBox4.rawValue == true)  
TextField.border.edge.color.value = "255,0,0";  
else  
TextField.border.edge.color.value = "255,255,255";
```

15.4.5. Scripting for the Change the Fill Color of the Fillable Area check box

When the check box is selected, the fillable area of the text field changes to green. When the check box is deselected, the fillable area of the text field changes to white.

```
if (CheckBox5.rawValue == true) {  
xfa.resolveNode("TextField.ui.#textEdit.border.fill.color").value = "0,255,0";  
}  
else {  
xfa.resolveNode("TextField.ui.#textEdit.border.fill.color").value =  
"255,255,255";  
}
```

15.4.6. Scripting for the Expand to Fit the Width of the Value check box

When the check box is selected, the fillable area of the text field adjusts to accommodate the value. When the check box is deselected, the fillable area of the text field does not adjust.

```
if (CheckBox6.rawValue == true)  
TextField.minW = "0.25in";  
else  
TextField.maxW = "2.5in";
```

15.4.7. Scripting for the Make the Field Disappear check box

When the check box is selected, the field is hidden. When the check box is deselected, the field is visible.

```
if (CheckBox7.rawValue == true)
TextField.presence = "hidden";
else
TextField.presence = "visible";
```

15.4.8. Scripting for the Change the Font of the Value check box

When the check box is selected, the font of the value changes to Courier New. When the check box is deselected, the font of the value changes to Myriad Pro.

```
if (CheckBox8.rawValue == true)
TextField.font.typeface = "Courier New";
else
TextField.font.typeface = "Myriad Pro";
```

15.4.9. Scripting for the Change the Size of the Font check box

When the check box is selected, the font size changes to 14 pt. When the check box is deselected, the font size changes to 10 pt.

```
if (CheckBox9.rawValue == true)
TextField.font.size = "14pt";
else
TextField.font.size = "10pt";
```

15.4.10. Scripting for the Align Text Field Value Vertically check box

When the check box is selected, the text field value is aligned to the top. When the check box is deselected, the text field value is aligned to the middle.

```
if (CheckBox10.rawValue == true)
TextField.param.vAlign = "top";
else
TextField.param.vAlign = "middle";
```

15.4.11. Scripting for the Align Text Field Value Horizontally check box

When the check box is selected, the text field value is aligned to the center. When the check box is deselected, the text field value is aligned to the left.

```
if (CheckBox11.rawValue == true)
TextField.param.hAlign = "center";
```

```
else  
TextField.para.hAlign = "left";
```

15.4.12. Scripting for the Display a Set Value check box

When the check box is selected, the value that is defined by using a script appears in the text field. When the check box is deselected, the default value (which is also defined by using a script) appears in the text field.

```
if (CheckBox12.rawValue == true)  
TextField.rawValue = "This is a value set using a script.";  
else  
TextField.rawValue = "This is a default value.";
```

15.4.13. Scripting for the Change the Caption Text check box

When the check box is selected, the alternate caption text that is defined by using a script appears as the caption. When the check box is deselected, the default caption (which is also defined by using a script) appears in the text field.

```
if (CheckBox13.rawValue == true)  
xfa.resolveNode("TextField.caption.value.#text").value = "Alternate Caption:";  
else  
xfa.resolveNode("TextField.caption.value.#text").value = "Caption:";
```

15.4.14. Scripting for the Change Field Border from 3D to Solid check box

When the check box is selected, the field border changes to a solid box. When the check box is deselected, the field border changes to 3D.

```
if (CheckBox14.rawValue == true)  
xfa.resolveNode("TextField.ui.#textEdit.border.edge").stroke = "solid";  
else  
xfa.resolveNode("TextField.ui.#textEdit.border.edge").stroke = "lowered";
```

15.4.15. Scripting for the Clear All Check Boxes button

Use the `resetData` method to reset all of the check boxes to their default value (Off).

```
xfa.host.resetData();
```

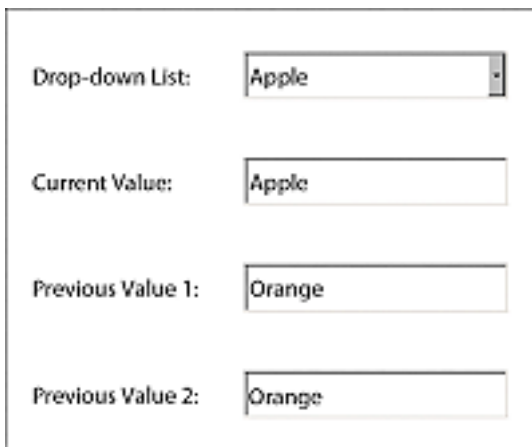
Use the `remerge` method to remerge the form design and form data. In this case, the method effectively returns the text field to its original state.

```
xfa.form.remerge();
```

15.5. Getting the current or previous value of a drop-down list

This example demonstrates how to obtain the current value of a drop-down list as well as the different ways to access the previous value of a drop-down list on a form. In addition to the actual scripts that set the current and previous values, it is important to note that the scripts are located on the `change` event for the drop-down list.

In the following example, when a form filler selects a value from the drop-down list, the selected value appears in the Current Value field. Then, when the form filler selects another value from the drop-down list, the new value appears in the Current Value List and the previous value appears in the Previous Value 1 field.



The screenshot shows a form with four rows. The first row is labeled 'Drop-down List:' and contains a drop-down menu with 'Apple' selected. The second row is labeled 'Current Value:' and contains a text field with 'Apple'. The third row is labeled 'Previous Value 1:' and contains a text field with 'Orange'. The fourth row is labeled 'Previous Value 2:' and contains a text field with 'Orange'.

NOTE: Each of the methods for obtaining the previous value of a drop-down list uses a different script. The Previous Value 1 text field is populated by a direct reference to the `rawValue` property of the drop-down list, whereas the Previous Value 2 text field is populated using the `prevText` property. For consistent results, it is recommended that you access the previous value by using the `prevText` property.

To see this scripting example and others, visit the [DeveloperCenter](#).

15.5.1. Scripting for populating the Current Value text field

Populate the value of the Current Value text field by using the `newText` property:

```
CurrentValue.rawValue = xfa.event.newText;
```

15.5.2. Scripting for populating the Previous Value 1 text field

Populate the value of the Previous Value 1 text field by referencing the `rawValue` of the drop-down list:

```
PreviousValue1.rawValue = DropDownList.rawValue;
```

15.5.3. Scripting for populating the Previous Value 2 text field

Populate the value of the Previous Value 2 text field by using the `prevText` property:

```
PreviousValue2.rawValue = xfa.event.prevText;
```

15.6. Preserving rich text formatting when copying field values

This example demonstrates how to maintain rich text formatting of field data when copying values between fields.



TextField1 and TextField2 are configured to Allow Multiple Lines and display rich text formatting.

The Copy Rich Text button copies the value of TextField1, including rich text formatting, and pastes it in TextField2.

15.6.1. Scripting for the Copy Rich Text button

Rich text field values are stored in XML format within a child object of the field that contains the value. The following script, located on the click event of the Copy Rich Text button, uses the `saveXML` method to store the XML definition of the rich text value. Subsequently, the XML data is loaded into the corresponding child object of TextField2.

```
var richText = TextField1.value.exData.saveXML();  
TextField2.value.exData.loadXML(richText, 1, 1);
```

In this example, the rich text value is set to overwrite the existing value of TextField2. Adjusting the script to the following would append the rich text data to the current value of TextField2:

```
var richText = TextField1.value.exData.saveXML();  
TextField2.value.exData.loadXML(richText, 1, 0);
```


15.7. Adjusting the height of a field at run time

The example demonstrates how to expand a field to match the height of the content in another field.

In this example, when the form filler types multiple lines in TextField1 and then clicks the Expand button, the height of TextField2 increases to match the height of TextField1.



To see this scripting example and others, visit the [DeveloperCenter](#).

15.7.1. Scripting for the Expand button

The following script is for the Expand button:

```
var newHeight = xfa.layout.h(TextField1, "in");
TextField2.h = newHeight + "in";
```

15.8. Setting a field as required at run time

This example demonstrates how to make a field required at run time.

In this example, when the Set as Required button is clicked, if the form filler attempts to submit a form without typing some text in TextField1, an error message appears.

TextField1

Set as Required

email

To see this scripting example and others, visit the [DeveloperCenter](#).

15.8.1. Scripting for the Set as Required button

The following script is for the Set as Required button:

```
TextField1.validate.nullTest = "error";
```

You can also use one of these two scripts:

```
TextField1.mandatory = "error"  
TextField1.mandatoryMessage = "this field is mandatory!"
```

15.9. Calculating the field sums

This example demonstrates how to calculate the sums of fields located at different levels of the form hierarchy when the form filler opens the form in a client application, such as Acrobat Professional, Adobe Reader, or HTML client.

NumericField1	3
NumericField1	3
NumericField1	3
Sum	<hr/> 9

To see this scripting example and others, visit [DeveloperCenter](#).

15.9.1. Scripting for calculating the sum of repeating fields in a form

To calculate the sum of repeating fields in a form, you add a `calculate` event to the Sum field:

```
var fields = xfa.resolveNodes("NumericField1[*]");

var total = 0;
for (var i=0; i <= fields.length-1; i++) {
    total = total + fields.item(i).rawValue;
}

this.rawValue = total;
```

15.9.2. Scripting for calculating the sum of repeating fields

Similarly, to calculate the sum of repeating fields, you add a `calculate` event to the Sum field:

```
var fields = xfa.resolveNodes("detail[*].NumericField1");

var total = 0;
for (var i=0; i <= fields.length-1; i++) {
    total = total + fields.item(i).rawValue;
}

this.rawValue = total;
```

15.9.3. Scripting to calculate the sum of the fields on the page

To calculate the sum of the fields on the page, you add a `calculate` event to the Sum field:

```
var fields = xfa.layout.pageContent(0 , "field", 0);

var total = 0;
for (var i=0; i <= fields.length-1; i++) {
  if (fields.item(i).name == "NumericField1") {
    total = total + fields.item(i).rawValue;
  }
}

this.rawValue = total;
```

15.10. Highlighting fields in response to form filler interaction

This example demonstrates how to highlight the current field that a form filler is working with, highlight fields that a form filler is required to fill, and use message boxes to provide feedback to the form filler.

In this example, an asterisk (*) appears to the right of the required fields. When a field is selected, the field border changes to blue. If the form filler clicks the Verify Data button without having filled the required fields, a message appears and the field changes to red. If all the required fields are filled, a confirmation message appears when the form filler clicks the Verify Data button.



To see this scripting example and others, visit [DeveloperCenter](#).

15.10.1. Scripting for adding a blue border around a selected field

To add the blue border around the selected field, add the following scripts to each text field:

For example, add an `enter` event to the Name field:

```
Name.border.edge.color.value = "0,0,255";
```

For example, add an `exit` event to the Name field:

```
Name.border.edge.color.value = "255,255,255";
```

For example, add a `mouseenter` event to the Name field:

```
Name.border.edge.color.value = "0,0,255";
```

For example, add a `mouseleave` event to the Name field:

```
Name.border.edge.color.value = "255,255,255";
```

15.10.2. Scripting for the Verify Data button

The following script, which is created for the Verify Data button, performs a series of checks to verify that the required fields contain data. In this case, each field is individually checked to verify that the value of the field is non-null or an empty string. If the value of the field is null or an empty string, an alert message appears indicating that data must be input into the field and the background color of the fillable area is changed to red.

Use this variable to indicate whether a field does not contain data:

```
var iVar = 0;
```

```
if ((Name.rawValue == null) || (Name.rawValue == "")) {  
  xfa.host.messageBox("Please enter a value in the Name field.");
```

This script changes the color of the fillable area of the text field:

```
xfa.resolveNode("Name.ui.#textEdit.border.edge").stroke = "solid";  
xfa.resolveNode("Name.ui.#textEdit.border.fill.color").value = "255,100,50";
```

```
// Set the variable to indicate that this field does not contain data.  
iVar = 1;  
}  
else {  
  // Reset the fillable area of the text field.  
  xfa.resolveNode("Name.ui.#textEdit.border.edge").stroke = "lowered";  
  xfa.resolveNode("Name.ui.#textEdit.border.fill.color").value = "255,255,255";  
}
```

```
if ((Address.rawValue == null) || (Address.rawValue == "")) {  
  xfa.host.messageBox("Please enter a value in the Address field.");
```

This script changes the color of the fillable area of the text field:

```
xfa.resolveNode("Address.ui.#textEdit.border.edge").stroke = "solid";  
xfa.resolveNode("Address.ui.#textEdit.border.fill.color").value = "255,100,50";
```

This script sets the variable to indicate that this field does not contain data:

```
iVar = 1;  
}  
else {
```

This script resets the fillable area of the text field:

```
xfa.resolveNode("Address.ui.#textEdit.border.edge").stroke = "lowered";
xfa.resolveNode("Address.ui.#textEdit.border.fill.color").value =
"255,255,255";
}
```

```
if ((City.rawValue == null) || (City.rawValue == "")) {
xfa.host.messageBox("Please enter a value in the City field.");
```

This script changes the color of the fillable area of the text field:

```
xfa.resolveNode("City.ui.#textEdit.border.edge").stroke = "solid";
xfa.resolveNode("City.ui.#textEdit.border.fill.color").value = "255,100,50";
```

This script sets the variable to indicate that this field does not contain data:

```
iVar = 1;
}
else {
```

This script resets the fillable area of the text field:

```
xfa.resolveNode("City.ui.#textEdit.border.edge").stroke = "lowered";
xfa.resolveNode("City.ui.#textEdit.border.fill.color").value = "255,255,255";
}
```

```
if ((State.rawValue == null) || (State.rawValue == "")) {
xfa.host.messageBox("Please enter a value in the State field.");
```

This script changes the color of the fillable area of the text field:

```
xfa.resolveNode("State.ui.#textEdit.border.edge").stroke = "solid";
xfa.resolveNode("State.ui.#textEdit.border.fill.color").value = "255,100,50";
```

This script sets the variable to indicate that this field does not contain data:

```
iVar = 1;
}
else {
```

This script resets the fillable area of the text field:

```
xfa.resolveNode("State.ui.#textEdit.border.edge").stroke = "lowered";
xfa.resolveNode("State.ui.#textEdit.border.fill.color").value = "255,255,255";
}
```

```
if ((ZipCode.rawValue == null) || (ZipCode.rawValue == "")) {
xfa.host.messageBox("Please enter a value in the Zip Code field.");
```

This script changes the color of the fillable area of the text field:

```
xfa.resolveNode("ZipCode.ui.#textEdit.border.edge").stroke = "solid";
xfa.resolveNode("ZipCode.ui.#textEdit.border.fill.color").value = "255,100,50";
```

This script sets the variable to indicate that this field does not contain data:

```
iVar = 1;
}
else {
```

This script resets the fillable area of the text field:

```
xfa.resolveNode("ZipCode.ui.#textEdit.border.edge").stroke = "lowered";
xfa.resolveNode("ZipCode.ui.#textEdit.border.fill.color").value =
"255,255,255";
}
```

```
if ((Country.rawValue == null) || (Country.rawValue == "")) {
xfa.host.messageBox("Please enter a value in the Country field.");
}
```

This script changes the color of the fillable area of the text field:

```
xfa.resolveNode("Country.ui.#textEdit.border.edge").stroke = "solid";
xfa.resolveNode("Country.ui.#textEdit.border.fill.color").value = "255,100,50";
```

This script sets the variable to indicate that this field does not contain data.

```
iVar = 1;
}
else {
```

This script resets the fillable area of the text field.

```
xfa.resolveNode("Country.ui.#textEdit.border.edge").stroke = "lowered";
xfa.resolveNode("Country.ui.#textEdit.border.fill.color").value =
"255,255,255";
}
```

If all of the required fields contain data, the `iVar` variable is set to zero, and a confirmation message appears:

```
if (iVar == 0) {
xfa.host.messageBox("Thank you for inputting your information.");
}
```

15.11. Resetting the values of the current subform

This example demonstrates how to reset the values of a specific set of fields, not the whole form. To do this, reset only the fields in the required subform object.

In this example, the form filler can click the Clear button to reset the field values.

1	<input type="text"/>	<input type="text"/>	<input type="text"/>	Clear
2	<input type="text"/>	<input type="text"/>	<input type="text"/>	Clear
3	<input type="text"/>	<input type="text"/>	<input type="text"/>	Clear

To see this scripting example and others, visit the [DeveloperCenter](#).

15.11.1. Scripting for the values that appear in the left column

Type this script for the values appearing in the left column:

```
this.rawValue = this.parent.index + 1;
```

To reset the default values add a `click` event to the Clear button. You need a dynamic reference syntax expression because the detail is a repeating subform and must be reflected in the reference syntax expression. In this situation, it is easier to build the `resetData` parameters separately.

```
var f1 = this.parent.somExpression + ".TextField2" + ",";
var f2 = f1 + this.parent.somExpression + ".DropDownList1" + ",";
var f3 = f2 + this.parent.somExpression + ".NumericField1";

// ...and pass the variable as a parameter.
xfa.host.resetData(f3);
```

15.12. Changing the presence of a form design object

Designer provides the following presence settings for the different objects on a form through various tabs in the Object palette. The Invisible and Hidden (Exclude from Layout) settings are unavailable for groups, content areas, master pages, page set, and subform set objects.

NOTE: To change the presence setting of an object by using scripts, you must change the value of two underlying XML Form Object Model properties: *presence* and *relevant*.

The following table lists the presence settings and the corresponding reference syntax.

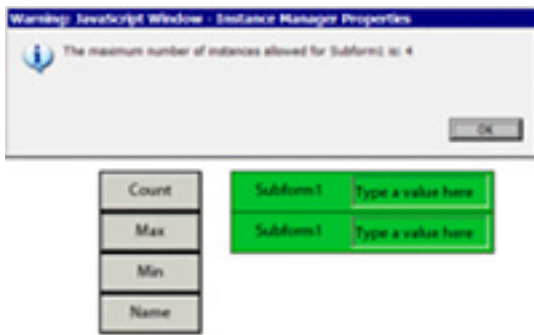
Presence setting	Reference syntax
Visible	FormCalc <code>ObjectName.presence = "visible"</code> JavaScript <code>ObjectName.presence = "visible";</code>
Visible (Screen Only)	FormCalc <code>ObjectName.presence = "visible"</code> <code>ObjectName.relevant = "-print"</code> JavaScript <code>ObjectName.presence = "visible";</code> <code>ObjectName.relevant = "-print";</code>

Presence setting	Reference syntax
Visible (Print Only)	FormCalc <i>ObjectName.presence</i> = "visible" <i>ObjectName.relevant</i> = "+print" JavaScript <i>ObjectName.presence</i> = "visible"; <i>ObjectName.relevant</i> = "+print";
Invisible	FormCalc <i>ObjectName.presence</i> = "invisible" JavaScript <i>ObjectName.presence</i> = "invisible";
Hidden (Exclude from Layout)	FormCalc <i>ObjectName.presence</i> = "hidden" JavaScript <i>ObjectName.presence</i> = "hidden";
One-sided Printing Only	FormCalc <i>ObjectName.presence</i> = "simplex" JavaScript <i>ObjectName.presence</i> = "simplex";
Two-sided Printing Only	FormCalc <i>ObjectName.presence</i> = "duplex" JavaScript <i>ObjectName.presence</i> = "duplex";

15.13. Using the properties of the instance manager to control subforms

This example demonstrates how to use the properties of the instance manager (which is part of the XML Form Object Model) to retrieve information about subforms at run time.

In the following form, the four buttons provide information about Subform1 by using the instance manager's scripting properties. For example, when the form filler clicks the Max button, a message describing the allowed maximum number of supported Subform1 instances appears.



15.13.1. Scripting for the message box to output the value of the count property

The following script uses the `messageBox` method to output the value of the `count` property:

```
xfa.host.messageBox("The current number of Subform1 instances on the form is:" + properties.Subform1.instanceManager.count, "Instance Manager Properties", 3);
```

You can also write this script by using the underscore (`_`) notation to reference the `count` property of the instance manager, as shown here:

```
xfa.host.messageBox("The current number of Subform1 instances on the form is: " + properties._Subform1.count, "Instance Manager Properties", 3);
```

The underscore (`_`) notation is especially important if no subform instances currently exist on the form.

15.13.2. Scripting for the message box to output the value of the max property

The following script uses the `messageBox` method to output the value of the `max` property:

```
xfa.host.messageBox("The maximum number of instances allowed for Subform1 is: " + properties.Subform1.instanceManager.max, "Instance Manager Properties", 3);
```

You can also write this script by using the underscore (`_`) notation to reference the `max` property of the instance manager, as shown here:

```
xfa.host.messageBox("The maximum number of instances allowed for Subform1 is: " + properties._Subform1.max, "Instance Manager Properties", 3);
```

15.13.3. Scripting for the message box to output the value of the min property

The following script uses the `messageBox` method to output the value of the `min` property:

```
xfa.host.messageBox("The minimum number of instances allowed for Subform1 is: " + properties.Subform1.instanceManager.min, "Instance Manager Properties", 3);
```

You can also write this script by using the underscore (_) notation to reference the `min` property of the instance manager, as shown here:

```
xfa.host.messageBox("The minimum number of instances allowed for Subform1  
is: " + properties._Subform1.min, "Instance Manager Properties", 3);
```

15.13.4. Scripting for the message box to output the name of the subform property

The following script uses the `messageBox` method to output the name of the subform property:

```
xfa.host.messageBox("The name of the subform using the instance manager name  
property is: " + properties.Subform1.instanceManager.name +  
".\n\nNote: This value is different than the value returned by the name  
property for the Subform1 object." , "Instance Manager Properties", 3);
```

You can also write this script by using the underscore (_) notation to reference the `name` property of the instance manager, as shown here:

```
xfa.host.messageBox("The name of the subform using the instance manager name  
property is: " + properties._Subform1.name +  
".\n\nNote: This value is different than the value returned by the name  
property for the Subform1 object." , "Instance Manager Properties", 3);
```

15.14. Using the methods of the instance manager to control subforms

This example demonstrates how to use the methods of the instance manager (which is part of the XML Form Object Model) to perform operations on subform objects at run time. For example, you can add remove instances of a particular subform, table, or table row.

In the following form, the form filler uses the four buttons to use the various instance manager scripting methods. For example, when the form filler clicks the Add button a new Subform2 instance is added to the form.

Add	Subform2	Type a value here
Remove	Subform2	Type a value here
Set		
Move		

Add	Subform2	Type a value here
Remove	Subform2	Type a value here
Set	Subform2	Type a value here
Move		

NOTE: The Move button reorders the first two Subform2 instances, and the Set button displays the maximum number of Subform2 instances. In both cases, you may need to add or remove subforms, or make changes to the data in the text fields to see the changes applied to the Subform2 instances.

15.14.1. Scripting to determine whether you added the maximum number of subforms to a form

The following script determines whether the supported maximum number of Subform2 instances exist on the form. If the maximum number exists, the script displays a message. Otherwise, a new Subform2 instance is added to the form.

```
if (methods.Subform2.instanceManager.count ==
methods.Subform2.instanceManager.max) {
  xfa.host.messageBox("You have reached the maximum number of items allowed.",
  "Instance Manager Methods", 1);
}
else {
  methods.Subform2.instanceManager.addInstance(1);
  xfa.form.recalculate(1);
}
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here:

```
if (methods._Subform2.count == methods._Subform1.max) {
  xfa.host.messageBox("You have reached the maximum number of items
allowed.", "Instance Manager Methods", 1);
}
else {
  methods._Subform2.addInstance(1);
  xfa.form.recalculate(1);
}
```

15.14.2. Scripting to determine whether there are more subforms to remove on the form

The following script determines whether any Subform2 instances exist on the form. If none exist, the script displays a message indicating that no instances exist. If instances do exist, the script removes the first instance from the form.

```
if (methods.Subform2.instanceManager.count == 0) {
    xfa.host.messageBox("There are no subform instances to remove.",
        "Instance Manager Methods", 1);
}
else {
    methods.Subform2.instanceManager.removeInstance(0);
    xfa.form.recalculate(1);
}
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here:

```
if (methods._Subform2.count == 0) {
    xfa.host.messageBox("There are no subform instances to remove.",
        "Instance Manager Methods", 1);
}
else {
    methods._Subform2.removeInstance(0);
    xfa.form.recalculate(1);
}
```

15.14.3. Scripting to force four subform instances to appear on the form

The following script forces four Subform2 instances to appear on the form regardless of how many instances currently exist:

```
methods.Subform2.instanceManager.setInstances(4);
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here:

```
methods._Subform2.setInstances(4);
```

15.14.4. Scripting to force the first and second subforms to switch locations on the form

The following script forces the first and second Subform2 instances to switch locations on the form.

```
methods.Subform2.instanceManager.moveInstance(0,1);
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here.

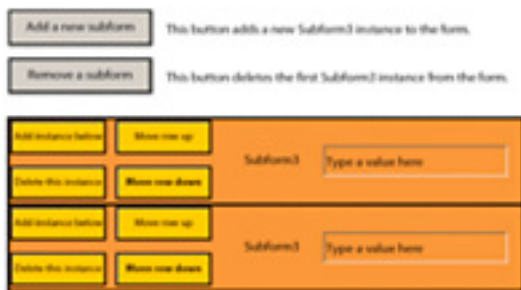
```
methods._Subform2.moveInstance(0,1);
```

15.15. Using the instance manager to control subforms at run time

This example demonstrates how to use properties and methods of the instance manager to retrieve information about subforms and perform operations on subform objects at run time.

In this example, the form filler uses the buttons to perform various actions using instances of Subform3. For example, when the form filler clicks the Add Row Below button a new Subform3 instance is added below the current instance.

NOTE: You may need to add or remove subforms, or make changes to the data in the text field, to see the changes applied to the instances of Subform3.



NOTE: If no instances of a particular subform exist on your form, you must use the underscore (_) notation provided with each example below. For more information about using the underscore (_) notation, see Designer Help.

15.15.1. Scripting the Add a New Subform button

The following script determines whether the supported maximum number of Subform3 instances exist on the form. If the maximum number exist, the script displays a message. Otherwise, a new Subform3 instance is added to the form.

```
if (advanced.Subform3.instanceManager.count ==
advanced.Subform3.instanceManager.max) {
xfa.host.messageBox("You have reached the maximum number of items
allowed.", "Combining Instance Manager Concepts", 1);
}
else {
advanced.Subform3.instanceManager.addInstance(1);
xfa.form.recalculate(1);
}
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here:

```
if (advanced._Subform3.count == advanced._Subform3.max) {
xfa.host.messageBox("You have reached the maximum number of items
allowed.", "Combining Instance Manager Concepts", 1);
}
```

```
}  
else {  
  advanced._Subform3.addInstance(1);  
  xfa.form.recalculate(1);  
}
```

15.15.2. Scripting the Remove a Subform button

The following script determines whether any Subform3 instances exist on the form. If none exist, the script displays a message indicating that no instances exist. If instances exist, the script removes the first instance from the form.

```
if (advanced.Subform3.instanceManager.count == 0) {  
  xfa.host.messageBox("There are no subform instances to remove.",  
    "Combining Instance Manager Concepts", 1);  
}  
else {  
  advanced.Subform3.instanceManager.removeInstance(0);  
}
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here:

```
if (advanced._Subform3.count == 0) {  
  xfa.host.messageBox("There are no subform instances to remove.",  
    "Combining Instance Manager Concepts", 1);  
}  
else {  
  advanced._Subform3.removeInstance(0);  
}
```

15.15.3. Scripting the Add Instance Below button

The following if-else statement prevents the script from proceeding if the form currently contains the maximum number of Subform3 instances:

```
if (Subform3.instanceManager.count < Subform3.instanceManager.occurences.max) {  
  //oNewInstance stores an instance of Subform3 created by the addInstance()  
  method.  
  var oNewInstance = Subform3.instanceManager.addInstance(1);  
  //nIndexFrom and nIndexTo store the before and after index values to use with the  
  moveInstance() method.  
  var nIndexFrom = oNewInstance.index;  
  var nIndexTo = Subform3.index + 1;
```

In this case, when the script references the value for nIndexFrom, the new instance of Subform3 is added to the form in the position specified in the moveInstance method:

```
Subform3.instanceManager.moveInstance(nIndexFrom, nIndexTo);  
}  
else {
```

```
xfa.host.messageBox("You have reached the maximum number of items  
allowed.", "Combining Instance Manager Concepts", 1);  
}
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here:

```
if (_Subform3.count < _Subform3.occurences.max) {  
    var oNewInstance = _Subform3.addInstance(1);  
    var nIndexFrom = oNewInstance.index;  
    var nIndexTo = Subform3.index + 1;  
    _Subform3.moveInstance(nIndexFrom, nIndexTo);  
}  
else {  
    xfa.host.messageBox("You have reached the maximum number of items allowed.",  
        "Combining Instance Manager Concepts", 1);  
}
```

15.15.4. Scripting the Delete This Instance button

The following if-else statement prevents the script from proceeding if the form currently contains the minimum number of Subform3 instances.

```
if (Subform3.instanceManager.count >  
    Subform3.instanceManager.occurences.min) {
```

This script uses the `removeInstance` method to remove an instance of Subform3.

NOTE: This script uses the value `parent.parent.index` to indicate the Subform3 instance to remove. The `parent` reference indicates the container of the object using the reference. In this case, using the reference `parent.index` would indicate the untitled subform that contains the Add Instance Below, Delete This Instance, Move Row Up, and Move Row Down buttons.

```
Subform3.instanceManager.removeInstance(parent.parent.index);  
}  
else {  
    xfa.host.messageBox("You have reached the minimum number of items  
allowed.", "Combining Instance Manager Concepts", 1);  
}
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here:

```
if (_Subform3.count > _Subform3.occurences.min) {  
    Subform3.removeInstance(Subform3.index);  
}  
else {  
    xfa.host.messageBox("You have reached the minimum number of items allowed.",  
        "Combining Instance Manager Concepts", 1);  
}
```


15.15.5. Scripting the Move Row Up button

The following if-else statement prevents the script from proceeding if the instance of Subform3 appears as the first instance in the list:

```
if (Subform3.index != 0) {  
  //nIndexFrom and nIndexTo store the before and after index values to use with the  
  moveInstance method.  
  var nIndexFrom = Subform3.index;  
  var nIndexTo = Subform3.index - 1;  
  Subform3.instanceManager.moveInstance(nIndexFrom, nIndexTo);  
}  
else {  
  xfa.host.messageBox("The current item cannot be moved because it is the  
  first instance in the list.", "Combining Instance Manager Concepts", 1);  
}
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here:

```
if (Subform3.index != 0) {  
  var nIndexFrom = Subform3.index;  
  var nIndexTo = Subform3.index - 1;  
  Subform3.moveInstance(nIndexFrom, nIndexTo);  
}  
else {  
  xfa.host.messageBox("The current item can't be moved since it already is the  
  first instance in the list.", "Combining Instance Manager Concepts", 1);  
}
```

15.15.6. Scripting the Move Row Down button

This variable stores the index value of the instance of Subform3:

```
var nIndex = Subform3.index;
```

The following if-else statement prevents the script from proceeding if the instance of Subform3 appears as the last instance in the list:

```
if ((nIndex + 1) < Subform3.instanceManager.count) {  
  // nIndexFrom and nIndexTo store the before and after index values to use with  
  the moveInstance() method.  
  var nIndexFrom = nIndex;  
  var nIndexTo = nIndex + 1;  
  
  Subform3.instanceManager.moveInstance(nIndexFrom, nIndexTo);  
}  
else {  
  xfa.host.messageBox("The current item cannot be moved because it is the last  
  instance in the list.", "Combining Instance Manager Concepts", 1);  
}
```

You can also write this script by using the underscore (_) notation to reference the properties and methods of the instance manager, as shown here:

```
var nIndex = Subform3.index;
if ((nIndex + 1) < Subform3.instanceManager.count) {
var nIndexFrom = nIndex;
var nIndexTo = nIndex + 1;
_Subform3.moveInstance(nIndexFrom, nIndexTo);
}
else {
xfa.host.messageBox("The current item can't be moved since it already is the
last instance in the list.", "Combining Instance Manager Concepts", 1);
}
```