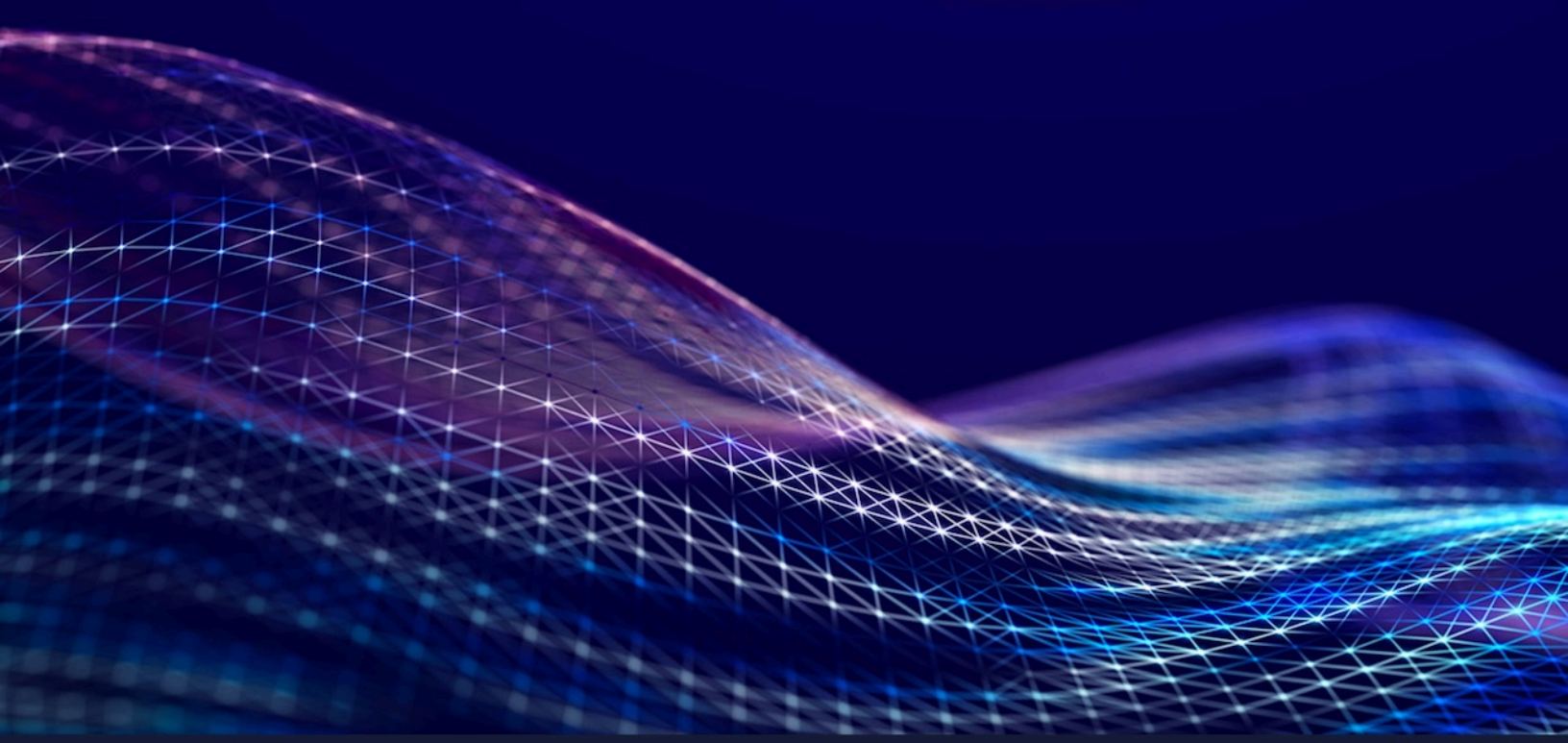


Tungsten Power PDF

JavaScript Reference Guide

2025.3

TUNGSTEN AUTOMATION

© 2025 Tungsten Automation. All rights reserved.

Tungsten and Tungsten Automation are trademarks of Tungsten Automation Corporation, registered in the U.S. and/or other countries. All other trademarks are the property of their respective owners. No part of this publication may be reproduced, stored, or transmitted in any form without the prior written permission of Tungsten Automation.

Table of Contents

Chapter 1: JavaScript API Guide....8

Introduction....8

Syntax and objects....9

Paths....10

Safe path....10

Privileged context....10

Dummy data in code samples....10

JavaScript API....10

Annot3D....11

Annotation....11

Annotation types....11

Annotation properties....13

Annotation methods....27

app....27

App properties....27

App methods....33

Bookmark....41

Bookmark properties....41

Bookmark methods....43

Certificate....45

Certificate properties....45

console....47

console methods....47

Data....48

Data properties....48

Doc....49

Doc Properties....49

Doc Methods....60

Event....88

Form processing order....92

Event properties....92

Field....98

Field properties....98

Field methods....116

FullScreen....127

FullScreen properties....127
global....128
global methods....128
RDN....129
RDN properties....129
search....130
search properties....130
search methods....132
security....134
security constants....134
security properties....134
security methods....134
SecurityHandler....135
SecurityHandler properties....135
SecurityHandler methods....138
SignatureInfo....140
SignatureInfo properties....140
util....145
util methods....145
Chapter 2: JavaScript 3D....150
JavaScript objects for 3D in Power PDF....150
Background....150
Bone....150
BoundingBox....150
Camera....150
CameraEvent....151
CameraEventHandler....152
Canvas....152
ClippingPlane....152
Color....152
Console....152
Dummy....153
HitInfo....153
Host....153
Image....153
KeyEvent....153
KeyEventHandler....154
Light....154

Material....155
Matrix4x4....155
MenuEvent....156
MenuEventHandler....156
Mesh....157
MouseEvent....158
MouseEventHandler....158
Node....159
Procedural....159
Quaternion....159
RenderEvent....160
RenderEventHandler....160
RenderOptions....160
Resource....161
Runtime....161
Scene....162
SceneObject....164
SceneObjectList....165
ScrollWheelEvent....165
ScrollWheelEventHandler....165
SelectionEvent....165
SelectionEventHandler....166
StateEvent....166
StateEventHandler....166
Texture....166
TimeEvent....167
TimeEventHandler....167
ToolEvent....167
ToolEventHandler....167
Vector3....168
View....168

Preface

This document describes the JavaScript objects, properties, methods, and constants currently supported by the Power PDF JavaScript API. Each topic has a short description and examples are provided, too. Descriptions focus on structures and connections specific to Power PDF, and are not intended to provide information on general JavaScript or PDF issues. This document aims to provide help for interactive PDF programmers and intelligent document designers.

Requirements

Software requirements:

- Power PDF 3.0 or later, either Standard, Advanced, or Business edition installed and running on your computer.

  Power PDF 2.x also supports JavaScript, but only with a subset of assets.

Personal requirements:

- The ideal reader is familiar with basic JavaScript 1.6 programming concepts such as variables, decisions, cycles, properties, methods, objects, and events. This guide is not a textbook for JavaScript programming but requires the reader to understand programming on a beginner level at least.
- Knowledge of the Power PDF user interface to build scenarios for examples. It is not required but useful to know the PDF file format structure.

Related documentation

This document refers to the following sources:

- *PDF Reference version 1.7*
- *XFA Specification, version 2.2*

Other useful sources:

- Power PDF Online Help — Refer to the online help for details on features and concepts in Power PDF.
- JavaScript core tutorials and reference — Refer to the core documentation, reference, or tutorials to learn more about the JavaScript language, programming concepts, and assets.

Getting help with Tungsten Automation products

The Tungsten Automation Knowledge Portal repository contains articles that are updated on a regular basis to keep you informed about Tungsten Automation products. We encourage you to use the Knowledge Portal to obtain answers to your product questions.

To access the Tungsten Automation Knowledge Portal, go to:

https://knowledge.tungstenautomation.com/

The Knowledge Portal is optimized for use with Google Chrome, Mozilla Firefox, or Microsoft Edge.

The Knowledge Portal provides:

- Powerful search capabilities to help you quickly locate the information you need.
  Type your search terms or phrase into the **Search** box, and then select the search icon.
- Product information, configuration details and documentation, including release news.
  To locate articles, go to the Knowledge Portal home page and select the applicable Solution Family for your product, or select the View All Products button.

From the Knowledge Portal home page, you can:

- Access the Tungsten Automation Community (for all customers).
  On the Tungsten Automation Resources menu, select the **Community** link.
- Access the Tungsten Automation Customer Portal (for eligible customers).
  Go to the Support Portal Information page and select **Log in to the Tungsten Automation Customer Portal**.
- Access the Tungsten Automation Partner Portal (for eligible partners).
  Go to the Support Portal Information page and select **Log in to the Tungsten Automation Partner Portal**.
- Access support commitments, lifecycle policies, electronic fulfillment details, and self-service tools.
  Go to the Support Details page and select the appropriate article.

Chapter 1

JavaScript API Guide

Introduction

Power PDF has a built-in JavaScript engine providing scripting capabilities for designers of interactive PDF documents, for plug-in developers or form designers. The engine covers most of the functionality of the software, making possible to script:

- Form processing
- Batch processing
- Data import and export

Power PDF provides several ways to run or call JavaScript.

- JavaScript Console
- Document JavaScript
- PDF document events

For details on working with JavaScript, see the "Use JavaScript" topic in *Power PDF Help*.

JavaScript libraries added to the application's JavaScript folder share their functions for any script running in Power PDF. Follow the instructions to install the libraries:

1. Prepare your JavaScript library (.js) files. A single file may contain multiple functions.

   - No limitations apply on the number of files to add.
   - The size of the .js files is limited to 4 MB.

2. Copy the .js files to the following folder:
   `%AppData%\Kofax\PDF\PowerPDF\JavaScript`

   The above folder contains a file named global.js. Do not edit or delete this file; use your custom .js files instead.

3. Restart Power PDF.

To test your JavaScript function, do the following:

1. In Power PDF, start the JavaScript console at **Edit** > **JavaScript** > **JavaScript Console** .

2. Type the name of the function with the required actual parameters, then press Ctrl + Enter. Example:

```
MyFunction(MyParameter);
```

3. Verify the output of your function.

Syntax and objects

Power PDF follows the core JavaScript syntax and uses an object hierarchy following application and document features and structures.

All identifiers are case sensitive in JavaScript.

Static objects

Power PDF JavaScript has the built in objects (such as `app` or `Doc`) to reflect Power PDF features and PDF document structure. Most of the objects are dynamic, but the following objects are static and managed by the application:

- app
- event
- global
- search
- security
- util

Other objects are dynamic, so you can assign them to a variable. This script assigns a newly created document to the oDOC variable:

```
var oDOC = app.newDoc();
```

Now all properties and methods are accessible with this variable. This script displays the Print dialog:

```
oDOC.print();
```

Arguments

Methods may receive their arguments two ways:

- Standard argument list — Separate neighboring arguments by a comma, and provide them in the order specified in the method definition.

```
app.alert( "Power PDF 3.0", 3);
```

- Single object argument with corresponding property names and values — Surround the argument object by curly brackets. Specify an argument by its parameter name, followed by a colon and a value. Separate argument specifications by a comma.

```
app.alert({ cMsg: "Power PDF 3.0", nIcon: 3});
```

Paths

You should provide *device-independent path*s in some method arguments (refer to *PDF Reference version 1.7* for details on the format).

Safe path

Safe path means the following:

- Paths passed to JavaScript methods and properties writing to files may not point to a system or critical folder (such as the Windows folder or the root folder).
- The specified file name extension should correspond to the data type to write.
- Some methods are blocked to overwrite any file.
- A path or a file name considered not safe raises a `NotAllowedError` exception and the method fails.

Privileged context

Some methods can only run in privileged context. Privileged context provides an elevated state, where special operations (which are restricted in other contexts) may run. The following may grant privileged context:

- Running in the JavaScript Console.
- Running as a Document Action (Document JavaScript); see getPath for example.

Dummy data in code samples

This document uses various dummy data in the code samples. Replace the following samples with production data:

- Domain names (such as `example.com`)
- E-mail addresses (such as `john.doe@example.com`)
- Paths (such as `/C/IDs/JohnDoe.pfx`)

JavaScript API

The following chapters describe the supported JavaScript objects in alphabetical order.

Annot3D

Description	Represents an annotation created using the 3D tool.
Properties	• activated — The `true` value displays the 3D artwork; `false` only the posterboard. • context3D • innerRect — Four numbers specifying the lower-left x, lower-left y, upper-right x, and upper-right y coordinates of the annotation box containing the 3D artwork. • name • page • rect
Functions	• `getAnnot3D(nPage, cName)` • `getAnnots3D(nPage)`

Annotation

This object describes a Power PDF annotation. Annotations are designed with the annotation tool, and they have a randomized name (such as `5e944574-926d-c926-6eb7-e5bf3e230653`). Since the user cannot gather these names using the UI, currently the best way of accessing annotations is calling the `Doc.getAnnots` method, which returns with an array of `Annotation` objects. All objects in this array have the annotation name in their `name` property, so annotations can be accessed and renamed, as it is demonstrated in the example provided for the `name` property (see name for details).

To access an annotation by name with JavaScript first it should be bound to a variable using the `Doc.getAnnot`.

```
var oAnnot = this.getAnnot(0, "ReviewNotes");
```

As this script assigned the variable, you can then reach the annotation named “ReviewNotes” on the first page (page numbering starts with 0) by means of the variable `oAnnot`.

The following example shows how to read and write object properties. The first script line stores the type of the annotation in the variable `mytype`, then the second line of code changes the author to `John Doe`.

```
var mytype = oAnnot.type; // reads property
oAnnot.author = "John Doe"; // writes property
```

Annotation types

There are different kind of Annotations, the type can be determined by checking the `type` property. The table below lists annotation types, providing all documented properties.

Annotation type	Properties
Caret	author, borderEffectIntensity, borderEffectStyle, caretSymbol, contents, creationDate, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, width
Circle	author, borderEffectIntensity, borderEffectStyle, contents, creationDate, dash, fillColor, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, width
FileAttachment	attachIcon, author, borderEffectIntensity, borderEffectStyle, contents, creationDate, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, point, print, readOnly, rect, refType, rotate, strokeColor, style, subject, oggleNoView, type, width
FreeText	alignment, author, borderEffectIntensity, borderEffectStyle, callout, contents, creationDate, dash, fillColor, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, print, readOnly, rect, refType, rotate, strokeColor, style, subject, textFont, toggleNoView, type, width
Highlight	author, borderEffectIntensity, borderEffectStyle, contents, creationDate, dash, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, width
Ink	author, borderEffectIntensity, borderEffectStyle, contents, creationDate, gestures, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, width
Line	arrowBegin, arrowEnd, author, borderEffectIntensity,borderEffectStyle, contents, creationDate, dash, doCaption, fillColor, hidden, inReplyTo, intent, leaderExtend, leaderLength, lineEnding, lock, modDate, name, noView, opacity, page, points, popupOpen, popupRect, print, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, width
Polygon	author, borderEffectIntensity, borderEffectStyle, contents,creationDate, dash, fillColor, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, vertices, width
PolyLine	arrowBegin, arrowEnd, author, borderEffectIntensity,borderEffectStyle, contents, creationDate, dash, fillColor, hidden, inReplyTo, intent, lock, modDate, name, noView,opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, vertices, width
Sound	author, borderEffectIntensity, borderEffectStyle, contents,creationDate, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, point, print, readOnly, rect, refType, rotate, soundIcon, strokeColor, style, subject, toggleNoView, type, width
Square	author, borderEffectIntensity, borderEffectStyle, contents,creationDate, dash, fillColor, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, width
Squiggly	author, borderEffectIntensity, borderEffectStyle, contents,creationDate, hidden, inReplyTo, intent, lock, modDate,name, noView, opacity, page, popupOpen, popupRect, print, quads, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, width

Annotation type	Properties
Stamp	AP, author, borderEffectIntensity, borderEffectStyle,contents, creationDate, hidden, inReplyTo, intent, lock,modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type
StrikeOut	author, borderEffectIntensity, borderEffectStyle, contents,creationDate, hidden, inReplyTo, intent, lock, modDate,name, noView, opacity, page, popupOpen, popupRect, print, quads, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, width
Text	author, borderEffectIntensity, borderEffectStyle, contents,creationDate, hidden, inReplyTo, intent, lock, modDate, name, noView, noteIcon, opacity, page, point, popupOpen,popupRect, print, readOnly, rect, refType, rotate,strokeColor, style, subject, toggleNoView, type, width
Underline	author, borderEffectIntensity, borderEffectStyle, contents,creationDate, hidden, inReplyTo, intent, lock, modDate,name, noView, opacity, page, popupOpen, popupRect, print, quads, readOnly, rect, refType, rotate, strokeColor, style, subject, toggleNoView, type, width

Annotation properties

Some properties are stored as names, others are represented by strings in the PDF document. A name property can have 127 characters at most. For further details, refer to *PDF Reference version 1.7*.

alignment

Description	Determines the alignment of the text for a `FreeText` annotation.
Values	`0` — Left aligned `1` — Centered `2` — Right aligned
Type	Number
Access	Read/Write
Annotations	FreeText

AP

Description	The name of the standard stamp fashion to use when the stamp annotation is displayed. In Power PDF, changing the `AP` property does not change the graphical design but alters only the Subject line on the Author and Subject tab of the Stamp Properties dialog box. To find a particular name stamp search the `Stamps` folder.

Values	`Approved`
	`AsIs`
	`Confidential`
	`Departmental`
	`Draft`
	`Experimental`
	`Expired`
	`Final`
	`ForComment`
	`ForPublicRelease`
	`NotApproved`
	`NotForPublicRelease`
	`Sold`
	`TopSecret`

Type	String
Access	Read/Write
Annotations	Stamp
Example	Add a Final stamp to the first page of the document, then run this script to change its `AP` property to `Expired`. `var aAnnots = this.getAnnots({ nPage:0 }); var oAN = this.getAnnot(0, aAnnots[0].name); oAN.AP = "Expired";`

Related concepts

icons

arrowBegin

Description	Specifies the line cap style that describes the shape to be used at the start of the line annotation.

Values	`None` (default)
	`OpenArrow`
	`ClosedArrow`
	`ROpenArrow`
	`RClosedArrow`
	`Butt`
	`Diamond`
	`Circle`
	`Square`
	`Slash`

Type	String
Access	Read/Write

Annotations	Line, PolyLine

arrowEnd

Description	Specifies the line cap style that describes the shape to be used at the end of the line annotation.

Values	`None` (default)
	`OpenArrow`
	`ClosedArrow`
	`ROpenArrow`
	`RClosedArrow`
	`Butt`
	`Diamond`
	`Circle`
	`Square`
	`Slash`

Type	String
Access	Read/Write
Annotations	Line, PolyLine

attachIcon

Description	The label of an icon that is used when the annotation is displayed.

Values	`Paperclip` (default)
	`Insert`
	`Graph`
	`Tag`

Type	String
Access	Read/Write
Annotations	FileAttachment

author

Description	Represents the author of the annotation.
Type	String
Access	Read/Write
Annotations	All

borderEffectIntensity

Description	Represents the intensity of the border effect.

Type	Number
Access	Read/Write
Annotations	All

borderEffectStyle

Description	Currently, there are only two border values supported; empty string and `C` for cloudy.
Values	"" (empty string) `C`
Type	String
Access	Read/Write
Annotations	All

callout

Description	An array of four or six numbers that denotes a callout line connected to the free text annotation.
Type	Array
Access	Read/Write
Annotations	FreeText

caretSymbol

Description	The icon related to the Caret annotation, which shows the presence of text edits.
Values	"" (empty string) `P` (paragraph symbol) `S` (space symbol)
Type	String
Access	Read/Write
Annotations	Caret

contents

Description	Provides the content of an annotation that has a connected pop-up window. For file attachment and sound annotations, it defines the text that is displayed as a description.
Type	String
Access	Read/Write
Annotations	All

Example	Add a Note annotation to the first page of the document, then run this script to fill its `contents` property. `var aAnnots = this.getAnnots({ nPage:0 }); var oAN = this.getAnnot(0, aAnnots[0].name); oAN.contents = "This note was filled by JavaScript.";`

creationDate

Description	Creation date and time of the annotation.
Type	Date
Access	Read
Annotations	All

dash

Description	This array specifies the sequence of dashes and gaps in a dashed border.
Type	Array
Access	Read/Write
Annotations	FreeText,Line, PolyLine, Polygon, Circle, Square

Related concepts
style

doc

Description	Specifies the `Doc` object of the document where the annotation is placed.
Type	Doc object
Access	Read
Annotations	All

doCaption

Description	When the property holds true the rich contents are drawn in the line appearance
Type	Boolean
Access	Read/Write
Annotations	Line

fillColor

Description	Defines the background color for fillable annotations.

Values	Values are set using transparent, gray, RGB or CMYK colors.
Type	Color
Access	Read/Write
Annotations	Circle, Square, Line, Polygon, PolyLine, FreeText

gestures

Description	An array of arrays, each describing a stroked path by a series of x and y coordinate offsets within a default user space. The points along the path are linked by straight lines or curves, dependent on the implementation.
Type	Array
Access	Read/Write
Annotations	Ink

hidden

Description	When this property is `true`, the annotation is not displayed, not printable, and the user cannot interact with it.
Type	Boolean
Access	Read/Write
Annotations	all

inReplyTo

Description	When the property value is non-empty, then this annotation defines the `name` value of the annotation that this annotation is in reply to.
Type	String
Access	Read/Write
Annotations	All

Related concepts

inReplyTo

intent

Description	With this property, the markup annotation type acts differently, which depends on the planned use of the annotation. This property is specified for all the annotations but presently have values of `intent` only for free text, pplygon, and line annotations.
Type	String
Access	Read/Write

Annotations	All Refer to the Annotations with special appearances table that lists the tools that are used for creating annotations with special appearances, which are available through UI.

Annotations with special appearances

UI	Annotation type	Intent
Callout Tool	`FreeText`	`FreeTtextCallout`
Cloud Tool	`Polygon`	`PolygonCloud`
Arrow Tool	`Line`	`LineArrow`
Dimensioning Tool	`Line`	`LineDimension`

leaderExtend

Description	Defines the leader line extensions length that originates from both endpoints of the line, and is perpendicular to it. These line extensions are exactly 180 degrees from the leader lines.
Values	>= `0` (`0` is the default, when there is no leader line extension)
Type	Number
Access	Read/Write
Annotations	Line

leaderLength

Description	Defines the length of the leader lines that originates from both endpoints of the line, which is perpendicular to the line. The negative length value indicates an alternate orientation of the leader lines.
Values	(0 is the default, when there is no leader line)
Type	Number
Access	Read/Write
Annotations	Line

lineEnding

Description	Defines the way the end of a callout line is stroked, and is applicable only to free text annotation when the value of `intent` is `FreeTextCallout`.

Values	`None` (default)
	`OpenArrow`
	`ClosedArrow`
	`ROpenArrow`
	`RClosedArrow`
	`Butt`
	`Diamond`
	`Circle`
	`Square`
	`Slash`

Type	String
Access	Read/Write
Annotations	Line, FreeTextCallout

lock

Description	When the property holds true, the annotation is locked, that is similar to being read-only. Locked annotations are only accessible through the properties dialog box in the UI.
Type	Boolean
Access	Read/Write
Annotations	All

Related concepts

readOnly

modDate

Description	States the date when the annotation was last modified.
Type	Date
Access	Read/Write
Annotations	All

Example	Add a Note annotation to the first page of the document, then run this script to display its last modification date on the console.
	`var aAnnots = this.getAnnots({`
	`  nPage:0`
	`});`
	`var oAN = this.getAnnot(0, aAnnots[0].name);`
	`console.println(util.printd("mmmm dd, yyyy", oAN.modDate));`

name

Description	Specifies an annotation name and the value of which is used by the `Doc.getAnnot` method to search and access the methods and properties of the annotation.

Type	String
Access	Read/Write
Annotations	All
Example	This code uses `getAnnots` to retrieve annotation object names, then grabs the very first annotation on page 0 by `getAnnot` and renames it. `var aAnnots = this.getAnnots({ nPage:0, nSortBy: ANSB_Author, bReverse: false }); var oAN = this.getAnnot(0, aAnnots[0].name); if (aAnnots.length > 0) oAN.name = "FirstOnPage0"; console.println("The name of the very first annotation is: " + oAN.name);`

noteIcon

Description	Specifies the icon label which is used to display the annotation.

Values	`Note`
	`Key`
	`Tag`
	`Insert`
	`Speaker`
	`Graph`
	`Paragraph`
	`Paperclip`
	`New Paragraph`
	`Help`

Type	String
Access	Read/Write
Annotations	Text

noView

Description	When the property holds `true` the annotation is hidden, and appears only when used for printing.
Values	None
Type	Boolean
Access	Read/Write
Annotations	All

opacity

Description	Specifies a constant opacity value, which is used for painting the annotation. This value is applicable to all visible parts of the annotation, excluding the pop-up window, which is displayed when the annotation opens.
Values	`0.0-1.0` (the default is `0.5`)
Type	Number
Access	Read/Write
Annotations	All

page

Description	Specifies the page where the annotation is placed.
Values	None
Type	Integer
Access	Read/Write
Annotations	All

point

Description	An array with two numbers [Xul, Yul] defining the upper left-hand corner in default user space of the icon meant for a sound, text, or a file attachment annotation.
Values	None
Type	Array
Access	Read/Write
Annotations	Text, Sound, FileAttachment

Related concepts

noteIcon

points

Description	An array of two point [[X1, Y1], [X2, Y2]] arrays defining the starting and the ending coordinates of the line in the default user space.
Values	None
Type	Array
Access	Read/Write
Annotations	Line

Related concepts
arrowBegin
arrowEnd

popupOpen

Description	If the property holds `true`, the pop-up text appears to be open when the page is displayed.
Values	None
Type	Boolean
Access	Read/Write
Annotations	All except FreeText, Sound, and FileAttachment

popupRect

Description	A four number [xll, yll, xur, yur] array that defines the lower-left x, lower-left y, upper-left x, and upper-right y coordinates of the rectangle of the pop-up annotation window, which is related to a parent annotation. The coordinates should be interpreted in the default user space.
Values	None
Type	Array
Access	Read/Write
Annotations	All except FreeText, Sound, FileAttachment

print

Description	If this property is set `true`, then the annotation appears on the print output, otherwise not.
Values	None
Type	Boolean
Access	Read/Write
Annotations	All

quads

Description	An array of quad lists defining the coordinates of quadrilaterals in the default user space. A quad list itself is also an array of four coordinate points (4×2 numbers altogether) defining an area comprises one or more word of contiguous text underlying the annotation.
Values	None
Type	Array
Access	Read/Write
Annotations	Highlight, StrikeOut, Underline, Squiggly

rect

Description	The `rect` array contains four numbers [xll, yll, xur, yur] to define the lower-left x, lower-left y, upper-left x, and upper-right y coordinates in the default user space. This way `rect` represents the rectangle that specifies the area of the annotation on the page.
Type	Array
Access	Read/Write
Annotations	All

Related concepts
popupRect

readOnly

Description	When this property holds `true` the annotation is displayed but do not interact with the user.
Values	None
Type	Boolean
Access	Read/Write
Annotations	All

refType

Description	Specifies the reference type of an annotation, it differentiates the relationship that `inReplyTo` denotes either as plain threaded discussion relationship or a group relationship.
Values	`R` `Group`
Type	String
Access	Read/Write
Annotations	All

Related concepts
inReplyTo

rotate

Description	Defines the number of degrees the annotation is rotated counter-clockwise with respect to the page. Only applicable to free text annotations.
Values	`0`, `90`, `180`, `270` degrees
Type	Integer

Access	Read/Write
Annotations	FreeText

soundIcon

Description	Defines the icon for the sound annotation.

Values	`Sound`
	`Mic`
	`Ear`

Type	String
Access	Read/Write
Annotations	Sound

strokeColor

Description	Specifies the display color of the annotation. For free text annotation, `strokeColor` specifies the border and the text colors.
Values	Values are defined by using `transparent`, `gray`, `RGB` or `CMYK` color objects.
Type	Array
Access	Read/Write
Annotations	All

style

Description	Specifies the border style. This property is defined for all the annotation types but is applicable only to the line, free text, circle, square, polyline, and polygon annotations.
Values	`solid` (Solid style, available for Circle, FreeText, Line, Polygon, Polyline, and Square annotations) `dash` (Dashed style, available for Circle, FreeText, Line, Polygon, Polyline, and Square annotations) `cloudy` (Cloudy style, available for Circle, Polygon, and Square annotations)
Type	String
Access	Read/Write
Annotations	Circle, FreeText, Ink, Line, Polygon, Polyline, and Square

Related concepts

dash

subject

Description	The text used for the short description of the subject of the annotation. This text is displayed also in the title bar of the pop-up window or the properties dialog box.

Values	None
Type	String
Access	Read/Write
Annotations	All

toggleNoView

Description	If this property holds `true`, then the `noView` flag is toggled when the annotation is selected, or mouse hovers over the annotation. Setting both `noView` and `toggleNoView` to `true` turns the annotation object practically invisible.
Values	None
Type	Boolean
Access	Read/Write
Annotations	All

Related concepts

noView
toggleNoView

type

Description	Specifies the annotation type. Annotation type cannot be modified by setting this property.

Values	`Text`
	`FreeText`
	`Line`
	`Square`
	`Circle`
	`Polygon`
	`PolyLine`
	`Highlight`
	`Underline`
	`Squiggly`
	`StrikeOut`
	`Stamp`
	`Caret`
	`Ink`
	`FileAttachment`
	`Sound`

Type	String
Access	Read
Annotations	All

vertices

Description	An array of coordinate arrays specifying the coordinates (horizontal and vertical respectively) of each vertex consisting a polygon or polyline annotation in the default user space.
Type	Array of arrays
Access	Read/Write
Annotations	Polygon, PolyLine

width

Description	Specifies the border width in points. Zero results in no border.
Values	`0-12` (`1` is the default)
Type	Number
Access	Read/Write
Object Type	Square, Circle, Line, Ink, FreeText

Annotation methods

destroy

Description	Destroys the annotation, eliminating it from the page. As a result the object becomes invalid.
Annotations	All
Example	The following script deletes the very first annotation (if there is any) on the first page. `var aAnnots = this.getAnnots({ nPage:0 }); var oAN = this.getAnnot(0, aAnnots[0].name); oAN.destroy();`

app

This static object represents the Power PDF application instance currently running. Provides tools related to timing, view, printing, paths, and others.

App properties

activeDocs

Description	An array containing a `Doc` object for each document currently opened, including the hidden ones. If there are no active documents, then this is an empty array.
Type	Array
Access	Read

Example	This code lists the title of each opened documents to the console.
	`var aDocs = app.activeDocs;`
	`for (var i=0; i < aDocs.length; i++)`
	`  console.println(aDocs[i].title);`

calculate

Description	Set this property to `false` to stop the automatic calculation of the fields. The default is `true`. This property is deprecated, use `Doc.calculate` instead.
Type	Boolean
Access	Read/Write
Object Type	None

Related concepts
Doc.calculate

focusRect

Description	Set this property to `false` to turn off the dotted rectangle drawn around the UI control (check box, button, radio button or signature) which currently has the keyboard focus.
Type	Boolean
Access	Read/Write
Example	This script turns off the focus rectangle. `app.focusRect = false;`

formsVersion

Description	This property holds the version number of the form viewer software module. `formsVersion` may return a different version number than `viewerVersion`.
Type	Number
Access	Read

Example	This script queries both the viewer and form software version numbers.
	`console.println("Version of the form software: " + app.formsVersion);`
	`console.println("Version of the viewer software: " +`
	`app.viewerVersion);`
	`console.println("Variation of the viewer application: " +`
	`app.viewerVariation);`
	`console.println("Type of the viewer: " + app.viewerType);`

Related concepts
viewerVersion
viewerType
viewerVariation

fromPDFConverters

Description	This array of strings contains all file type conversion IDs.
Type	Array
Access	Read

Example	This script list all type-conversion IDs to the console.
	`for ( var i = 0; i < app.fromPDFConverters.length; i++)`
	`console.println("#" + i + ": " + app.fromPDFConverters[i]);`

fs

Description	A `FullScreen` object, which grants access to presentation mode properties.
Type	Object
Access	Read

Related concepts
FullScreen object
FullScreen.isFullScreen

fullscreen

Description	Determines if the application is in normal or foll screen (presentation) mode. This property is deprecated, please use `isFullScreen` instead.
Type	Boolean
Access	Read/Write

Related concepts
FullScreen object
FullScreen.isFullScreen

language

Description	The application language code.
Values	• `CHS` — Chinese Simplified • `CHT` — Chinese Traditional • `DAN` — Danish • `DEU` — German • `ENU` — English • `ESP` — Spanish • `FRA` — French • `ITA` — Italian • `KOR` — Korean • `NLD` — Dutch • `NOR` — Norwegian • `PTB` — Brazilian Portugese • `SUO` — Finnish • `SVE` — Swedish
Type	String
Access	Read
Example	This code prints the current language code to the console. `console.println("Current language code is: " + app.language);`

numPlugIns

Description	This property holds the number of the currently loaded plug-ins in Power PDF. This property is deprecated, use `app.plugins.length` instead.
Type	Number
Access	Read

Related concepts
plugIns

openInPlace

Description	Set this property to `true` to open cross-document links in the current window/tab. Set this property to `false` to open cross-document links in a new window/tab.
Type	Boolean
Access	Read/Write

Example	Run this code to open cross-document links in a new window or in a new tab. `app.openInPlace = false;`

platform

Description	This property indicates the operation system in use.
Values	`WIN` `MAC`
Type	String
Access	Read
Example	This code prints the current OS code to the console. `console.println("The code of the current OS is: " + app.platform);`

plugIns

Description	This array contains one plug-in object for each plug-in currently loaded.
Type	Array
Access	Read

Example	This code lists all plug-ins to the console.
	`var aPL = app.plugIns;`
	`var nPl = aPL.length;`
	`console.println("there are " + nPl + " plug-in(s) loaded:");`
	`for ( var i = 0; i < nPl; i++)`
	`  console.println("\#"+i+" - " + aPL[i].name);`

printerNames

Description	This array lists the available printers, returning an empty array if there are no printers installed on the system. Use any of these items in the `printerName` property of the `PrintParams` object.
Type	Array of Strings
Access	Read

Example	This script prints the list of printers to the console.
	`var aPn = app.printerNames;`
	`var nPn = aPn.length;`
	`console.println("There are " + nPn +" printer(s) available:");`
	`for ( var i = 0; i < nPn; i++)`
	`  console.println("#" + i + " - " + app.printerNames[i]);`

Related concepts
getPrintParams

viewerType

Description	This property reports which viewer application is running. Valid values are: • `Reader` • `Exchange` • `Exchange-Pro`
Type	String
Access	Read
Example	See the example provided for the `formsVersion` method.

Related concepts
formsVersion
viewerVariation
viewerVersion

viewerVariation

Description	This property holds the variation of the viewer application.

Values	`Reader`
	`Fill-in`
	`Business Tools`
	`Full`

Type	String
Access	Read
Example	See the example provided for the `formsVersion` method.

Related concepts
formsVersion
viewerType
viewerVersion

viewerVersion

Description	This property holds the version number of the viewer application. `formsVersion` may return a different version number than `viewerVersion`.
Type	Number
Access	Read
Example	See the example provided for the `formsVersion` method.

Related concepts
formsVersion
viewerType
viewerVariation

App methods

alert

Description	Launches a message dialog box.
Parameters	`cMsg` — Provide a string value to display as the message. `nIcon` — (optional) The type of the icon: • `0` — Error (white X on a red disk, default) • `1` — Warning (exclamation mark on a yellow traffic sign) • `2` — Question (question mark on a blue disk) • `3` — Status (blue information sign) `nType` — (optional) This code determines what button group to display: • `0` — A single OK button (default) • `1` — OK, Cancel • `2` — Yes, No • `3` — Yes, No, Cancel `cTitle` — (optional) Specify the text displayed in the title here. The default is Power PDF Standard or Power PDF Advanced. `oDoc` — (optional) The `Doc` object to associate the message with.
Return value	The return value depends on the type of button `nButton` pressed by the user: • `1` — OK • `2` — Cancel • `3` — No • `4` — Yes

Example	This code displays a message box with a question and offers `Yes`, `No`, or `Cancel`.
	`app.alert({`
	`  cMsg: "Is Power PDF the best PDF editor?",`
	`  nIcon: 2,`
	`  nType: 3,`
	`  cTitle: "Power PDF is the best!"`
	`});`

beep

Description	Use this method to play a sound.

Parameters	`nType` — Specify the type of the sound with one of the sound codes: • `0` — Error • `1` — Warning • `2` — Question • `3` — Status • `4` — Default
Example	Check the example provided with the `Bookmark.setAction` method.

Related concepts
Bookmark.setAction

clearInterval

Description	Clears an interval, which earlier was created using the `setInterval` method.
Parameters	`oInterval` — The interval object to cancel.
Example	Check the example provided with the `Field.buttonAlignY` method.

Related concepts
clearTimeOut
setTimeOut
setInterval
Field.buttonAlignY

clearTimeOut

Description	Clears a time-out interval, which earlier was created using the `setTimeOut` method.
Parameters	`oTime` — The time-out interval object to cancel.
Example	Check the example provided with the `Field.buttonAlignY` method.

Related concepts
clearTimeOut
clearInterval
setInterval
setTimeOut
Field.buttonAlignY

execMenuItem

Description	Executes the specified menu item. Not all menu items are allowed to execute, only the ones considered safe.

Parameters	`cMenuItem` — The menu item to launch. For menu names, refer to the Names of actions/menus chapter in the Appendix of *Power PDF Automation and Command Line Developer's Guide*. `oDoc` — (optional) The `Doc` object to associate the message with.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which acts the same way as you click on the File > Print menu. `app.execMenuItem("Print");`

getNthPlugInName

Description	Returns with the name of the *n*th plug-in loaded. This method is deprecated, use the `app.plugins` property instead.
Parameters	`nIndex` — Specify the index number of the plug-in to point.
Return value	`cName` holds the plug-in name
Example	This script prints the name of the plug-in loaded first. `console.println("The first plug-in is: " + app.getNthPlugInName(0));`

Related concepts

plugIns

getPath

Description	This method returns with the installation path for the selected component of Power PDF. Application folders and user folders are distinguished.

Parameters	• `cCategory` — (optional) Choose from `app` (global folder) or `user` (user-specific folder).
	• `cFolder` — (optional) Identify the component with one of the valid string values:
	  • `root`
	  • `eBooks`
	  • `preferences`
	  • `sequences`
	  • `documents`
	  • `javascript`
	  • `stamps`
	  • `dictionaries`
	  • `plugIns`
	  • `spPlugIns`
	  • `help`
	  • `temp`
	  • `messages`
	  • `resource`
	  • `update`

Example	Add this code to the `Document Save Requested` item in the `Document Actions` dialog box, this way the `getPath` method runs in a privileged context, as required. The script runs when next time you save the document and displays user-level and application-level JavaScript folders on the console.
	`console.println("JavaScript user folder: " +`
	` app.getPath("user","javascript"));`
	`console.println("JavaScript global folder: " +`
	` app.getPath("app","javascript"));`

goBack

Description	Switches back to the previous view. An error message box pops up if the view history is empty.
Example	This code steps back in the view history. `app.goBack();`

Related concepts

goForward

goForward

Description	Steps forward to the next view.

Example	This code steps forward in the view history. `app.goForward();`

Related concepts
goBack

launchURL

Description	Launches the default browser and loads the specified URL. Asks for confirmation before launch.
Parameters	`cURL` — Specify the URL in this string. `bNewFrame` — (optional) Set this to `true` if you want to open a new browser window or tab for the content. Set this false to replace the window or tab currently opened.
Return value	Returns with undefined, or throws an exception on error.
Example	This script launches www.example.com replacing the content of the current browser window or tab. `app.launchURL("http://www.example.com/", false);`

mailMsg

Description	Sends a custom mail message.
Parameters	`bUI` — (optional) If `true`, then the compose new mail window shows up, the message fields are populated using the rest of the parameters and the user may edit all the fields. If `false`, then Power PDF fills the message fields based on the parameters below. In this case only the `cTo` parameter is required. `cTo` — (optional) List of the recipients, separated by semicolon. `cCc` — (optional) List of the CC recipients, separated by semicolon. `cBcc` — (optional) List of the BCC recipients, separated by semicolon. `cSubject` — (optional) The subject line, limited to 64 KB. `cMsg` — (optional) The message body, limited to 64 KB.
Return value	None
Object types	None
Example	This code sends a short email message. Even if `bUI` is `false`, the message window may still show up if the script is not running in a privileged context. `app.mailMsg(false, "name1@example.com; name2@example.com", "", "", "The subject", "The body of the mail could be also composed using form field data.");`

Related concepts
mailDoc
mailForm

newDoc

Description	This method creates a new document. Needs to run in a privileged context.
Parameters	`nWidth` — (optional) The page width of the new document in points. (The default is `612`.) `nHeight` — (optional) The page height of the new document in points. (The default is `792`.)
Return value	The `Doc` object representing the new document.
Example	Add this code to the `Document Close Requested` item in the `Document Actions` dialog box, this way the `newDoc` method runs in a privileged context, as required. `app.newDoc();`

Related concepts
Event

popUpMenu

Description	Shows a custom pop-up menu at the current mouse position. This method is deprecated, please use `popUpMenuEx` instead.
Parameters	`cItem` — (optional) You may specify multiple arguments, either strings or arrays. • If the argument is a string, then it is added to the menu as a new item. Specify a *dash* (-) to insert a separator line. • If the argument is an array, then shows up as a submenu, using the first item as a parent menu entry.
Return value	Returns with a string containing the name of the selected menu item. Returns `null` if there was no item selected.

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to pop up a structured menu. Select any item to fill the `Text1` field.
	`var cCH = app.popUpMenu("Select your text", "-", "Free", [ "Pay per week", "Pay per month", "Pay per year"]);`
	`var oTB = this.getField("Text1");`
	`oTB.value = cCH;`

Related concepts
popUpMenuEx

popUpMenuEx

Description	Shows a custom pop-up menu at the current mouse position.

Parameters	This method requires one or more `MenuItem` objects as argument. The `MenuItem` object has the following properties: • `cName` — The name of the menu item. This string appears in the menu. Specify a *dash* (-) to insert a separator line. • `bMarked` — (optional) Set this property to `true` to mark this item with a check. The default value is `false` (not marked). • `bEnabled` — (optional) Set this property to `false` to disable (gray out) the item. The default value is `true` (enabled). • `cReturn` — (optional) `popUpMenuEx` returns with this string if the menu item was selected. Equals to `cName` by default. • `oSubMenu` — (optional) Either a single `MenuItem` object or an array of `MenuItem` objects to insert here as a submenu item or submenu structure.
Return value	See the `cReturn` property of the `MenuItem` object above.
Example	This script shows a mixed menu structure using most of the features of the `popUpMenuEx` method. `var cCH = app.popUpMenuEx ({cName: "Top menu checked and disabled", bMarked:true, bEnabled:false}, {cName: "-"}, {cName: "First parent menu", oSubMenu: [{cName: "Second level menu (submenu), the first item"}, { cName: "Second level menu (submenu), the second item", oSubMenu: {cName:"Third level menu (sub-submenu), a single item", cReturn: "0"} }] }, {cName: "Second item in the main menu"}, {cName: "Third item", bMarked:true, cReturn: "1"}) app.alert("You selected the \"" + cCH + "\" menu item");`

Related concepts
popUpMenu

response

Description	Launches a dialog box with a question and a free text field for the answer.
Parameters	`cQuestion` — The question to display in the dialog box. `cTitle` — (optional) The dialog box title line. Power PDF adds a warning text to that, to make sure, that the user is aware of the JavaScript activity. `cDefault` — (optional) The default answer as a string. `bPassword` — (optional) Set this to `true` to show asterisks or bullets to mask the response. The default is `false`. `cLabel` — (optional) A short text for the label placed front of the text input field.
Return value	The content of the response as a string.

Example	Run this code to input a nickname, then pop up a greeting with that nickname.
	`var s = app.response({`
	`  cQuestion: "What is your nickname?",`
	`  cTitle: "Welcome",`
	`  cDefault: "John",`
	`  cLabel: "Nickname:"`
	`});`
	`app.alert("Welcome on board, "+ s +"!", 3);`

setInterval

Description	This method sets a timer with the given time period, executes the script when this period is over and restarts the timer, so the script runs multiple times. Store the returned `interval` object in a variable in order to keep it working. Use `clearInterval` on that variable to cancel the timer.
Parameters	`cExpr` — The script to execute. `nMilliseconds` — The time period in milliseconds.
Return value	`interval` object
Example	Check the example provided with the `Field.buttonAlignY` method.

Related concepts
clearTimeOut
clearInterval
setTimeOut
Field.buttonAlignY

setTimeOut

Description	This method sets a timer with the given time period and executes the script (one time only) when this period is over. Store the returned timeout object in a variable in order to keep it working. Use `clearTimeOut` on that variable to cancel the timer.
Parameters	`cExpr` — The script to execute. `nMilliseconds` — The time period in milliseconds.
Return value	`timeout` object
Example	Check the example provided with the `Field.buttonAlignY` method.

Related concepts
clearTimeOut
clearInterval
setInterval
setTimeOut
Field.buttonAlignY

Bookmark

This object represents a junction in the bookmark tree that is displayed in the Bookmarks panel. Bookmarks help the user to move through the topics as required and serves as the source for the table of contents.

Bookmark properties

children

Description	An array of `Bookmark` objects that are children of the given bookmark under the bookmark tree.
Type	Array
Access	Read

Example	List all bookmarks in the document to the console.
	`function ListBookmarks(oBK, nDepth)`
	`{`
	`var s = "";`
	`for (var i = 0; i < nDepth; i++) s += "--- ";`
	`console.println(s + "+-" + oBK.name);`
	`if (oBK.children != null)`
	`  for (var i = 0; i < oBK.children.length; i++)`
	`  ListBookmarks(oBK.children[i], nDepth + 1);`
	`}`
	`console.show(); console.clear();`
	`console.println("Listing all bookmarks in the document.");`
	`ListBookmarks(this.bookmarkRoot, 0);`

Related concepts

parent

Doc.bookmarkRoot

color

Description	Determines the color of the bookmark.
Values	Property values are specified using the RGB, gray, or CMYK color.
Type	Array
Access	Read/Write

Example	This script colors the top level bookmarks to blue, green and magenta.
	`var oBK = this.bookmarkRoot;`
	`var n = 0;`
	`var mycolors = new Array(color.blue, color.green, color.magenta);`
	`for (var i = 0; i < oBK.children.length; i++) {`
	`  oBK.children[i].color = mycolors[n];`
	`  n++;`
	`  if (n > 2) n = 0;`
	`}`

Related concepts
style

doc

Description	Specifies the document object containing the bookmark.
Type	Object
Access	Read

name

Description	The bookmark text string that is displayed in the navigational panel.
Type	String
Access	Read/Write

Example	The following script displays the top bookmark in the console.
	`var oBK = this.bookmarkRoot.children[0];`
	`console.println( "Name of the very first bookmark: " + oBK.name );`

open

Description	Determines whether the child subtree of the bookmark opens (`true`) or collapses (`false`) in the navigational panel.
Type	Boolean
Access	Read/Write

parent

Description	Specifies the parent bookmark of the `Bookmark` object. It has a `null` value if given bookmark is the root bookmark.
Type	Object
Access	Read

Related concepts
children
Doc.bookmarkRoot

style

Description	Represents the style of the bookmark font.
Values	• `0`: normal • `1`: italic • `2`: bold • `3`: bold-italic
Type	Integer
Access	Read/Write

Example	This script turns the first top-level bookmark bold-italic.
	`var oBK = this.bookmarkRoot.children[0];`
	`oBK.style = 3;`

Related concepts
color

Bookmark methods

createChild

Description	Forms a new child bookmark at the given location.
Parameters	`cName` — The bookmark label that is displayed in the navigational panel. `cExpr` — An expression that is assessed when the bookmark is clicked. It is an optional parameter and its default value is no expression. `cIndex` — The zero-based array index in the children array, where to place the new child. This is an optional parameter, and its default value is zero.

Example	Insert a child bookmark at the top of the bookmark hierarchy that moves to the next page in the document.
	`this.bookmarkRoot.createChild(">> Move to the next page >>",`
	`"this.pageNum++");`

Related concepts
children
insertChild
remove
execute

execute

Description	Performs the action assigned to the bookmark.

Example	Run the code detailed at the setAction method to bind a sound alert action to the bookmark on the very top. Then, if you bind the following script to a button click, then clicking on the button will also play the alarm.
	`var oBM = bookmarkRoot.children[0];`
	`oBM.execute();`

Related concepts
createChild

insertChild

Description	Adds the specified bookmark as a child bookmark to the given position in the tree of bookmarks. If the bookmark is already present in the bookmark tree, then it is unreferenced before adding it again. This prevents situations where a bookmark gets duplicated as its own descendant.
Parameters	`oBookmark` — The bookmark object to add as a child. `nIndex` — The zero-based array index in the children array, where to place the new child. This is an optional parameter, and its default value is zero.

Example	Bind this script to a button click in a document with a well developed multilevel bookmark tree. Then click to pull the second chapter under the first one, one level inside. As you click multiple times the bookmark tree gets scrambled.
	`var oBM = bookmarkRoot.children[1];`
	`bookmarkRoot.children[0].insertChild(oBM);`

Related concepts
children
createChild
remove

remove

Description	Deletes the bookmark as well as its children from the bookmark tree.
Example	Removes the first second-level bookmark from the document. `bookmarkRoot.children[0].children[0].remove();`

Related concepts
children
createChild
insertChild

setAction

Description	Binds a JavaScript action to a bookmark. This method overwrites the existing action for this bookmark.

Parameters	`cScript` — Specifies the JavaScript code to execute when the bookmark is clicked.

Example	Bind a sound alert action to the bookmark on the very top. After this script finished, the top bookmark will play the default sound on click.
	`var oBM = bookmarkRoot.children[0];`
	`oBM.setAction("app.beep(0);");`

Related concepts
setPageAction
setAction
app.beep

Certificate

The object facilitates the read-only access for the properties of an X.509 public key certificate.

This object has no security constraints.

Related concepts
RDN
Field.signatureInfo
Field.signatureValidate

Certificate properties

binary

Description	Specifies the raw bytes of the certificate as a hex-encoded string.
Type	String
Access	Read

issuerDN

Description	Unique name of the certificate issuer, that is returned as an `RDN` object.
Type	`RDN` object
Access	Read

Related concepts
RDN object

MD5Hash

Description	Represents the MD5 hash of the certificate as a hex-encoded string. The property can serve as the source of a unique fingerprint of this certificate.
Type	String
Access	Read

SHA1Hash

Description	Represents the SHA1 hash of the certificate as a hex-encoded string. The property can serve as the source of a unique fingerprint of this certificate.
Type	String
Access	Read

serialNumber

Description	A distinctive identifier for the certificate object, used along with `issuerDN`.
Type	String
Access	Read

subjectCN

Description	The signer's common name.
Type	String
Access	Read

Related concepts
RDN object

subjectDN

Description	The unique name of the signer returned as an `RDN` object.
Type	`RDN` object
Access	Read

Related concepts
RDN object

console

This is a static object that facilitates access to the JavaScript console for executing JavaScript and displaying error messages.

console methods

show

Description	Displays the console window.
Example	This code shows the console window. `console.show();`

hide

Description	Shuts down the console window, but keeps its content for future use.
Example	This code closes the console window. `console.hide();`

printIn

Description	Prints the string value to the console window, together with the closing carriage return and line feed tokens (CR/LF). The next output will start in a new line.
Parameters	`cMessage`: The message to print (string).
Example	This code prints a status message to the console. `console.println("Processing started at " + util.printd("yyyy/mm/dd",new Date());`

clear

Method	**clear**
Description	Cleans the console window buffer of any output. The next line of output will show up at the top.
Example	This code clears the console window. `console.clear();`

Data

The `Data` object is the depiction of an embedded file or data stream that is kept embedded in the document. Data objects can be added from a file, queried, and extracted.

Related concepts

Doc.createDataObject
Doc.dataObjects
Doc.exportDataObjects
Doc.getDataObject
Doc.importDataObject
Doc.removeDataObject
Doc.openDataObject
Doc.getDataObjectContents
detDataObjectContents

Data properties

creationDate

Description	Specifies the creation date of the embedded file.
Type	Date
Access	Read

description

Description	The description meant for the data object.
Type	String
Access	Read/Write

MIMEType

Description	Specifies the MIME type related to the data object.
Type	String
Access	Read

modDate

Description	Specifies the modification date of the embedded file.
Type	Date
Access	Read

name

Description	The name related to the data object.
Type	String
Access	Read

Example	This script lists all data objects to the console.
	`var oDA = this.dataObjects;`
	`for (var i = 0; i < oDA.length; i++)`
	`  console.println("Data Object[" + i + "]=" + oDA[i].name);`

Related concepts
Data object

path

Description	Specifies the file name and the extension of the embedded file.
Type	String
Access	Read

size

Description	Specifies the size (in bytes) of the uncompressed data object.
Type	Number
Access	Read

Doc

The `Doc` object represents a PDF document open in Power PDF, providing methods and properties to access content. You can grab a `Doc` object in various ways:

- If you refer to the `this` keyword in an event action script belonging to a field or document, usually points to the `Doc` object of the current document. Check the example provided with the `app.popUpMenu` method (see popUpMenu).
- The `app.activeDocs` property is an array, which has a `Doc` object for each of the open documents. Check the example provided with the `app.activeDocs` property (see activeDocs).
- The `target` property of some events refers to the corresponding `Field` object, which has a `doc` property referring to the current document as a `Doc` object. The target property of some other events directly refers to the current `Doc` object.

Doc Properties

author

Description	This property holds the author metadata of the document.
Type	String
Access	Read/Write
Example	This script sets the author of the document to `"John Doe"`. `this.author = "John Doe";`

Related concepts
info

bookmarkRoot

Description	This is the root bookmark of the bookmark tree, designed for a programmatic purpose. Refer to `bookmarkRoot` and reach the elements of the bookmark tree using the `children` and `parent` properties.
Type	Object
Access	Read

Related concepts
Bookmark
Bookmark.children
Bookmark.parent

calculate

Description	Set this property `true` to enable calculations for this document. The `Doc.calculate` property replaces the `app.calculate` property, which is now deprecated.
Type	Boolean
Access	Read/Write

creationDate

Description	This property holds the creation date of the document. This property is obsolete, use the `info` property instead.
Type	Date
Access	Read

Related concepts
info

creator

Description	This property holds the name of the tool which was used to create this document. This property is obsolete, use the `info` property instead.
Type	String
Access	Read

Related concepts
info

dataObjects

Description	This array contains all the named `Data` objects of the document.
Type	Array
Access	Read

Example	`var oDA = this.dataObjects;`
	`for (var i = 0; i < oDA.length; i++)`
	`  console.println("Data Object[" + i + "]=" + oDA[i].name);`

Related concepts
createDataObject
openDataObject
getDataObject
getDataObjectContents
importDataObject
removeDataObject
setDataObjectContents

delay

Description	Redrawing of field objects is automatic and comes with property changes as necessary. In case of a massive amount of JavaScript property change requests, Power PDF performs better if you delay the redraw. Set this property to `true` to disable automatic redraw until `delay` property is set back to `false`.
Type	Boolean
Access	Read/Write

Related concepts
Field.delay

dirty

Description	This property indicates if the document was changed and saving is necessary. Set this property to false if you made only insignificant changes (such as a status field update) to the document, and you would like to suppress the Do you want to save changes to...? alert when closing. If the document was never saved, then setting the `dirty` property to `false` cannot block the alert on close.
Type	Boolean
Access	Read/Write

Example	The following script fills a text field (`StatusTextBox1`) with a status message while keeping the previous save state of the document.
	`var oTB = this.getField("StatusTextBox1");`
	`var d = this.dirty;`
	`oTB.value = "Please fill in the fields.";`
	`this.dirty = d;`

docID

Description	This property is an array of two strings. Both strings uses hex-encoded binary format. • The first string contains a hash derived from the original content, when the document was first created and saved. • The second string is updated with each incremental save operation, derived from the actual content of the document.
Type	Array
Access	Read
Example	The following script displays both `docID` strings on the console, separated by a colon. `console.println(this.docID[0] + " : "+  this.docID[1]);`

documentFileName

Description	The file name of the document, with the extension, not including the device-independent path. If the document was not saved yet, then this property returns the name of the `.tmp` temporary file.
Type	String
Access	Read

Example	Run the following script to a display the file name on the console.
	`console.println("The document file name with extension is:");`
	`console.println(this.documentFileName);`

Related concepts
filesize
path
URL

external

Description	This property is `true`, if the document is open in an external application, such as a browser.
Type	Boolean
Access	Read
Example	This script displays false on the console, if you run the PDF in the Power PDF. `console.println(this.external);`

filesize

Description	The size of the document file in bytes. If the document was not saved yet, then this property returns with an unreasonable value.
Type	Integer
Access	Read
Example	Run the following script to a display the size of the document on the console. `console.println("The document file size in bytes: " + this.filesize);`

Related concepts
documentFileName
path
URL

hidden

Description	The `hidden` property is `true` if the document window is hidden. This may occur because it was opened hidden, or because it is operating in batch mode.
Type	Boolean
Access	Read

icons

Description	This property is an array, containing all named icons in the document icon-tree. Has a `null` value, if there are no named icons.
Type	Array
Access	Read

Example	Create a button named Button1 prior running this script, which places a combo box on the right of the button and lists every named icon from the document.
	`var oBT = this.getField("Button1")`
	`var listRect = oBT.rect;`
	`listRect[0] = oBT.rect[2];`
	`listRect[2] = oBT.rect[2] + 144; // offset to the button`
	`var myIcons = new Array();`
	`var list = addField("IconList", "combobox", 0, listRect);`
	`list.textSize = 14;`
	`list.strokeColor = color.black;`
	`for (var i = 0; i < this.icons.length; i++)`
	`  myIcons[i] = this.icons[i].name;`
	`list.setItems(myIcons);`

Related concepts

addIcon
getIcon
importIcon
removeIcon
Field.buttonGetIcon
Field.buttonImportIcon
Field.buttonSetIcon

info

Description	Reach or create an object in the document information dictionary of the PDF document. You can use the `author`, `keywords`, `modDate`, `title`, and `subject` Doc properties to change corresponding document tree elements. The rest of the elements are read only.

Values	The following elements are provided by default.: • `Author` • `CreationDate` • `Creator` • `ModDate` • `Producer` You can add `keywords`, `subject`, `modDate`, or `title` elements by using the corresponding `Doc` properties.
Type	Object
Access	Read

Example	This script displays all the available document information dictionary elements.
	`for (var i in this.info)`
	`  console.println(i + ": "+ this.info[i])`

Related concepts
author
creator
creationDate
keywords
modDate
producer
subject
title

keywords

Description	This array of strings helps to categorize the document.
Type	Array
Access	Read/Write

Example	This script sets the keywords on the console:
	`this.keywords = ["PDF", "Guide", "How-to"];`
	This script lists the keywords on the console:
	`for (var i in this.info)`
	`  if (i == "Keywords")`
	`  console.println(i + ": "+ this.info[i]);`

Related concepts
author
creator
info
producer

subject
title

layout

Description	Changes the page view for the document..
Values	`SinglePage` — The view is restricted to a single page, you can move between pages, but no free (smooth) scrolling. `OneColumn` — Shows pages in a single column and you can scroll freely. `TwoColumnLeft` — Shows two pages side by side, starting with the first page on the left. You can scroll freely. `TwoColumnRight` — Shows two pages side by side, starting with the first page on the right. You can scroll freely.
Type	String
Access	Read/Write
Example	This script switches to single page.view: `this.layout = "SinglePage";`

modDate

Description	This property stores the date and time of the last document save operation. This property is obsolete, use the `info` property instead.
Type	Date
Access	Read
Example	This script prints the last modification date and time in default wild string format: `console.println(this.modDate);`

Related concepts
info

mouseX

Description	Gets the x coordinate of the mouse pointer position in the default user space, related to the page.
Type	Number
Access	Read

Example	These scripts let you check mouse coordinates. First create a document JavaScript function, which prints current mouse coordinates to the console:
	`function getMouseCoor() {`
	`console.println( "("+this.mouseX+","+ this.mouseY+")" );`
	`}`
	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. After you click this button, the `getMouseCoor` function gets called automatically in every 100 ms.
	`var oMI = app.setInterval("getMouseCoor()", 100);`
	Add another button and bind this code to its `Mouse Up` trigger. Click this button to clear the interval, so coordinates will not display anymore.
	`app.clearInterval(oMI);`

Related concepts
mouseY
app.clearInterval
app.setInterval

mouseY

Description	Gets the y coordinate of the mouse pointer position in the default user space, related to the page.
Type	Number
Access	Read

Related concepts
mouseX

nocache

Description	Set this property to `true` to turn off data caching in the internet browser.
Type	Boolean
Access	Read/Write

numFields

Description	This property holds the number of fields used all over the document.
Type	Integer
Access	Read

Related concepts
getNthFieldName
Field.strokeColor

numPages

Description	This property holds the number of pages in the document.
Type	Integer
Access	Read
Example	Run this script to print the number of pages to the console. `console.println("Number of pages: " + this.numPages);`

path

Description	The device-independent path of the document, including the file name and extension. Use the `documentFileName` property to get only the file name of the document with extension.
Type	String
Access	Read

Related concepts
documentFileName
URL

pageNum

Description	Read this property to obtain the number of the current page of the document. Set this property to move to the specified page. Page numbering is zero-based, so the very first page has 0 as page number.
Type	Integer
Access	Read/Write
Example	Run this script to move to the first page of the document. `this.pageNum = 0;`

producer

Description	This property holds the name of the tool which was used to create this document. This property is obsolete, use the `info` property instead.
Type	String
Access	Read

Related concepts
info

securityHandler

Description	Reveals the name of the security handler used to encrypt the document. Has a `null` value if no encryption applied to the document.
Type	String
Access	Read

Example	This script displays the name of the security handler currently used, on the console.
	`console.println(this.securityHandler != null ?`
	`"The current document security handler is:" + this.securityHandler`
	`+ "." : "This document is not encrypted.");`

subject

Description	This property holds the subject metadata of the document.
Type	String
Access	Read/Write
Example	This script sets the subject of the document to `Support article`. `this.subject = "Support article";`

Related concepts
info

title

Description	This property holds the title metadata of the document.
Type	String
Access	Read/Write
Example	This script sets the title of the document to `JavaScript Guide`. `this.title = "JavaScript Guide";`

Related concepts
info

URL

Description	Returns with the document URL. If the document is saved locally, then returns with a device independent path, starting with the `file///` scheme.
Type	String
Access	Read

Example	Save your PDF document into the `temp` folder on drive `C`, under the name `tryURL.pdf`. Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click to run it.
	`console.println(this.URL);`
	Check the output on the console:
	`file:\\\C|\temp\tryURL.pdf`

zoom

Description	Holds the current zoom ration of the document window/tab in percents.
Type	Number
Values	Between `8.33` and `6400`
Access	Read/Write

Example	This script triples the current zoom ratio:
	`this.zoom *= 3;`
	Run this script to print the current zoom ratio to the console.
	`console.println(this.zoom);`

zoomType

Description	Holds the current zoom type of the document. See the table below for `zoomtype` object constants.
Type	String
Access	Read/Write
Example	Set the zoom type of the document to fit the page. `this.zoomType = zoomtype.fitP;`

zoomtype object constants

Constant	Description
`zoomtype.none`	Leaves the zoom ratio intact.
`zoomtype.fitP`	Fits to the page.
`zoomtype.fitW`	Fits to the page width.
`zoomtype.fitH`	Fits to the page height.
`zoomtype.fitV`	Fits to the page width leaving no gap.

Doc Methods

addIcon

Description	Adds and names a new icon to the icon tree of the document.
Parameters	`cName` — The name to assign to the icon. `icon` — The icon object to add to the icon tree.

Example	Create a button (`Button1`) with icon, then go to the Actions tab in its Button properties dialog box. Add the following event handler scripts to the Mouse Up trigger to run. The script adds the icon to the document icon tree when you click on the button.
	`var oBT = this.getField("Button1");`
	`this.addIcon("Icon1", oBT.buttonGetIcon());`
	After the first srcipt finished, add another button (`Button2`) and add the following event handler scripts to its Mouse Up trigger to run. The script lists the named icons on the console.
	`for (var i = 0; i < this.icons.length; i++)`
	`  console.println(this.icons[i].name);`

addField

Description	Adds a new field to the document.
Parameters	`cName` — The name to assign to the field. You may use the dot as a separator between a parent node and its child. For example, specifying `name.first` creates a parent node (`name`), and its child node (`first`). `cFieldType` — The type of the field. Valid types follow: • `text` • `button` • `combobox` • `listbox` • `checkbox` • `radiobutton` • `signature` `nPageNum` — The 0-based index of the target page where the field will be placed. `cCoords` — An array, defining the boundaries of the field on the page by specifying the coordinates of the edges. See the `rect` property of the Field object for details.
Returns	The `Field` object just created.

Example	Run the following script to create a blank signature named Signature1.
	`var ip = 72; // point/inch rate`
	`var sRect = this.getPageBox( {nPage: 0} );`
	`sRect[0] += 0.5*ip;// half inch from the from upper left corner of page`
	`sRect[1] -= 0.5*ip;`
	`sRect[2] = sRect[0]+0.5*ip;// 0.5 inch width`
	`sRect[3] = sRect[1] - 24;// 24 points in height`
	`// Now to construct a blank signature field`
	`var oSI = this.addField("Signature1", "signature", 0, sRect);`

Related concepts
Field.rect

addLink

Description	Places a link on the specified page, sized according to the coordinates. The user should have permission to add links to the document.
Parameters	`nPage` — The 0-based index of the target page where the link will be placed. `cCoords` — An array, defining the boundaries of the link on the page by specifying the coordinates of the edges of the link area. See the `rect` property of the `Field` object for details.
Returns	The `Field` object just created.

Related concepts
Field.rect

addScript

Description	Defines a document-level script.
Parameters	`cName` — The unique name for the script. If there is a script already defined with this name, then `addScript` replaces it. `cScript` — The JavaScript code to add.
Example	Adds a script, that plays a default beep. Check the `Document JavaScript` window for the newly added script. `this.addScript("Beeper", "app.beep(4);");`

Related concepts
removeScripts
setAction
setPageAction
Bookmark.setAction
Field.setAction

calculateNow

Description	Call this method to compute all calculations in the document.

Example	User edits in calculated fields (and related ones) may cause significant lag. To avoid that, turn off calculation and wait until all the data input provided, then turn calculation on again and refresh the fields.
	`// Turn calculations off for smoother user experience`
	`this.calculate = false;`
	`// Bind this code to the Mouse Up trigger of a Calculate button`
	`// Turn calculations on`
	`this.calculate = true;`
	`// Refresh all calculated field content`
	`this.calculateNow();`

Related concepts
calculate

closeDoc

Description	Closes the document.
Parameters	`bNoSave` — (optional) This Boolean value determines whether to save the document or not. • `false` — The Save as dialog box displays if the document was not saved or had unsaved changes. This is the default value. • `true` — The document closes, discarding all the unsaved changes. Use with caution, because the user is not prompted before close.

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. All open documents will be closed as you click the button.
	`var aDocs = app.activeDocs;`
	`for( var i in aDocs ) aDocs[i].closeDoc();`

Related concepts
App.activeDocs

createDataObject

Description	Creates a data object with a string content.
Parameters	`cName` — The name associated with the data object. Data objects created via JavaScript always have a name, which makes possible to reach them in the `dataObjects` property. Embedded objects inserted via the UI do not have a name and cannot be accessed from JavaScript. `cValue` — A string, containing the data itself. `cMIMEType` — (optional) The MIME type (the encoding/decoding standard) of the data, which is responsible for the conversion between the string format and the original data format.
Example	`this.createDataObject("Data.txt",` `"This is the placeholder for some data.");`

Related concepts
dataObjects
getDataObject
getDataObjectContents
importDataObject
openDataObject
removeDataObject
setDataObjectContents
exportDataObject

deletePages

Description	Deletes the specified pages. Deletes the first page only, if there are no parameters. There must be at least one page in the document to succeed.
Parameters	`nStart` — (optional) The 0-based index of the first page to delete. The default value is `0`, which means the first page. `nEnd` — (optional) The 0-based index of the last page to delete. If not specified, then equals to `nStart`, which means only one page to delete.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to delete page 2 and page 3. `this.deletePages(1,2);`

exportAsFDF

Description	Export the form field values to an FDF file.
Parameters	`bAllFields` — (optional) Set to `true` to export all fields, even the ones with no values. Set to `false` to include only the fields with value. `bNoPassword` — (optional) Set this to `true` to not include password fields. `aFields` — (optional) This is either a string containing a single field name, or an array of strings, containing all field names to include in the export. You may list parent (non-terminal) fields to include all their child fields in the export. • If `bNoPassword` is `true`, then this affects the set of fields: password fields are excluded from the export, even if they are specified in `aFields`. • If this parameter is an empty array, then no fields are exported. • If this parameter is not specified or has `null` as value, then this has the same effect, just as all fields were listed in an array. `cPath` — (optional) Specify the device independent path and file name. `.fdf` should be used as a file extension. If this parameter is missing, then a dialog box prompts for a path and file name.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to export fields, a dialog box prompts for the file name. `this.exportAsFDF();`

Related concepts
Field.getArray
exportAsFDFStr
exportAsText
exportAsTextStr
exportAsXFDF
exportAsXFDFStr

exportAsFDFStr

Description	Export the form field values to a string, in FDF format.

Parameters	`bAllFields` — (optional) Set to `true` to export all fields, even the ones with no values. Set to `false` to include only the fields with value. `bNoPassword` — (optional) Set this to `true` to not include password fields. `aFields` — (optional) This is either a string containing a single field name, or an array of strings, containing all field names to include in the export. You may list parent (non-terminal) fields to include all their child fields in the export. • If `bNoPassword` is `true`, then this affects the set of fields: password fields are excluded from the export, even if they are specified in `aFields`. • If this parameter is an empty array, then no fields are exported. • If this parameter is not specified or has `null` as value, then this has the same effect, just as all fields were listed in an array. `cHRef` — (optional) You may specify a source or target file, which is enclosed in the FDF output as an `F` key.
Returns	The form field values exported in FDF format. Has the same content as it was provided by the `exportAsFDF` method, adding `cHRef` as extra.

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to export fields to the sFDF string, including an URL. Check the generated string on the console output.
	`var sFDF = this.exportAsFDFStr({`
	`cHRef: "http://www.niance.com/"`
	`});`
	`console.println(sFDF);`

Related concepts

Field.getArray
exportAsFDF
exportAsTextStr
exportAsText
exportAsXFDF
exportAsXFDFStr

exportAsText

Description	Export the form field values in tab-delimited format to a plain text file. Correctly handles multi-line text contents and quotes. The first line of the output contains the field names, the second line includes the corresponding field values.

Parameters	`bNoPassword` — (optional) Set this to `true` to not include password fields. `aFields` — (optional) This is either a string containing a single field name, or an array of strings, containing all field names to include in the export. You may list parent (non-terminal) fields to include all their child fields in the export. • If `bNoPassword` is `true`, then this affects the set of fields: password fields are excluded from the export, even if they are specified in `aFields`. • If this parameter is an empty array, then no fields are exported. • If this parameter is not specified or has `null` as value, then this has the same effect, just as all fields were listed in an array. `cPath` — (optional) Specify the device independent path and file name. `.txt` should be used as a file extension. If this parameter is missing, then a dialog box prompts for a path and file name.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to export fields to the `export.txt`, within the `Temp` folder on drive `C`, overwriting the existing file (if there is any). `this.exportAsText({cPath: "/C/Temp/export.txt"});`

Related concepts

exportAsFDF
exportAsFDFStr
exportAsTextStr
exportAsXFDF
exportAsXFDFStr

exportAsTextStr

Description	Export the form field values in tab-delimited format to a string. Correctly handles multi-line text contents and quotes. The first line of the output contains the field names, the second line includes the corresponding field values.
Parameters	`bNoPassword` — (optional) Set this to `true` to not include password fields. `aFields` — (optional) This is either a string containing a single field name, or an array of strings, containing all field names to include in the export. You may list parent (non-terminal) fields to include all their child fields in the export. • If `bNoPassword` is `true`, then this affects the set of fields: password fields are excluded from the export, even if they are specified in `aFields`. • If this parameter is an empty array, then no fields are exported. • If this parameter is not specified or has `null` as value, then this has the same effect, just as all fields were listed in an array.
Returns	The form field values exported in tab-delimited format. Has the same content as it was provided by the `exportAsText` method.

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to export fields (except password fields) to the `sText` string. Check the generated string on the console output.
	`var sText = this.exportAsTextStr({`
	`  bNoPassword: true,`
	`});`
	`console.println(sText);`

Related concepts
exportAsFDF
exportAsFDFStr
exportAsText
exportAsXFDF
exportAsXFDFStr

exportAsXFDF

Description	Export the form field values to an XFDF file.
Parameters	`bNoPassword` — (optional) Set this to `true` to not include password fields. `aFields` — (optional) This is either a string containing a single field name, or an array of strings, containing all field names to include in the export. You may list parent (non-terminal) fields to include all their child fields in the export. • If `bNoPassword` is `true`, then this affects the set of fields: password fields are excluded from the export, even if they are specified in `aFields`. • If this parameter is an empty array, then no fields are exported. • If this parameter is not specified or has `null` as value, then this has the same effect, just as all fields were listed in an array. `cPath` — (optional) Specify the device independent path and file name. `.xfdf` should be used as a file extension. If this parameter is missing, then a dialog box prompts for a path and file name.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to export fields, a dialog box prompts for the file name. `this.exportAsXFDF();`

Related concepts
exportAsFDF
exportAsFDFStr
exportAsText
exportAsTextStr
exportAsXFDFStr

exportAsXFDFStr

Description	Export the form field values to a string, in XFDF format.

Parameters	`bNoPassword` — (optional) Set this to `true` to not include password fields. `aFields` — (optional) This is either a string containing a single field name, or an array of strings, containing all field names to include in the export. You may list parent (non-terminal) fields to include all their child fields in the export. • If `bNoPassword` is `true`, then this affects the set of fields: password fields are excluded from the export, even if they are specified in `aFields`. • If this parameter is an empty array, then no fields are exported. • If this parameter is not specified or has `null` as value, then this has the same effect, just as all fields were listed in an array. `cHRef` — (optional) You may specify a source or target file, which is enclosed in the XFDF output as an `F` key.
Returns	The form field values exported in FDF format. Has the same content as it was provided by the `exportAsFDF` method, adding `cHRef` as extra.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to export the fields listed in the `aFields` parameter to the `sXFDF` string. Check the generated string on the console output. `var sXFDF = this.exportAsXFDFStr({ bNoPassword: true, aFields: ["Text16", "Text17", "Text18", "Text27", "Text29", "Text30"], }); console.println(sXFDF);`

Related concepts

exportAsFDF
exportAsFDFStr
exportAsText
exportAsTextStr
exportAsXFDF

exportDataObject

Description	Exports the embedded data object to a file.

Parameters	`cName` — The name associated with the data object. Data objects created via JavaScript always have a name, which makes possible to reach them in the `dataObjects` property. Embedded objects inserted via the UI do not have a name and cannot be accessed from JavaScript. `nLaunch` — (optional) Controls whether to ask for a file name and open the file after save. Valid values are: • `0` — No launch after save. • `1` — Power PDF prompts the user for a file name, then launches the file with the associated application. Prior to launch a security confirmation box appears, if the file format is other than PDF. • `2` — Power PDF saves the file to a temporary path with a random name, so does not prompt the user. Prior to launch a security confirmation box appears, if the file format is other than PDF.

Example	Use `importDataObject` to embed an external file into the PDF document and name it as `DataObject1`. Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which prompts the user for a path and file name and saves the data object, then launches it.
	
	`this.exportDataObject({`
	`  cName: "DataObject1",`
	`  nLaunch: 1`
	`});`

Related concepts

dataObjects
createDataObject
getDataObject
getDataObjectContents
importDataObject
openDataObject
removeDataObject
setDataObjectContents

flattenPages

Description	This method converts annotations (including non-printable ones) to passive page content within the specified page range. If no range specified, then all pages are involved.
Parameters	`nStart` — (optional) The 0-based index of the first page to operate on. Omitting `nStart` results in an empty page range, so the method does not process any page. `nEnd` — (optional) The 0-based index of the last page of the range. If not specified, then extends the range to the end of the document, so the method flattens the page specified by the first parameter together with all the following pages.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which flattens annotations on the first page only. `this.flattenPages();`

getAnnot

Description	This method provides access to the annotations by page number and name.
Parameters	`nPage` — The 0-based index of the page which contains the annotation. `cName` — The name associated with the `Annotation` object. Annotations created by the user get a random name. Use the `getAnnots` method to reach those objects and get their names.
Returns	`Annotation` object
Example	See the `Annotation.name` property.

Related concepts
Annotation.name
getAnnots

getAnnots

Description	This method scans the document for annotations, using the criteria specified by the parameters.
Parameters	`nPage` — The 0-based index of the target page where to look for annotations. If not specified, then the search runs throughout the whole document. `nSortBy` — Determines the sorting order of the array returned. See the table below for valid constants.
Returns	Array of `Annotation` objects
Example	This script lists all annotations in the document with details.

```
var aAnnots = this.getAnnots({
  nPage:0,
  nSortBy: ANSB_Author,
  bReverse: false
});
console.show();
console.println("Total number of annotations: " + aAnnots.length);
var s = "%s in a %s annotation named as \"%s\" said: \"%s\"";
for (var i = 0; i < aAnnots.length; i++) {
  console.println(util.printf(s, aAnnots[i].author, aAnnots[i].type,
 aAnnots[i].name,
  aAnnots[i].contents));
  var oAN = this.getAnnot(0, aAnnots[i].name);
  if (oAN == null)
    console.println("Not Found " + aAnnots[i].name)
  else
    console.println("Found " + aAnnots[i].name + "! type: " +
 oAN.type);
  }
```

nSortBy constants

Constant	Description
ANSB_None	Do not sort.
ANSB_Page	Sort by page number..
ANSB_Author	Sort by author.
ANSB_ModDate	Sort by the date of the last modification.
ANSB_Type	Sort by annotation type.

Related concepts

getAnnot

getDataObject

Description	Obtains a data object with the specified name.
Parameters	cName — The name associated with the data object. Data objects created via JavaScript always have a name, which makes possible to reach them in the dataObjects property. Embedded objects inserted via the UI do not have a name and cannot be accessed from JavaScript.
Returns	Returns with the object if found, returns with null otherwise.

Example	Use importDataObject to embed an external file into the PDF document and name it as DataObject1. Bind this code to the Mouse Up trigger of a button in the Button Properties dialog box. Click the button to run the script, which lists all details about DataObject1 on the console.
	`var oDA = this.getDataObject("DataObject1");`
	`console.println(oDA.name);`
	`console.println(oDA.path);`
	`console.println(oDA.size);`
	`console.println(oDA.MIMEType);`
	`console.println(oDA.modDate);`
	`console.println(oDA.creationDate);`

Related concepts

dataObjects
createDataObject
getDataObjectContents
importDataObject
openDataObject
removeDataObject
setDataObjectContents

getDataObjectContents

Description	Obtains a data object with the specified name.

Parameters	`cName` — The name associated with the data object. Data objects created via JavaScript always have a name, which makes possible to reach them in the `dataObjects` property. Embedded objects inserted via the UI do not have a name and cannot be accessed from JavaScript.
Returns	Returns with a `ReadStream` object.

Example	This script uses `importDataObject` to embed a text file (for example, `Test1.txt`) into the PDF document and names it as `Test1.txt`. Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which prompts for the file, then imports the file as a data object, and displays its content on the console.
	
	`this.importDataObject("Test1.txt");`
	`var oFile = this.getDataObjectContents("Test1.txt");`
	`var sContent = util.stringFromStream(oFile, "utf-8");`
	`console.println(sContent);`

Related concepts
dataObjects
createDataObject
getDataObject
importDataObject
openDataObject
removeDataObject
setDataObjectContents
util.stringFromStream

getField

Description	This method assigns a JavaScript variable to a field.
Parameters	`cName` — The name associated with the form field.
Returns	The `Field` object representing the form field.

Example	This script will set a greeting text to Button1.
	
	`var oBT = this.getField("Button1");`
	`oBT.buttonSetCaption("Hello World!");`

Related concepts
Field object
numFields

getIcon

Description	This method assigns a JavaScript variable to an icon.
Parameters	`cName` — The name associated with the icon.
Returns	The `icon` object.

Related concepts
addIcon
icons
importIcon
removeIcon
Field.buttonGetIcon
Field.buttonImportIcon
Field.buttonSetIcon

getNthFieldName

Description	This method returns with the name of the *n*th field in the document.
Parameters	`nIndex` — The 0-based index of the form field within the document.
Returns	String

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which lists all fields to the console.
	`for ( var i=0; i < this.numFields; i++) {`
	`  var fieldname = this.getNthFieldName(i);`
	`  console.println(i + ". " + fieldname);`
	`}`

Related concepts
Field object
getField
numFields

getPageBox

Description	This method retrieves the coordinates for the named box of the specified page in the rotated user space. Returns these coordinates in the form of a rectangle array (see rect for details).
Parameters	`cBox` — The type of the box according to *PDF Reference version 1.7*. Valid values follow: • `Art` • `Bleed` • `BBox` • `Crop` (default) • `Trim` `nPage` — (optional) The 0-based index of the page.
Returns	`rect` array (array of coordinates)
Example	See the `addField` method.

Related concepts
addField
setPageBoxes

getPageNthWord

Description	This method gets the *n*th word on the specified page. The scope embraces texts in field objects, button captions, and annotations.
Parameters	`nPage` — (optional) The 0-based index of the page. The default is `0`. `nWord` — (optional) The 0-based index of the word. The default is `0`. `bStrip` — (optional) Set this parameter to `false` to return the word with white space characters and punctuation. Set it to true (default) to remove these characters before return.
Returns	String

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which lists all words on the first page to the console.
	`var nW, sWord;`
	`nW = this.getPageNumWords(0);`
	`console.println("The first page has " + nW + " words:");`
	`for (var j = 0; j < nW; j++)`
	`  {`
	`  sWord = this.getPageNthWord(0, j);`
	`  console.println("Word #" + j + ": " + sWord);`
	`  }`

Related concepts
getPageNumWords
selectPageNthWord

getPageNumWords

Description	Queries the number of words on the specified page.
Parameters	`nPage` — (optional) The 0-based index of the page. The default is `0`.
Returns	Integer

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which reports the number of words to the console.
	`var nWords=0;`
	`for (var i = 0; i < this.numPages; i++)`
	`  nWords += getPageNumWords(i);`
	`console.println("This document contains " + nWords + " words.");`

Related concepts
selectPageNthWord

getPageRotation

Description	Queries the rotation angle of the specified page.
Parameters	`nPage` — (optional) The 0-based index of the page. The default is `0`.
Returns	Integer

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which reports the rotation angle of the first page to the console. `console.println("The rotation angle of the first page is " + this.getPageRotation(0) + " degrees.");`

Related concepts
setPageRotations

getPrintParams

Description	This method returns with a `PrintParams` object, representing the default print settings. You may alter some of the properties of the `PrintParams` object, then pass it to the print method to control print settings.
Returns	`PrintParams` object

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which loads current print settings, changes the destination to a `.prn` file, then prints the document.
	`var oPP = this.getPrintParams();`
	`// Set some properties, then print`
	`oPP.fileName = "/c/temp/myDoc.prn";`
	`pp.printerName = "";`
	`this.print(oPP);`

Related concepts
print

importDataObject

Description	Imports a file into the document as a data object.
Parameters	`cName` — The name associated with the data object. Data objects created via JavaScript always have a name, which makes possible to reach them in the `dataObjects` property. Embedded objects inserted via the UI do not have a name and cannot be accessed from JavaScript. `cDIPath` — (optional) The device-independent path to the file to embed. Power PDF prompts the user if this parameter is missing.
Returns	Returns `true` if the import was successful, an exception raises otherwise.

Example	Add the following JavaScript function to the document:.
	`function ListDataObjects()`
	`{`
	`var oDA = this.dataObjects;`
	`for (var i = 0; i < oDA.length; i++)`
	`  console.println("Data Object[" + i + "]=" + oDA[i].name);`
	`}`
	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which prompts the user for a data file and embeds it, then calls `ListDataObjects` to display the name of the new object on the console.
	`this.importDataObject("DataObject1");`
	`ListDataObjects();`

Related concepts
dataObjects
createDataObject
exportDataObject
getDataObject
getDataObjectContents
openDataObject
removeDataObject
setDataObjectContents

importAnFDF

Description	Imports form data from the specified file to load field values saved earlier.
Parameters	`cPath` — (optional) Specify the device independent path and file name. `.fdf` should be used as a file extension. If this parameter is missing, then a dialog box prompts for a path and file name.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to import `sample.fdf`. `this.importAnFDF("/c/temp/sample.fdf");`

Related concepts
importAnXFDF
importTextData

importAnXFDF

Description	Imports form data from the specified XML file to load field values saved earlier.
Parameters	`cPath` — (optional) Specify the device independent path and file name. `.xfdf` should be used as a file extension. If this parameter is missing, then a dialog box prompts for a path and file name.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to import `sample.xfdf`. `this.importAnXFDF("/c/temp/sample.xfdf");`

Related concepts
importAnFDF
importTextData

importTextData

Description	Imports form field values from the specified row in a tab-delimited text file. Searches for field names in the very first row (header) of the document, then loads data from the row specified. Updates all form field values, where a corresponding name found in the header.
Parameters	`cPath` — (optional) Specify the device independent path and file name. `.txt` should be used as a file extension. If this parameter is missing, then a dialog box prompts for a path and file name. `nRow` — (optional) Determines which row of the file to import. This index value is 0-based and starts with the line following the header row. If there are more than one data rows in the file, and `nRow` is missing, then Power PDF displays a dialog box and the user should pick a row.
Returns	Integer return code: • `-3` — Warning: Data is missing. • `-2` — Warning: The row select operation was cancelled by the user. • `-1` — Warning: The file select operation was cancelled by the user. • `0` — Success. • `1` — Error: Cannot open the file. • `2` — Error: Cannot load the data. • `3` — Error: Invalid row number was specified.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to export fields to the `export.txt`, within the `Temp` folder on drive `C`, overwriting the existing file (if there is any). `this.importAsText({cPath: "/C/Temp/export.txt"});`

Related concepts
importAnFDF
importAnXFDF

importIcon

Description	Imports a page as an icon from the specified page of the assigned PDF file. If there is no appropriate PDF file specified in the `cPath` parameter, then the Select icon dialog box shows up, and the user should browse to the file. In this case the user may switch to a supported file format other than PDF, the image files will be converted automatically.
Parameters	`cPath` — (optional) Device-independent path to the source image file. `nPage` — (optional) The 0-based index of the page in the source file to turn into an icon.

Returns	The method returns an integer error code: • `1` — The user cancelled the process by closing the dialog box. • `0` — The icon was imported successfully. • `-1` — The file could not be opened. • `-2` — The given page number was invalid.

mailDoc

Description	Saves the document, then sends it via e-mail in an attachment.
Parameters	`bUI` — (optional) If `true`, then the compose new mail window shows up, the message fields are populated using the rest of the parameters and the user may edit all the fields. If `false`, then Power PDF fills the message fields based on the parameters below. In this case only the `cTo` parameter is required. `cTo` — (optional) List of the recipients, separated by semicolon. `cCc` — (optional) List of the CC recipients, separated by semicolon. `cBcc` — (optional) List of the BCC recipients, separated by semicolon. `cSubject` — (optional) The subject line, limited to 64 KB. `cMsg` — (optional) The message body, limited to 64 KB.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to start a new mail window with the current document attached. `this.mailDoc({ bUI: false, cTo: "johndoe@nowhere.com", cCc: "johndoejr@nowhere.com", cSubject: "The Latest Document", cMsg: "Hi, please find the latest document attached in PDF." });`

Related concepts

mailForm

mailForm

Description	Exports form data in the document to an.`FDF` file then sends it via e-mail in an attachment.
Parameters	`bUI` — (optional) If `true`, then the compose new mail window shows up, the message fields are populated using the rest of the parameters and the user may edit all the fields. If `false`, then Power PDF fills the message fields based on the parameters below. In this case only the `cTo` parameter is required. `cTo` — (optional) List of the recipients, separated by semicolon. `cCc` — (optional) List of the CC recipients, separated by semicolon. `cBcc` — (optional) List of the BCC recipients, separated by semicolon. `cSubject` — (optional) The subject line, limited to 64 KB. `cMsg` — (optional) The message body, limited to 64 KB.

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to start a new mail window with the form filled data attached in a file. `this.mailForm({ bUI: false, cTo: "johndoe@nowhere.com", cCc: "johndoejr@nowhere.com", cSubject: "The Latest Document", cMsg: "Hi, please find the latest document attached in PDF." });`

Related concepts

mailDoc

movePage

Description	Moves a page to a new position within the document.
Parameters	`nPage` — (optional) The 0-based index of the page to move. The default value is `0`. `nAfter` — (optional) Determines where to move the page. Specify the 0-based index of the page after to insert the displaced page. Specify `-1` to move before the first page. The default value always appoints the last page of the document.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to move the first page to the last position. `this.movePage(0);`

openDataObject

Description	This method returns with the addressed embedded PDF document (data object) as a `Doc` object. This method does not open the specified embedded file, still provides access to its content, such as field values. An exception raises if the addressed embed is missing, or it is not a PDF document, or JavaScript access is not permitted.
Parameters	`cName` — The name associated with the data object. Data objects created via JavaScript always have a name, which makes possible to reach them in the `dataObjects` property. Embedded objects inserted via the UI do not have a name and cannot be accessed from JavaScript.
Returns	`Doc` object

Example	Prepare `test.pdf`, this file should contain a text field called `Text1`, containing a sample text.
	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click this button to run the code below which use `importDataObject` and prompts for a file to embed into the PDF document as a data object, and names it as `test.pdf`. (Provide `test.pdf` when required.)
	`this.importDataObject("test.pdf");`
	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which reaches into the embedded PDF file and retrieves the value of the field `Text1` to display it on the console.
	`var oDO = this.openDataObject("test.pdf");`
	`try {`
	`  var oFE = oDO.getField("Text1");`
	`  console.println(oFE.value);`
	`  oDO.closeDoc();`
	`} catch(e) { app.alert("Failed to retrieve data from the embedded test.pdf.");}`

Related concepts

dataObjects
createDataObject
getDataObjectContents
importDataObject
removeDataObject
setDataObjectContents
util.stringFromStream

print

Description	Prints the document or a selected page range in the document.

Parameters	`bUI` — (optional) Set this parameter to `true` to show up the print dialog and let the user set all parameters. `nStart` — (optional) The 0-based index of the first page to operate on. Omitting both `nStart` and `nEnd` results in the range of the whole document. Specifying only `nStart` (and omitting `nEnd`) results in a range consisted of a single page marked by `nStart`. `nEnd` — (optional) The 0-based index of the last page of the range. If only `nEnd` specified (and `nStart` is omitted), then the range is from the first page to `nEnd`. `bSilent` — (optional) Set this to `true` to hide the cancel printing dialog during print processing. The default value is `false`. `bShrinkToFit` — (optional) Set this to `true` to shrink oversized pages within the printable page area. The default value is `false`. `bPrintAsImage` — (optional) Set this to `true` to convert the content of the pages to images in the printing job. The default value is `false`. `bReverse` — (optional) Set this to `true` to print pages in reverse order. The default value is `false`. `bAnnotations` — (optional) Set this to false to exclude the annotations from the printing. The default value is `true`. `bPrintParams` — (optional) Pass a `PrintParams` object, which encompasses the printing settings, overriding other parameters.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which prints the current page. `this.print(false, this.pageNum, this.pageNum);`

Related concepts
getPrintParams

removeDataObject

Description	This method removes the specified data object from the document.
Parameters	`cName` — The name associated with the data object. Data objects created via JavaScript always have a name, which makes possible to reach them in the `dataObjects` property. Embedded objects inserted via the UI do not have a name and cannot be accessed from JavaScript.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which removes the embedded PDF file named as `DataObject1`. `this.removeDataObject("DataObject1");`

Related concepts
dataObjects
createDataObject
getDataObject
getDataObjectContents
importDataObject
openDataObject

setDataObjectContents
util.stringFromStream

removeField

Description	This method removes the specified field from the document. Removal impacts all representations of the field.
Parameters	`cName` — The field name to remove.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which removes the button (buttons are also field controls). `this.removeField("ButtonDummy");`

removeIcon

Description	This method removes the specified named icon from the document.
Parameters	`cName` — The icon name to remove.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which removes `Icon1`. `this.removeIcon("Icon1");`

removeScripts

Description	Deletes a document-level script.
Parameters	`cName` — The name of the script.
Example	Removes the script called `Beeper`, which was added by the `addScript` method. Check the `Document JavaScript` window for the change. `this.removeScript("Beeper");`

Related concepts
addScript
setAction
setPageAction
Bookmark.setAction
Field.setAction

resetForm

Description	Sets the default value for the specified fields.
Parameters	`aFields` — This array contains the names of the fields to reset. You may also add non-terminal fields to the array. If this parameter is omitted or has a `null` value, then the method resets all field in the document.

Example	This script resets the fields specified in the inline array constant. `this.resetForm(["TextBox1", "Name.First", "Name.Last"]);`

Related concepts
submitForm

scroll

Description	Scrolls the current view in a way, that the specified point on the page gets centered. Specify the coordinates of the location in points in the rotated user space. For details see the *PDF Reference version 1.7*.
Parameters	`nX` — The `x` coordinate in points, referring to the location to scroll. `nY` — The `y` coordinate in points, referring to the location to scroll.
Example	This script scrolls to the point that has a hundred points to the left and top of the page. `this.scroll(100,100);`

selectPageNthWord

Description	This method selects the *n*th word on the specified page. The scope embraces texts in field objects, button captions, and annotations.
Parameters	`nPage` — (optional) The 0-based index of the page. The default is `0`. `nWord` — (optional) The 0-based index of the word. The default is `0`. `bScroll` — (optional) Set this parameter to `true` (default) to scroll to the selected word automatically.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which selects the 19th word on the second page. `this.selectPageNthWord(1, 20);`

Related concepts
getPageNthWord
getPageNumWords

setAction

Description	This method sets a document specific script to run when the trigger activates.
Parameters	`cTrigger` — Specify here to what trigger to attach the action: • `WillClose` — Runs the action prior closing the document. • `WillSave` — Runs the action prior to saving the document. • `DidSave` — Runs the action after saving the document. • `WillPrint` — Runs the action prior to printing the document. • `DidPrint` — Runs the action after printing the document. `cScript` — The string, containing the JavaScript code to run when the trigger activates.

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to add the script. After you print the document, you will receive a warning message on the console. `this.setAction("DidPrint", "console.println('Do not forget to pick up your printout!')");`

Related concepts
addScript
setPageAction
Field.setAction

setDataObjectContents

Description	Replaces the content of the data object specified by the `cName` parameter, using `oStream` as the source of the new content.
Parameters	`cName` — The name associated with the data object. Data objects created via JavaScript always have a name, which makes possible to reach them in the `dataObjects` property. Embedded objects inserted via the UI do not have a name and cannot be accessed from JavaScript. `oStream` — A `ReadStream` object to provide the new content.

Related concepts
dataObjects
createDataObject
getDataObject
getDataObjectContents
importDataObject
openDataObject
removeDataObject
util.stringFromStream

setPageAction

Description	This method sets a page-specific script to run when the trigger activates.
Parameters	`nPage` — (optional) The 0-based index of the page. The default is `0`. `cTrigger` — Specify here to what trigger to attach the action: • `Open` — Runs the action when opening the document at the specified page or turning to it. • `Close` — Runs the action when closing the document or turning to another page. `cScript` — The string, containing the JavaScript code to run when the trigger activates.
Example	Run this script to play a default sound each time the user turns to the second page. `this.setPageAction(1, "Open", "app.beep(0);");`

setPageBoxes

Description	This method defines a named box for the specified pages in the rotated user space.
Parameters	`cBox` — The type of the box according to *PDF Reference version 1.7*. Valid values follow: • Art • Bleed • Media • Crop (default) • Trim Type `BBox` is available only in the `getPageBox` method. `nStart` — (optional) The 0-based index of the first page to operate on. Omitting both `nStart` and `nEnd` results in the range of the whole document. Specifying only `nStart` (and omitting `nEnd`) results in a range consisted of a single page marked by `nStart`. `nEnd` — (optional) The 0-based index of the last page of the range. If only `nEnd` specified (and `nStart` is omitted), then the range is from the first page to `nEnd`. `rBox` — (optional) An array, containing the coordinates of the box in the rotated user space.

Related concepts
addField
getPageBox
setPageBoxes
Field.rect

setPageRotations

Description	Sets the rotation angle of the specified page range.
Parameters	`nStart` — (optional) The 0-based index of the first page to operate on. Omitting both `nStart` and `nEnd` results in the range of the whole document. Specifying only `nStart` (and omitting `nEnd`) results in a range consisted of a single page marked by `nStart`. `nEnd` — (optional) The 0-based index of the last page of the range. If only `nEnd` specified (and `nStart` is omitted), then the range is from the first page to `nEnd`. `nRotate` — (optional) The rotation angle in degrees to apply on the specified page range. Valid values are `0`, `90`, `180`, or `270`, the default is `0`.
Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which rotates pages 2 and 3 to 90 degrees. `this.setPageRotations(1, 2, 90);`

Related concepts
getPageRotation

submitForm

Description	Submits the form to the specified URL. Supports HTTPS for secure connection.

Parameters	`cURL` — A full or relative URL, which may has a query string at the end. `bFDF` — (optional) If this parameter is `true` (default), then Power PDF sends the form data in `FDF` format. Otherwise, the submit works with URL-encoded HTML. ℹ This parameter is deprecated, please use `cSubmitAs` instead. `bXML` — (optional) If this parameter is `true` (default), then Power PDF sends the form data in `XML` format. ℹ This parameter is deprecated, please use `cSubmitAs` instead. `bEmpty` — Set this parameter to `true` to submit all fields, including the ones with no value. If the `cSubmitAs` parameter specifies `XDP`, `XML`, or `XFD` format, than fields with no value are also transferred, regardless of the `bEmpty` parameter. In case of other formats empty fields are omitted if `bEmpty` is set to `false` (default). `aFields` — (optional) This array contains the names of the fields to submit. You may also add non-terminal fields to the array. If this parameter is omitted or has a null value, then the method submits all field in the document. If the `cSubmitAs` parameter specifies `XDP`, `XML`, or `XFD` format, then all fields are transferred, regardless of the fields specified in the `aFields` parameter. `bGet` — Set this parameter to `true` to use the HTTP GET method. Set it to `false` (default) to use the HTTP POST method. `bAnnotations` — (optional) Set this to false to prevent annotations to be submitted. The default value is `false`. This parameter is applicable only if `FDF` or `SFDF` is specified in the `cSubmitAs` parameter. `bIncrChanges` — (optional) Set this parameter to `true` to include the incremental changes of the PDF document with the submitted `FDF` file. Only applicable if `FDF` format specified (see `cSubmitAs` for details). `bPDF` — (optional) Set this parameter to `true` to submit the whole PDF document. ℹ This parameter is deprecated, please use `cSubmitAs` instead. `bCanonical` — (optional) Set this to `true` to convert all dates to standard format (`D:YYYYMMDDHHmmSSOHH'mm'`). The default is `false`. For details refer to the *PDF Reference version 1.7*. `bExclNonUserAnnots` — (optional) Set this to `true` to exclude annotations added by others than the current user. The default is `false`. `cPassword` — (optional) Specify a string with the password here, required for the encryption key generation. You may pass over the boolean value `true` to use the password provided earlier within the session. (In this case, make sure not to use quotation marks.) Power PDF prompts the user for the password if needed. ℹ This parameter is applicable only if the `FDF` format is specified in the `cSubmitAs` parameter `bEmbedForm` — (optional) Set this `true` to embed the whole form. ℹ This parameter is applicable only if the `FDF` format is specified in the `cSubmitAs` parameter `cJavaScript` — (optional) Specify `Before`, `After` and `Doc` scripts here, passing over an `oJavaScript` object. ℹ This parameter is applicable only if the `FDF` format is specified in the `cSubmitAs` parameter

Example	This script sends the form data in the default format. `this.submitForm("http://www.yourserver.com/cgi-bin/myscript.cgi#FDF");`

Related concepts
docID
resetForm

Event

JavaScripts are always executed by a certain event. The currently running script can access the `event` object at any time, which is representing the context of the script regarding the situation.

Events have a unique `type` and `name` property combination. You may check these two properties with your script to recognize the situation. Some event properties are available only for certain type of events.

The `rc` property has a special role, it serves as a return code. Check the description of the event to see, if it is listening to the return code or not.

Event type/name combinations

App/Init
The Application Initialization event occurs when Power PDF started. This event does not process the `rc` code.
Batch/Exec
This event occurs every time a document gets processed in a batch sequence. The `target` property of this event links to the `Doc` object. This event processes the `rc` return code: if the `rc` property is `false`, then interrupts the batch sequence.
Bookmark/Mouse Up
This event occurs if a mouse click on a bookmark starts a script. The `target` property of this event links to the `Bookmark` object clicked. This event does not process the `rc` code.
Console/Exec
This event occurs if the user runs a script in the JavaScript Console window. This event does not process the `rc` code.
Doc/DidPrint
This event occurs after a document was printed. The `target` property of this event links to the `Doc` object. This event does not process the `rc` code.
Doc/DidSave

This event occurs after a document was saved. The `target` property of this event links to the `Doc` object. This event does not process the `rc` code.
Doc/Open
This event occurs when a document is opened. The `target` property of this event links to the `Doc` object. This event also defines a `targetName` property. This event does not process the `rc` code.
Doc/WillClose
This event occurs prior a document is closed. The `target` property of this event links to the `Doc` object. This event does not process the `rc` code.
Doc/WillPrint
This event occurs prior document printing. The `target` property of this event links to the `Doc` object. This event does not process the `rc` code.
Doc/WillSave
This event occurs prior a document is saved. The `target` property of this event links to the `Doc` object. This event does not process the `rc` code.
External/Exec
This event occurs when Power PDF works by external access, such as OLE. This event does not process the `rc` code.
Field/Blur
This event occurs when a field loses focus, either by the user clicked away the field or pressed the Tab key. The `target` property of this event links to the field under validation. This event also defines the `modifier`, `shift`, `targetName` and `value` properties. This event does not process the `rc` code.
Field/Calculate
This event occurs when a field recalculates its value. When a field value changes, then each field calculating with that value should refresh, while respecting the calculation order. The target property of this event links to the field object under calculation. This event also defines the `source` and `targetName` properties. This event processes the `rc` return code: if the `rc` property is `false`, then the new field value will not be stored.
Field/Focus

This event occurs between the `Mouse Down` and `Mouse Up` events. Useful for processing and validation that should happen prior the user interacts with the field.

The `target` property of this event links to the field under validation.

This event also defines the `modifier`, `shift` and `targetName` properties.

This event does not process the `rc` code.

Field/Format

This event occurs after all calculations finished. Allows using a formatting script to change the appearance of the field. For example, a script may represent the value as a currency, with a dollar sign and two decimals only.

For text fields you may edit the Format Script in the Format tab of the Text Field Properties dialog box.

The `target` property of this event is linked to the field with the formatting script currently running.

This event also defines the `commitKey`, `targetName` and `willCommit` properties.

This event does not process the `rc` code. The value of the field is used as the formatted appearance.

Field/Keystroke

This event occurs if:

- The user types while the text field or combo box field is in focus.
- The user cuts and pastes text while the text field or combo box is in focus.
- The user uses the keyboard and selects an item in a list box or combo box.
- Prior to the validate event this event is called to allow a final check on the value or format. During the pre-validation run the `willCommit` property is always set to `true`, indicating that the script is now running the last time before commit, and the user finished the input.
- When validating default field values, or values provided by the autofill feature. The target property will be `undefined` in these cases.

In case of list boxes and combo boxes you may edit the Selection Change script in the selection in the field properties dialog box. The Selection Change script added to a list box receives the export value of the selected item in the `changeEx` property. The Selection Change script added to a combo box receives the export value only if a listed item was selected.

The `target` property of this event is linked to the field with the keystroke script currently running. This event also defines the `commitKey`, `change`, `changeEX`, `keyDonw`, `modifier`, `selEnd`, `selStart`, `shift`, `targetName`, `value` and `willCommit` properties.

This event processes the `rc` return code: if the `rc` property is `false`, then the keystroke is ignored. The script may alter the `change` property and this way replace the keystroke. The script may alter also the `selStart` and `selEnd` properties to define a new selection.

Field/Mouse Down

This event occurs as the user pushes the mouse button, which is still down and not released yet. The event is always preceded by a `Mouse Enter` event. There is only a short time available between `Mouse Down` and `Mouse Up`, what is limiting the script in terms of runtime.

The `target` property of this event links to the field under validation.

This event also defines the `modifier`, `shift` and `targetName` properties.

This event does not process the `rc` code.

Field/Mouse Enter

This event occurs as the user moves the mouse pointer inside the field rectangle. You may use this event to display help texts for the field. The `target` property of this event links to the field under validation. This event also defines the `modifier`, `shift` and `targetName` properties. This event does not process the `rc` code.
Field/Mouse Exit
This event occurs as the user moves the mouse pointer outside the field rectangle. The event is always preceded by a mouse-enter event. The `target` property of this event links to the field under validation. This event also defines the `modifier`, `shift` and `targetName` properties. This event does not process the `rc` code.
Field/Mouse Up
This event occurs after the user clicks on the field, and then releases the mouse button. The event is always preceded by a Mouse Down event. Use this event to launch typical processing actions, such as submitting a form. The `target` property of this event links to the field under validation. This event also defines the `modifier`, `shift` and `targetName` properties. This event does not process the `rc` code.
Field/Validate
Validation takes place right after committing the field value, which means the user clicked outside of the field rectangle, tabbed away or pressed the Enter key. This is the first event raised after commit to allow the JavaScript to verify it. If successful, then the `calculate` event is triggered next. The `target` property of this event links to the field under validation. This event also defines the `change`, `changeEX`, `keyDown`, `modifier`, `shift` and `targetName` properties. This event processes the `rc` return code: if the `rc` property is `false`, then the field value is considered invalid, and remains unchanged.
Link/Mouse Up
When a link containing a JavaScript action is activated by the user, then this event occurs. The `target` property of this event links to the `Doc` object. This event does not process the `rc` code.
Page/Open
When a new page displays to the user and the page drawing is completed, then this event occurs. The `target` property of this event links to the `Doc` object. This event does not process the `rc` code.
Page/Close
When the user switched away from the current page or closed it, then this event occurs. The `target` property of this event links to the `Doc` object. This event does not process the `rc` code.

Form processing order

Form processing connects with mouse and keyboard handling closely and works according to the following:

- The field receives focus either by mouse or keyboard actions.
- The `Mouse Up` event may lead the focus away: the user may hold down the mouse button over the field

  rectangle (`Field/Focus` occurs), but still may move away the mouse pointer before releasing the mouse

  button.

  • The field processes (a series of) mouse and keyboard actions by `Keystroke` and `Selection Change`

  events. All changes are passed in the `change` and `changeEX` properties to the events.

  • The `Validate` event occurs right after committing the field value.

  • The `Calculate` event follows on a successful validation (if the field is dependant on other fields).

  • The `Field/Format` event raises.

  • Finally the `Field/Blur` event occurs when the field loses focus either by mouse or keyboard actions.

Event properties

change

Description	This property holds the last keystroke or the last text pasted. You may hook a JavaScript code to the `Keystroke` event of a field to intercept and change the text before it displays in the field.
Type	String
Access	Read/Write
Example	Create a text field, then put this script into the Keystroke Script box on the Format page in the Text Field Properties dialog box. (Select the Custom option to enable the Keystroke Script text field, then click Edit.) This code runs each time a key is pressed when the text field is in focus and translates all letters to uppercase. Also converts pasted contents. `event.change = event.change.toUpperCase();`

changeEx

Description	This property works with list box and combo box fields only. Contains the export value of the list item selected last time (during the `Field/Keystroke` event). Does not work when typing in custom values into a combo box.
Type	Various
Access	Read

Example	Add the `browseHelp()` function to the document, using the Document JavaScript menu.
	`function browseHelp() {`
	`  if ( !event.willCommit && (event.changeEx != "") )`
	`  app.launchURL(event.changeEx);`
	`}`
	Create a list box (`ListBox1`), then run the following code to populate it:
	`var oLB = getField("ListBox1");`
	`oLB.setItems([`
	`  ["Document Automation & Security",`
	`  "https://docshield.tungstenautomation.com/print-capture.html"`
	`format="html" scope="external"],`
	`  ["Workflow Automation",`
	`  "https://docshield.tungstenautomation.com/ia-platform.html"],`
	`  ["Invoice Automation",`
	`  "https://docshield.tungstenautomation.com/fpa.html"],`
	`]);`
	Finally, run this code to hook the `browseHelp()` function to the `Keystroke` action:
	`oLB.setAction("Keystroke", "browseHelp()");`
	Now as the user selects any items in the list, the corresponding online help loads into the default browser application (after confirmation).

commitKey

Description	Indicates how the field editing closed when the field lost focus.
Values	`0` — The user aborted the field edit, so no value was committed. Perhaps the user left the field by pressing the Esc key. `1` — The user altered the field value, then clicked outside of the field, so the value was committed. `2` — The user altered the field value, then pressed the Enter key, so the value was committed. `3` — The user altered the field value, then pressed the Tab key, so the value was committed.
Type	Number
Access	Read

Example	Create a text field, then put this script into the Keystroke Script box on the Format page in the Text Field Properties dialog box. (Select the Custom option to enable the Keystroke Script text field, then click Edit.) This code runs each time the text field loses focus and displays an alert if the field value was edited.
	`if (event.commitKey != 0)`
	`  app.alert("The field was edited.");`

keyDown

Description	This property works with list box and combo box fields only. Contains `true` if the user used the arrow keys to select an item (during the `Field/Keystroke` event). Does not work when typing in custom values into a combo box.
Type	Boolean
Access	Read

modifier

Description	It is set to `true` if the Ctrl modifier key was held down during the event.
Type	Boolean
Access	Read

name

Description	The name of the current event. Use in combination with the `type` property to identify an event.

Values	`Blur`
	`Calculate Init`
	`DidPrint`
	`Focus`
	`Format DidSave`
	`Keystroke`
	`Mouse Down Open`
	`Mouse Enter Close`
	`Mouse Exit`
	`Mouse Up Exec`
	`Validate`
	`WillPrint`
	`WillSave`

Type	String
Access	Read

Related concepts

type

rc

Description	This property determines if the event should succeed or not. Set to `false` to block the event chain and prevent a commit.
Type	Boolean
Access	Read/Write
Events	Keystroke, Validate, Menu

selEnd

Description	This property holds the ending position of the current selection during a `Keystroke` event.
Type	Integer

Access	Read/Write
Events	Keystroke

Related concepts
selStart

selStart

Description	This property holds the starting position of the current selection during a `Keystroke` event.
Type	Integer
Access	Read/Write
Events	Keystroke

Related concepts
selEnd

source

Description	This property represents the `Field` object which triggered the calculation event. Most of the time this is not the target object, which is the field under calculation.
Type	Object
Access	Read

Related concepts
target

shift

Description	It is set to `true` if the Shift modifier key was held down during the event.
Type	Boolean
Access	Read

Example	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button. If you held the Shift key while clicking, then the script displays a message on the console.
	`if (event.shift)`
	`  console.println("Shifted!");`

target

Description	This property represents the `Field` object which triggered the event.
Type	Object
Access	Read

Related concepts
Field

targetName

Description	Returns the name of the JavaScript currently running. This can be useful when searching for the origins of errors and exceptions. (Exceptions report `targetName` if applicable.) The return value may represent various entities, depending on the context: • `App/Init` events — The folder-level script file name. • `Doc/Open` events — The document-level script name. • `Batch/Exec` events — The name of the PDF file processed. • `Field events` — The field name. • `Menu/Exec` events — The menu item name. • `Screen` events — The screen annotation name.
Type	String
Access	Read
Example	Run this code in various contexts (such as `Mouse Up` or Document Javascript actions) to see the context reported on the console. `console.println("Running in: " + event.targetName);`

Related concepts
Event

type

Description	The type of the current event. Use in combination with the `name` property to identify an event.

Values	`Batch`
	`Console`
	`App`
	`Doc`
	`Page`
	`External`
	`Bookmark`
	`Link`
	`Field`
	`Menu`

Type	String
Access	Read

Related concepts
name

value

Description	This property may return various entities, depending on the event context:: • `Field/Validate` events — The value of the field as it is committed. For combo boxes and list boxes represents the face value, not the export value. • `Field/Calculate` events — The calculated value of the field. • `Field/Format` events — The field value as to display. For combo box fields this means the face value. For details on using the export value see the `changeEx` method. • `Field/Keystroke` events — The current value of the field, before the actual keystroke gets applied. • `Field/Blur` and `Field/Focus` events — The current value of the field, as read-only. ℹ If multiple items are selected, then `event.value` returns an empty string and does not accept settings. See `Field.multipleSelection` for details.
Type	Various
Access	Read/Write

Example	Create a text field, then put this script to the Keystroke Script box in its Text Field Properties dialog box. This code runs each time a key pressed or the text field loses focus and displays an alert if the field value is not in the specified range.
	`if (event.value < 0 || event.value > 20) {`
	`  app.alert("Keep the value of the " + event.target.name + "field between 0 and 20" );`
	`  event.rc = false;`
	`}`

Related concepts
changeEx
Field.multipleSelection

willCommit

Description	If this property is `true` then the value is right before commit and the event was called to handle the last value checking prior commit. If this value is `false` then the event was called to handle a keystroke-level checking.
Type	Boolean
Access	Read

Example	This code example is without impact, demonstrating only the typical structure of `Keystroke` events.
	`var value = event.value`
	`if (event.willCommit)`
	`  // Place here the code checking the final value (committing).`
	`else`
	`  // Place here the code checking keystrokes (editing).`

Field

This object represents a form field in the PDF document. Fields can be created either by the UI, using the tools in the Form Elements group of the Forms ribbon, or by JavaScript, using the `Doc.addField` method. To use a field by JavaScript, first you need to assign it to a variable with the `Doc.getField` method:

```
var oTB = this.getField("Text1");
```

The graphical presentation of the field is called widget. The same field may have multiple widgets, holding the same value, but displaying it on different pages, positions, in various style (color, font, etc.). If the user creates a field, then Power PDF names it automatically (such as, `Text2`). If there is a `Text1` field already existing, and the user renames `Text2` to `Text1`, then the second field will share on the same value. You may set visual properties (such as text or fill color, border) of the widgets uniquely using the properties dialog box, but not by JavaScript. The `getField` method cannot address widgets, only the terminal fields, therefore changing properties reflects on all widgets belonging to the field.

You may build a field hierarchy by using the period (".") separator between parent and child field names. For example, create a field named `Name.First`, then create another one named `Name.Last`. Both `Name.First` and `Name.Last` are the children of `Name`, which is an internal field (not visible), created automatically. `Name.First` and `Name.Last` are terminal fields, which are visible on the screen and have separate values. Setting properties of the field `Name` may change both children or may not work. (Setting `fillColor` for `Name` turns the fill color of both children, but setting `Name.value` does nothing.)

Field properties

Some properties are stored as names, others are represented by strings in the PDF document. A name property can have 127 characters at most. For further details, refer to *PDF Reference version 1.7*.

alignment

Description	Determines where to snap the text in a field.

Values	`left`
	`center`
	`right`

Type	String
Access	Read/Write
Fields	text

Example	Create a text field (Text1) with its Alignment option set to Left. Run this code to change the text alignment to center.
	`var oTB = this.getField("Text1");`
	`oTB.alignment = "center";`

borderStyle

Description	Determines what line style to use for drawing the border rectangle around the `Field` object. Use the following values, or see the table below for `border` object constants.

Values	`solid`
	`dashed`
	`beveled`
	`inset`
	`underline`

Type	String
Access	Read/Write
Fields	All

Example	Create a button (Button1) with solid, black and thick border, then run this code to set the border style dashed:
	`var oBT = this.getField("Button1");`
	`oBT.borderStyle = border.d;`

border object constants

Value	Constant	Description
solid	`border.s`	Uses a solid line.
beveled	`border.b`	An additional beveled border runs inside the solid border, resulting in a pushed-out look.
dashed	`border.d`	Uses a dashed line.
inset	`border.i`	An additional inset border runs inside the solid border, resulting in a pushed-in look.
underline	`border.u`	Draws a line under the field.

buttonAlignX

Description	Determines (in percents) how far the icon positioned to the left of the button.
Values	0-100 (the default is 50)
Type	Integer
Access	Read/Write
Fields	button

buttonAlignY

Description	Determines (in percents) how far the icon positioned to the bottom of the button.

Values	0-100 (the default is 50)
Type	Integer
Access	Read/Write
Fields	button

Example	Prepare a very tall button (Button2) with an icon. Add the following `MoveIcon` method as a document JavaScript:
	`function MoveIcon()`
	`{`
	`if ( oBT.buttonAlignY == 0 ) {`
	`  oBT.buttonAlignY++;`
	`  oTI.dir = true;`
	`  return;`
	`}`
	`if (oBT.buttonAlignY == 100 ) {`
	`  oBT.buttonAlignY--;`
	`  oTI.dir = false;`
	`  return;`
	`}`
	`if (oTI.dir)`
	`  oBT.buttonAlignY++;`
	`else`
	`  oBT.buttonAlignY--;`
	`}`
	Run the following script to make the icon moving like an elevator. The script uses the above `MoveIcon` script and the timer of the app object.
	`var oBT = this.getField("Button2");`
	`oBT.buttonAlignY = 0;`
	`oTI = app.setInterval("MoveIcon()", 100);`
	`oTI.dir = true;`
	`oTIover = app.setTimeOut("app.clearInterval(oTI);`
	` app.clearTimeOut(oTIover)", 2*20000+100);`

Related concepts
app.clearInterval
app.clearTimeOut
app.setInterval
app.setTimeOut

buttonFitBounds

Description	Set this property to true to scale the icon to the bounds of the button face. Other icon-setting properties also have their effect on the display.
Values	`true` or `false`
Type	Boolean
Access	Read/Write
Fields	button

Related concepts
buttonAlignX

buttonAlignY
buttonFitBounds
buttonPosition
buttonScaleHow
buttonScaleWhen

buttonPosition

Description	Determines how the icon and the text share layout on the button face. The `position` object provides the constant properties for valid values, see the table below for layout descriptions.

Values	`position.textOnly`
	`position.iconOnly`
	`position.iconTextV`
	`position.textIconV`
	`position.iconTextH`
	`position.textIconH`
	`position.overlay`

Type	Integer
Access	Read/Write
Fields	button

position object constants

Constant property	Description
`position.textOnly`	Text Only
`position.iconOnly`	Icon Only
`position.iconTextV`	Icon top, Text bottom
`position.textIconV`	Text top, Icon bottom
`position.iconTextH`	Icon left, Text right
`position.textIconH`	Text left, Icon right
`position.overlay`	Text in Icon (overlaid)

buttonScaleHow

Description	Determines how the icon is scaled to fit on the button face. The `scaleHow` object provides the constant properties for valid values, see the table below for layout descriptions.
Values	`scaleHow.proportional` `scaleHow.anamorphic`
Type	Integer
Access	Read/Write

Fields	button

scaleHow object constants

Constant property	Description
scaleHow.proportional	Proportionally, keeping the ratio of sides.
scaleHow.anamorphic	Non-proportionally, scaling vertically and horizontally is not bound together.

buttonScaleWhen

Description	Determines in what cases the icon is scaled to fit on the button face. The `scaleWhen` object provides the constant properties for valid values, see the table below for descriptions.

Values	`scaleWhen.always`
	`scaleWhen.never`
	`scaleWhen.tooBig`
	`scaleWhen.tooSmall`

Type	Integer
Access	Read/Write
Fields	button

scaleWhen object constants

Constant property	Description
`scaleWhen.always`	Always scale to fit.
`scaleWhen.never`	Never scale to fit.
`scaleWhen.tooBig`	Scale if the icon is oversized.
`scaleWhen.tooSmall`	Scale if the icon is too small.

calcOrderIndex

Description	Controls the computing order of the calculated numeric fields in the document. Computing starts with the field with the lowest calcOrderIndex and progresses in order.
Type	Integer
Access	Read/Write
Fields	combobox, text

Example	Add two text fields to the document, first Text1, then Text2. Set both fields as numeric, calculated fields. This script changes the original calculation order of the fields by altering the `calcOrderIndex` property.
	`var oTBnew = this.getField("Text2");`
	`var oTBold = this.getField("Text1");`
	`console.println("Originally, Text1: " + oTBold.calcOrderIndex);`
	`console.println("Originally, Text2: " + oTBnew.calcOrderIndex);`
	`oTBnew.calcOrderIndex = oTBold.calcOrderIndex + 1;`
	`console.println("After, Text1: " + oTBold.calcOrderIndex);`
	`console.println("After, Text2: " + oTBnew.calcOrderIndex);`

charLimit

Description	Set this property to limit the maximum number of characters to type into the field.
Type	Integer
Access	Read/Write
Fields	text

Example	Create a text field (Text1), then limit the maximum number of characters by setting the `charLimit` property.
	`var oTB = this.getField("Text1");`
	`oTB.charLimit = 10;`

comb

Description	Set this property true to draw each character in a separate box. Prior this, set the `charLimit` property to specify the number of characters/boxes. Be aware, that setting this property also sets the `doNotScroll` property.
Values	`true` or `false`
Type	Boolean
Access	Read/Write
Fields	text

Example	This script adds a text field with input length limited to 8 characters, then turns it to a comb box.
	`var ip = 72; // point/inch rate`
	`var sRect = this.getPageBox( {nPage: 0} );`
	`//use sRect to calculate the size and position of the text field`
	`sRect[0] += 0.5*ip;// half inch from the from upper left corner of page`
	`sRect[1] -= 0.5*ip;`
	`sRect[2] = sRect[0]+3*ip;// 3 inch width`
	`sRect[3] = sRect[1] - 24;// 24 points in height`
	`var oTB = this.addField("Comb text", "text", 0, sRect);`
	`oTB.strokeColor = color.gray;`
	`oTB.textColor = color.blue;`
	`oTB.fillColor = ["RGB",1,0.66,0.75];`
	`oTB.charLimit = 8;`
	`oTB.comb = true;`

Related concepts
charLimit
doNotScroll

commitOnSelChange

Description	Set this property to `true` to commit the field value immediately as the selection changes. Set this property `false` to delay the commit until the field loses focus, so the user may apply multiple selections without committing the field value multiple times.
Values	`true` or `false`
Type	Boolean
Access	Read/Write
Fields	combobox, listbox

currentValueIndices

Description	When reading, this property returns with the selection. If there is only a single item selected, then returns with an integer value, which holds the 0-based index of the selected item in the `options` array. If there are multiple items selected, then returns an array of indexes sorted in ascending order. If the current value became invalid due to random user editing, then it returns `-1`. When writing, this property may receive an integer on an array of integers. Pass an integer to set a single item selected, or pass an array of integers to set multiple items selected. In an editable combobox, you may set the `value` property to an arbitrary string, which is not among the list items. Set the fields `multipleSelection` property to control the ability of multiple selections.
Type	Integer or Array
Access	Read/Write
Fields	combobox, listbox

Example	Create a list box (ListBox1) and populate it. Run the script below to turn the top item selected.
	`var oLB = this.getField("ListBox1");`
	`oLB.currentValueIndices = 0;`
	Create a list (ListBox1), populate it, then select one or multiple elements. The following script sends the selected item(s) to the console.
	`var oLB = this.getField("ListBox1");`
	`var myselection = oLB.currentValueIndices;`
	`if (typeof myselection == "number") // A single item is selected`
	`  console.println("Selection: " + oLB.getItemAt(myselection, false));`
	`else // Multiple items are selected`
	`{  console.println("Selection:");`
	`  for (var i = 0; i < myselection.length; i ++)`
	`  console.println(" " + oLB.getItemAt(myselection[i], false)); }`

Related concepts
deleteItemAt
getItemAt
insertItemAt
multipleSelection
numItems
setItems

defaultValue

Description	The default value of the field, which loads at form reset. You may use either the user value or the export value as a default. If there are identical export and user values for different list items, then the first export value is used.
Type	String
Access	Read/Write
Fields	all except `button` and `signature`

Example	Create a text field (Text1), then run the following code to set its default value.
	`var oTB = this.getField("Text1");`
	`oTB.defaultValue = "Enter first name here.";`
	`resetForm(["Text1"]);`

doNotScroll

Description	Set this property to `true` to disable scrolling during text editing. This constrains the text editing to the area of the field and limits the number of characters.
Type	Boolean
Access	Read/Write
Fields	text

delay

Description	Redrawing of field objects is automatic and comes with property changes as necessary. In case of a massive amount of JavaScript property change requests, Power PDF performs better if you delay the redraw. Set this property to `true` to disable automatic redraw until `delay` property is set back to `false`.
Type	Boolean
Access	Read/Write
Fields	All

Example	Change the look of CheckBox1. Set the delay to true and make all changes, then set the delay property back to false to let the display update.
	`var oCB = this.getField("CheckBox1");`
	`// Blocks display update`
	`oCB.delay = true;`
	`oCB.borderStyle = border.d;`
	`// You may insert a number of other property changes here`
	`oCB.strokeWidth = 5;`
	`// Allows updating the field display with the changes now`
	`oCB.delay = false;`

display

Description	Controls if the field is displayed or printable. This property is recommended over the `hidden` and `print` properties.
Values	See the table below for `display` object constants.
Type	Integer
Access	Read/Write
Fields	All

Example	Set the display property of Text1, then check, if it is visible but not printable.
	`// Set the display property`
	`var oTB = getField("Text1");`
	`oTB.display = display.noPrint;`
	`// Test whether the field gets printed`
	`if (oTB.display == display.noPrint)`
	`  console.println("This field is visible but does not get printed.");`

display object constants

Constant	Description
`display.visible`	The field is visible on screen and printable.
`display.hidden`	The field is hidden on screen and not printable.
`display.noPrint`	The field is visible on screen but not printable.

Constant	Description
`display.noView`	The field is hidden on screen but printable..

Related concepts
hidden
print

doc

Description	Specifies the `Doc` object of the document where the field is placed.
Type	`Doc` object
Access	Read
Fields	All

editable

Description	If this property is set to `true`, then user may type in a selection. If this property is `false`, then the user must select from the provided items.
Type	Boolean
Access	Read/Write
Fields	combobox

Example	Turns ComboBox1 to editable.
	`var oCB = this.getField("ComboBox1");`
	`oCB.editable = true;`

exportValues

Description	Export values can bound to `radiobutton` and `checkbox` items. The `exportValues` property is an array of strings, containing the export values mapped to the annotations in the order of creation (tab-order is insignificant in this case). This property is required for `radiobutton` fields. A group of buttons returns the export value of the currently selected `radiobutton` field or returns with "Off" if there is no button selected. `Yes` is the default export value for `checkbox` fields when the field is checked. `Off` is the default when the field is cleared.
Type	Array
Access	Read/Write
Fields	checkbox, radiobutton

Example	This script creates a group of four radio buttons, then assigns the export values.
	`var offs = 24;`
	`var x1=100; y1=600; x2=110; y2=590;`
	`var oRB = this.addField("Radio1","radiobutton",0, [x1, y1, x2, y2]);`
	`var oRB = this.addField("Radio1","radiobutton",0, [x1, y1+offs, x2, y2+offs]);`
	`var oRB = this.addField("Radio1","radiobutton",0, [x1, y1+2*offs, x2, y2+2*offs]);`
	`var oRB = this.addField("Radio1","radiobutton",0, [x1, y1+3*offs, x2, y2+3*offs]);`
	`oRB.exportValues = ["1", "2", "3", "4"];`
	Run this script to send the value of the selected button to the console:
	`var oRB = this.getField("Radio1");`
	`if ( typeof oRB.page == "object" )`
	`  console.println("Selected value:" + oRB.value);`

fillColor

Description	Defines the background color for the field.
Values	Values are defined by using `transparent`, `gray`, `RGB` or `CMYK` color objects.
Type	Array
Access	Read/Write
Fields	All

Example	Alternates the background color of a text field, switching from the default white to red and back. If the current color is white, then it changes to red, otherwise changes back to white.
	`var oTB = this.getField("Text1");`
	`if (color.equal(oTB.fillColor, color.red))`
	`  oTB.fillColor = color.white;`
	`else`
	`  oTB.fillColor = color.red;`
	`// Get the field in focus to display the color change`
	`this.getField("Text1").setFocus();`

hidden

Description	When this property is `true`, the annotation is not displayed, not printable, and the user cannot interact with it. This property is obsolete, use `display` instead.
Type	Boolean
Access	Read/Write
Annotations	All

highlight

Description	Controls how should the button behave when on mouse click. Use the following supported values , or see the table below for `highlight` object constants.
Values	`none` — No visual feedback on button click. `invert` — The button face inverts on click. `push` — The down state of the button shows up for a short time if button down face was defined. `outline` — The border of the button area gets inverted momentarily.
Type	String
Access	Read/Write
Fields	button

Example	This script sets the highlight fashion of Button1 to `outline`.
	`var oBT = this.getField("Button1");`
	`oBT.highlight = highlight.o;`

highlight object constants

Value	Constant
none	`highlight.n`
invert	`highlight.i`
push	`highlight.p`
outline	`highlight.o`

lineWidth

Description	Specifies the border width of the field rectangle. Zero results in no border. Has no affect in case of transparent stroke color, unless with a beveled border.
Values	`0` — none `1` — thin `2` — medium `3` — thick
Type	Integer
Access	Read/Write
Fields	All

Example	This script changes the border thickness on the button to thick.
	`var oBT = this.getField("Button1");`
	`oBT.lineWidth=3;`

multipleSelection

Description	If this property holds `true`, then the user may select multiple items in the list.
Type	Boolean
Access	Read/Write
Fields	listbox

Related concepts
currentValueIndices
type
value

name

Description	Returns the full name of the field as a string.
Type	String
Access	Read
Fields	All

Example	This script reads the full name of Text1 and sends it to the console output.
	`var oTB = this.getField("Text1");`
	`// Displays "Text1" in the console window`
	`console.println(oTB.name);`

numItems

Description	This property returns the number of items in the list.
Type	Integer
Access	Read
Fields	combobox, listbox

Example	Displays the item number of Listbox1 on the console.
	`var oLB = this.getField("ListBox1");`
	`console.println("Number of items in the list: " + oLB.numItems + ".");`

page

Description	If the field has only a single appearance in the document, then the `page` property returns the 0-based page number, on which the field is located. If the field appears on more than one pages, then `page` property returns an array of integers. Each integer represents a 0-based page number on which this field has a widget. The order of numbers is based on the creation order of the individual widgets. The `page` property returns `-1` for widgets placed on a hidden page.

Type	Integer or Array
Access	Read
Fields	All

Example	This script takes the number of widgets associated with the Radio1 field name and sends it to the console. In other words, this example gets the number of radio buttons in the Radio1 radio button field.
	`var oRB = this.getField("Radio1");`
	`if ( typeof oRB.page == "object" )`
	`  console.println("There are " + oRB.page.length + " radio buttons in this field.");`
	This script counts the numbers of widgets connected to Text99.
	`var oTB = this.getField("Text99");`
	`if (typeof oTB.page == "number")`
	`  // There is only one field placed in the document`
	`  // with that name (page holds the page number).`
	`  console.println("Text99 field occurs only once on page " + oTB.page)`
	`else`
	`  // The document contains more than one fields`
	`  // with that name (page is an array of page numbers).`
	`  console.println("Text99 field occurs " + oTB.page.length + " times");`

password

Description	If this property holds `true`, then the field data is not saved with the document, and only asterisks show up when text entered.
Type	Boolean
Access	Read/Write
Fields	text

print

Description	If this property holds `true`, then the field is printed with the document. The `display` property is recommended over the `hidden` and `print` properties.
Type	Boolean
Access	Read/Write
Fields	All

Related concepts

display

required

Description	Determines if the field is required to fill. If `true`, then the field's `value` property should not be `null` when submitting. Clicking on the submit button when there is a required field with a `null` value, results in a warning message and the process fails.
Type	Boolean
Access	Read/Write
Object Type	All except `button`

Example	This script sets the Text1 field as required.
	`var oBT = this.getField("Text1");`
	`oBT.required = true;`

radioInUnison

Description	If this property holds `true`, then radio buttons with the same name and export value turn on and off in unison. This property is available on the user interface, named Buttons with the same name and value are selected in unison on the Options page of the Radio Button Properties dialog box.
Type	Boolean
Access	Read/Write
Fields	All

readOnly

Description	Set this property to `true` to block any user edits and changes on the field.
Type	Boolean
Access	Read/Write
Fields	All

Example	This script sets the Text1 field read-only and loads a warning text into the text field.
	`var oTB = this.getField("Text1");`
	`oTB.value = "You cannot change this text.";`
	`oTB.readonly = true;`

rect

Description	The `rect` array contains four numbers [xll, yll, xur, yur] to define the lower-left x, lower-left y, upper-left x, and upper-right y coordinates in the default user space. This way `rect` represents the rectangle that specifies the area of the field on the page.
Type	Array
Access	Read/Write
Fields	All

Example	This script adds a new button to the first page of the document, placed half an inch from the upper left corner. The button takes one inch in width and 24 points in height.
	`var ip = 72; // point/inch rate`
	`var sRect = this.getPageBox( {nPage: 0} );`
	`sRect[0] += 0.5*ip;// half inch from the from upper left corner of page`
	`sRect[1] -= 0.5*ip;`
	`sRect[2] = sRect[0]+1*ip;// one inch width`
	`sRect[3] = sRect[1] - 24;// 24 points in height`
	`var oBT = this.addField("Button1", "button", 0 , sRect);`

rotation

Description	Defines the number of degrees the widget is rotated counter-clockwise with respect to the page.
Values	`0`, `90`, `180`, `270` degrees
Type	Integer
Access	Read/Write
Fields	All

Example	Create a rotated text field on the first page and fill it with text.
	`var oTB = this.addField("Text1", "text", 0, [16, 16+72, 28, 16]);`
	`oTB.rotation = 270;`
	`oTB.value = "Take care of your neck.";`

strokeColor

Description	Specifies the border color of the rectangle of the field.
Values	Values are defined by using `transparent`, `gray`, `RGB` or `CMYK` color objects.
Type	Array
Access	Read/Write
Fields	All

Example	Change the stroke color of each text field in the document to green.
	`for ( var i=0; i < this.numFields; i++) {`
	`  var fieldname = this.getNthFieldName(i);`
	`  var oField = this.getField(fieldname);`
	`  if (oField.type == "text") oField.strokeColor = color.green;`
	`}`

style

Description	Specifies the glyph style of the markable area of the field. The glyph is the graphical object working as the sensitive area of the widget. Use the following supported values, or see the table below for `style` object constants.

Values	`check`
	`cross`
	`diamond`
	`circle`
	`star`
	`square`

Type	String
Access	Read/Write
Fields	checkbox, radiobutton

Example	This script sets the glyph style of CheckBox1 to circle.
	`var oCB = this.getField("CheckBox1");`
	`oCB.style = style.ci;`

style object constants

Value	Constant
check	`style.ch`
cross	`style.cr`
diamond	`style.di`
circle	`style.ci`
star	`style.st`
square	`style.sq`

textColor

Description	Specifies the color of text and other foreground objects in the field. For example, this includes texts, list items, a caption on a button, or the color of the radio button and check-glyphs. Using `transparent` color space in `textColor` raises an exception.
Values	Values are defined by using `gray`, `RGB` or `CMYK` color objects.
Type	Array
Access	Read/Write
Fields	All

Example	This script sets the foreground color to green for each text field with a value greater than zero.
	`for ( var i=0; i < this.numFields; i++) {`
	`  var fieldname = this.getNthFieldName(i);`
	`  var oField = this.getField(fieldname);`
	`  if (oField.type == "text" && oField.value > 0)`
	`  oField.textColor = color.green;`
	`}`

submitName

Description	Specifies an alternative field name for form submission. Only applicable with HTML format.
Type	String
Access	Read/Write
Annotations	All

type

Description	Specifies the field type, which cannot be modified by setting this property.

Values	`button`
	`checkbox`
	`combobox`
	`listbox`
	`radiobutton`
	`signature`
	`text`

Type	String
Access	Read
Fields	All

Example	This script displays the number of buttons in the document.
	`var num = 0;`
	`for ( var i=0; i<this.numFields; i++) {`
	`  var fieldname = this.getNthFieldName(i);`
	`  if (this.getField(fieldname).type == "button") num++; }`
	`app.alert("I have counted " + num + " buttons.");`

userName

Description	This property serves as a short description string for the field, displaying as a tooltip when entering the field.
Type	String
Access	Read/Write
Fields	All

Example	Add a tooltip to a text field.
	`var oTB = this.getField("Text1");`
	`oTB.userName = "Type your name here.";`

value

Description	The value of the field, as it is entered by the user. Use the `value` property for field calculations. `listbox` objects with multiple selections provide `value` as an array of the selected items. For `listbox` fields it is more preferred to use `currentValueIndices` to get or set selected items. For a group of `radiobutton` objects (with unique export values for all the items) you may check the `value` property, which holds the export value of the selected `radiobutton` widget. The `value` property holds an empty string if no widget selected in the group.
Type	various
Access	Read/Write
Fields	All except `button`

Example	Run this code to display the current date in the Text1 field.
	`var oTB = this.getField("Text1");`
	`oTB.value = util.printd("yyyy/mm/dd",new Date());`

Related concepts
currentValueIndices

valueAsString

Description	The value of the field, converted to `String`.
Type	String
Access	Read
Fields	All except `button`

Example	Run this code to display the current date in the Text1 field.
	`var oTB = this.getField("Text1");`
	`oTB.value = util.printd("yyyy/mm/dd",new Date());`

Field methods

buttonGetIcon

Description	Returns the `Icon` object of an icon associated with the button.
Parameters	`nFace` — (optional) Specifies the caption type to get: • `0` — normal icon (default) • `1` — button down icon • `2` — mouse over icon
Returns	The `Icon` object..

Related concepts
buttonImportIcon
buttonSetIcon

buttonImportIcon

Description	Imports a page as an icon from the specified page of the assigned PDF file. If there is no appropriate PDF file specified in the `cPath` parameter, then the Select icon dialog box shows up, and the user should browse to the file. In this case the user may switch to a supported file format other than PDF, the image files will be converted automatically.
Parameters	`cPath` — (optional) Device-independent path to the source image file. `nPage` — (optional) The 0-based number of the page in the source file to turn into an icon.
Returns	The method returns an integer error code: • `1` — The user cancelled the process by closing the dialog box. • `0` — The icon was imported successfully. • `-1` — The specified or selected file could not be opened. • `-2` — The given page number was invalid.

Example	Add the following script to import an avatar picture as an icon and print the error code to the console:
	`var oBT = this.getField("Button1");`
	`var result = oBT.buttonImportIcon("/C/IDs/avatar.pdf");`
	`console.println(result);`

buttonSetCaption

Description	Sets the caption for the button.
Parameters	`cCaption` — The caption to set for the button. `nFace` — (optional) Specifies the caption type to get: • `0` — normal icon (default) • `1` — button down icon • `2` — mouse over icon
Returns	The button caption as a string.

Example	Create a button (Button1), then run this script to change the caption of it.
	`var oBT = this.getField("Button1");`
	`oBT.buttonSetCaption("Hello World!")`

Related concepts
buttonGetCaption

buttonGetCaption

Description	Returns with the caption text associated with the button.
Parameters	`nFace` — (optional) Specifies the caption type to get: • `0` — normal icon (default) • `1` — button down icon • `2` — mouse over icon
Returns	The button caption as a string.

Example	Create a button with both caption and icon, then go to the Actions tab in its Button properties dialog box. Add the following event handler scripts to turn the button more mouse-sensitive: the scripts will surround the caption with pointing arrows when mouse is over. Add the following script to MouseEnter trigger to run:
	`event.target.buttonSetCaption("=> "+ event.target.buttonGetCaption() +" <=");`
	Add the following script to the MouseExit trigger:
	`var s = event.target.buttonGetCaption();`
	`s = s.replace(/=> | <=/g, "");`
	`event.target.buttonSetCaption(s);`

Related concepts
buttonSetCaption

buttonSetIcon

Description	Adds an icon to the button. Running `buttonSetIcon` in the JavaScript console raises an exception. This method should run in a script embedded within the document.
Parameters	`oIcon` — The `Icon` object to associate with the button. `nFace` — (optional) Specifies the caption type to get: • `0` — normal icon (default) • `1` — button down icon • `2` — mouse over icon

Example	This script interchanges the icons on Button1 and Button2.
	`var oBT1 = this.getField("Button1");`
	`var oBT2 = this.getField("Button2");`
	`var t = oBT1.buttonGetIcon();`
	`oBT1.buttonSetIcon(oBT2.buttonGetIcon());`
	`oBT2.buttonSetIcon(t);`

Related concepts
buttonGetIcon
buttonImportIcon
getIcon

checkThisBox

Description	Selects or clears a check box widget. Applicable only on `checkbox` objects. This method does not work with radiobutton objects. To reset a radiobutton use the `resetForm` method. (Prior that `defaultIsChecked` should be set to `false`.)
Parameters	`nWidget` — The 0-based index of the widget for the field. A field may have multiple widgets, and you may address an individual object by its index. The index goes along the creation order of the widgets, and it is not affected by tab order. `bCheckIt` — (optional) Set this property to `true` (default) to turn the `checkbox` widget selected, or set it to `false` to clear it.

Example	Create a check box (CheckBox1), then run the following script to turn the check box selected:
	`var oCB = this.getField("CheckBox1");`
	`oCB.checkThisBox(0,true);`

Related concepts
defaultIsChecked
getField
resetForm

clearItems

Description	Removes all items from a `listbox` or a `combobox`.
Example	Create a combo box (ComboBox1) with several items. Run the following script to remove all items from the combo box: `this.getField("ComboBox1").clearItems();`

Related concepts
currentValueIndices
deleteItemAt
getField
getItemAt

insertItemAt
numItems
setItems

defaultIsChecked

Description	Set the default state for a `checkbox` or `radiobutton` object with this property.
Parameters	`nWidget` — The 0-based index of the widget for the field. A field may have multiple widgets, and you may address an individual object by its index. The index goes along the creation order of the widgets, and it is not affected by tab order. Each entry in the Fields panel has a suffix with this index (such as `MyField #0`). `bIsDefaultChecked` — (optional) Set this property to `true` (default) to turn the `checkbox` widget selected by default on a field reset, or set it to `false` to clear it.
Returns	`true` on success.

Example	Create a check box (CheckBox1), then run the following script to set the default state for this check box selected. To display the result, the script resets the form, so all fields get restored to default.
	`var c = this.getField("CheckBox1");`
	`c.defaultIsChecked(0,true);`
	`this.resetForm(["CheckBox1"]);`

Related concepts
defaultIsChecked
getField
resetForm

deleteItemAt

Description	Removes the specified item from a combo box or list box. As you delete the selected item, no item will have selected status in the list box. If the script calls the method again, this may cause malfunctioning. Preventing this, good to select an item using the `currentValueIndices` property.
Parameters	`nIdx` — (optional) The 0-based index of the item to remove. The method removes the currently selected item if no parameter specified.

Example	Create a list box (ListBox1) and populate it. Select one of the items, then run the script below. This script removes the selected item from the list and then turns the top item selected.
	`var oLB = this.getField("ListBox1");`
	`oLB.deleteItemAt();`
	`oLB.currentValueIndices = 0;`

Related concepts
currentValueIndices

getArray

Description	Returns with the array of terminal child field objects connected with the current field. (A terminal field is a field, which can have a value.)
Returns	Array

Example	Create three fields: `Value.First`, `Value.Second`, and `Value.Third`. Due to the way they are named, these three fields now are children of the `Value` parent field. The script below sums the children field values in `nSum` and displays the result on the console.
	`var oValue = this.getField("Value");`
	`var children = oValue.getArray();`
	`var nSum = 0.0;`
	`for (i =0; i < children.length; i++)`
	`  nSum += children[i].value;`
	`console.println("Total sum of children fields: " + nSum);`

getItemAt

Description	Returns with the export value or name of the specified list item.
Parameters	`nIdx` — The 0-based index of the item. Set this parameter to `-1` to point to the last item in the list. `bExportValue` — (optional) Determines which value to return with: • `true` — Returns the export value, if the item has any, otherwise returns the item name. • `false` — Returns the name of the item.
Returns	Either the export value or the name of the item.

Example	In this example `n` holds the export value of the first item in the list.
	`// Filling up the list`
	`var oLB = this.getField("ListBox1");`
	`oLB.setItems([["First", 1],["Second", 2], "Third"]);`
	`// Due to zero-based indexing, returns the value of the first list item, which is 1`
	`var n = oLB.getItemAt(0);`
	`console.println(n);`
	In this example the second, optional parameter is also used. By setting it to `false`, the item name is retrieved instead of the export value.
	`var oLB = this.getField("ListBox1");`
	`oLB.setItems([["First", 1],["Second", 2], "Third"]);`
	`for (var i=0; i < oLB.numItems; i++)`
	`  console.println(oLB.getItemAt(i,true) + ": " + oLB.getItemAt(i,false));`
	Console output reads as:
	`1:First`
	`2:Second`
	`Third:Third`

Related concepts

clearItems

currentValueIndices

deleteItemAt
insertItemAt
numItems
setItems

insertItemAt

Description	Inserts a new item into the selected position in a list box or combo box.
Parameters	`nIdx` — The 0-based index of the desired position in the list. You may use special values as follows: • `0` — Inserts the new item at the top of the list • `-1` — Inserts the new item at the end of the list.. `cName` — The item name as appears on the list. `cExport` — (optional) The export value of the item. If not specified, then `cName` is reported instead.

Example	The following script adds an item (`Top line`) to the ListBox1 listbox:
	`var oLB = this.getField("ListBox1");`
	`oLB.insertItemAt("Top line");`
	The following script adds an item (`Bottom line`) to the ListBox1 list box, at the bottom of the list:
	`var oLB = this.getField("ListBox1");`
	`oLB.insertItemAt("Bottom line", "Bottom line", -1);`

Related concepts
clearItems
currentValueIndices
deleteItemAt
getItemAt
numItems
setItems

isBoxChecked

Description	Use this method to see if a `checkbox` widget is selected or not.
Parameters	`nWidget` — The 0-based index of the widget for the field. A field may have multiple widgets, and you may address an individual object by its index. The index goes along the creation order of the widgets, and it is not affected by tab order. Each entry in the Fields panel has a suffix with this index (such as `MyField #0`).
Returns	Returns `true` if the widget is selected, otherwise returns `false`.

Example	This script inspects the state of the CheckBox1 check box and displays the report in an alert box.
	`var oCB = this.getField("CheckBox1");`
	`var status = (oCB.isBoxChecked(0)) ? " " : " not ";`
	`app.alert("The box is" + status + "checked");`

isDefaultChecked

Description	Test this property to see if the widget turns to selected by a `resetForm` operation, or not. For a group of `radiobutton` objects (with unique export values for all the items) you may check the `value` property, which holds the export value of the selected `radiobutton` widget. The `value` property holds an empty string if no widget selected in the group.
Parameters	`nWidget` — The 0-based index of the widget for the field. A field may have multiple widgets, and you may address an individual object by its index. The index goes along the creation order of the widgets, and it is not affected by tab order. Each entry in the Fields panel has a suffix with this index (such as `MyField #0`).
Returns	Returns `true` if the widget is selected by default, otherwise returns `false`.

Example	This script inspects the default state of CheckBox1 and displays the report in an alert box.
	`var oCB = this.getField("CheckBox1");`
	`var dstatus = (oCB.isDefaultChecked(0)) ? "Checked" : "Cleared";`
	`app.alert("The Default: " + dstatus);`

Related concepts
defaultIsChecked
getField
resetForm

setAction

Description	Bind a JavaScript action to a field event trigger. This method overwrite the action previously was bind to the trigger.

Parameters	`cTrigger` — This string parameter specifies the trigger to bind. Valid parameters are: • `MouseUp` • `MouseDown` • `MouseEnter` • `MouseExit` • `OnFocus` • `OnBlur` • `Keystroke` • `Validate` • `Calculate` • `Format` For `listbox` fields bind the `Keystroke` trigger to the Selection Change event. `cScript` — This string contains the JavaScript to execute when the trigger gets activated.

Example	This script adds a button field with addField, sets the look of the button, and adds a sound to the `MouseUp` trigger using `setAction`.
	`var ip = 72; // point/inch rate`
	`var sRect = this.getPageBox( {nPage: 0} );`
	`sRect[0] += 0.5*ip;// half inch from the from upper left corner of page`
	`sRect[1] -= 0.5*ip;`
	`sRect[2] = sRect[0]+1*ip;// one inch width`
	`sRect[3] = sRect[1] - 24;// 24 points in height`
	`var oBT = this.addField("Button1", "button", 0 , sRect);`
	`oBT.setAction("MouseUp", "app.beep(0);");`
	`oBT.delay = true;`
	`oBT.fillColor = color.ltGray;`
	`oBT.buttonSetCaption("Click Alarm!");`
	`oBT.borderStyle = border.b;`
	`oBT.lineWidth = 3;`
	`oBT.strokeColor = color.red;`
	`oBT.highlight = highlight.p;`
	`oBT.delay = false;`

Related concepts
Bookmark.setAction
Doc.setAction
Doc.addScript
Doc.setPageAction

setFocus

Description	Pulls the keyboard focus onto the field, which may include changing page or scroll.

Example	This script brings the Value.First field in focus.
	`this.getField("Value.First").value = "Enter the first value here: "`
	`this.getField("Value.First").setFocus();`

setItems

Description	Use this method to fill up a list in a `combobox` or `listbox` object with items.
Values	`oArray` — An array describing the list items. • If an array element is a string (or convertible), then serves both as item name and export value. • If an array element is an array of strings (or convertible), then the first subelement serves as item name, the second subelement provides the export value.

Example	This script fills the list box with strings, these serve as export values also.
	`var oLB = this.getField("ListBox1");`
	`oLB.setItems(["First", "Second", "Third"]);`
	This script fills the combo box with country names, adding their country codes as export values.
	`var oCB = this.getField("ComboBox1");`
	`oCB.setItems([["United States", "US"],["United Kingdom", "UK"],`
	`["Hungary", "HU"]]);`
	The third item in the following list displayes as LN2, and has the natural logarithm of 2 as export value.
	`var oLB = this.getField("ListBox1");`
	`oLB.setItems(["1", 2, 3, ["LN2", Math.LN2]]);`
	`console.println(oLB.getItemAt(3));`

Related concepts
clearItems
currentValueIndices
deleteItemAt
getItemAt
numItems
setItems

signatureInfo

Description	Returns with a `SignatureInfo` object, which is a snapshot of the signature used on the specified security handler.
Parameters	`oSig` — (optional) The SecurityHandler object with the signature.
Returns	A `SignatureInfo` object with the properties and values copied from the signature.
Example	Check the example provided for the method signatureValidate.

Related concepts
SignatureInfo
SecurityHandler

signatureSign

Description	This method signs the field with the specified security handler. You cannot sign fields already signed, prior that you need to clear the field with the `formReset` method. The security-restricted `signatureSign` method may run only in a privileged context, that means console, batch and application initialization events. Triggers available in the Button Properties dialog box, such as `Mouse Up` or `On Focus` are considered non-privileged. If the certificate of the document is trusted for running embedded high privilege JavaScript, then security-restricted methods may run without restrictions.
Parameters	`oSig` — The `SecurityHandler` object to use. If not specified, then selects a handler based on user preferences, or prompts the user, if the `bUT` parameter is `true`. An exception raises if the handler does not support signing. `oInfo` — (optional) A `signatureInfo` object with properties filled correctly. `cDIPath` — (optional) The device-independent path for saving the file after the operation. If not specified, then the file is written back to its original location. `bUT` — (optional) Set this `true` to show the user interface for user login. In this case the `oInfo` and `cDIPath` parameters are used as default values for login.
Returns	Returns `true` if signing was successful, otherwise returns `false`.

Example	Sign the Signature1 field with the PPKLite signature handler:
	`var oSH = security.getHandler( "Adobe.PPKLite" );`
	`oSH.login( yourpassword, "/c/YourFolder/YourFileName.pfx");`
	`var sf = this.getField("Signature1");`
	`// Sign the field`
	`var result = sf.signatureSign( oSH,`
	`{password: "yourpassword",// provide the same password again`
	`location: "Budapest, HU",`
	`reason: "I am approving this document.",`
	`contactInfo: "johndoe@example.com",`
	`appearance: "Standard"});`
	`console.println("Signing was " + ((result) ? "successful." : "not successful."));`
	You should not necessarily provide the password if Password Timeout not expired yet.

Related concepts

SecurityHandler
securityHandler.setPasswordTimeout
security.getHandler

signatureValidate

Description	Returns the validity status for a signature. The validation process may take a significant amount of time.

Parameters	`oSig` — (optional) A `SecurityHandler` or a `SignatureParameters` object to use (see SignatureParameters object). If not specified, then uses the `handlerName` property of the signature object. `bUT` — (optional) Set this `true` to show the user interface for validation, if needed to select a validation handler (none specified). The default is `false`.
Returns	Returns with a validity status value: `-1` — The specified field is not a signature. `0` — The signature is blank. `1` — The status is unknown. `2` — The signature is invalid. `3` — The document signature is valid, but signer identity cannot be verified. `4` — The document signature is valid, signer identity is valid.

Example	You can check the validity of Signature1 with the following script:
	`var f = this.getField("Signature1")`
	`//Validate returns with validity status code`
	`var status = f.signatureValidate();`
	`//Retrieving signature details`
	`var si = f.signatureInfo();`
	`if ( status < 3 )`
	`  var msg = "This signature is not valid! " + si.statusText;`
	`else`
	`  var msg = "This signature is valid! " + si.statusText;`
	`app.alert(msg);`

SignatureParameters object

Description	The properties of the `SignatureParameters` generic object specify which security handlers are to be used for validation by `signatureValidate`:
Parameters	`oSecHdlr` — The `SecurityHandler` object to use. `bAltSecHdlr` — Set this `true` to select an alternative security handler based on user preferences. The default is `false`, which means that the `handlerName` property of the signature is referred for a handler. This parameter is ignored if `oSecHdlr` is provided.

Related concepts
security.getHandler
SecurityHandler
SigInfo.handlerName

FullScreen

This object grants access to the properties of the application when running in presentation mode.

FullScreen properties

isFullScreen

Description	Determines if the application runs in full screen (presentation) mode. Set this property to `true` to switch to full-screen view (this requires to have at least one document opened).
Type	Boolean
Access	Read/Write
Example	This script turns the application to presentation mode. `app.fs.isFullScreen = true;`

Related concepts
FullScreen object
app.fs

global

This constant object offers persistent properties to store and share variables across opened documents.

To create a new property just assign a value to it. The following script creates a counter for the filled forms:

```
global.countFormsFilled = 0;
```

See the method `setPersistent` for details how to make the global variable persistent across Power PDF sessions.

global methods

setPersistent

Description	Sets a global variable persistent. That means, the value of that variable is stored in the glob.js file located the JavaScript user folder. Therefore the value of the variable will not be volatile any more and can be reached by other PDF files. To avoid collision use a naming convention with well-setup prefixes. (for example, *companyname_variablename*)
Parameters	`cVariable` — The variable name (without the `global.` prefix) to turn persistent.

Example	This code creates a global variable to store the number of filled forms across PDF documents.
	`global.countFormsFilled = 0;`
	`global.setPersistent("countFormsFilled");`

RDN

RDN properties

This generic object represents a *Relative Distinguished Name* in the `securityHangler.newUser` and the `certificate.subjectDN` properties.

Related concepts

SecurityHandler.newUser
certificate.subjectDN

c

Description	The country or region, according to the *ISO 3166* standard. For example, `HU` for Hungary of Europe.
Type	String
Access	Read

cn

Description	The common name of the person. For example, `John Doe`.
Type	String
Access	Read

e

Description	The email address. For example, `john.doe@example.com`.
Type	String
Access	Read

o

Description	The name of the organization. For example, `Mozilla Foundation`.
Type	String
Access	Read

ou

Description	The name of the organizational unit. For example, `Document Imaging`.
Type	String

Access	Read

search

This static object offers search capabilities. You may start a search query using JavaScipt, the results will display in the Search window.

search properties

attachments

Description	Set this property to true to extend the scope of the search operations to PDF attachments.
Type	Boolean
Access	Read/Write

Example	The following script list all the search object properties to the console.
	`console.println("attachments: " + search.attachments);`
	`console.println("available: " + search.available);`
	`console.println("name: " + search.name);`
	`console.println("path: " + search.path);`
	`console.println("bookmarks: " + search.bookmarks);`
	`console.println("markup: " + search.markup);`
	`console.println("matchCase: " + search.matchCase);`
	`console.println("matchWholeWord: " + search.matchWholeWord);`
	`console.println("maxDocs: " + search.maxDocs);`
	`console.println("stem: " + search.stem);`
	`console.println("wordMatching: " + search.wordMatching);`

available

Description	Check this property before initiating any search operation. This property returns `true` if the search plug-in is loaded and ready for a query.
Type	Boolean
Access	Read
Example	Check the example provided with the `query` method.

Related concepts

query

bookmarks

Description	Set this property to `true` to extend the scope of the search operations to bookmarks. The default is `false`.
Type	Boolean

Access	Read/Write
Example	Check the example provided with the `attachments` method.

Related concepts
attachments

indexes

Description	An array of `Index` objects, each item represents a search index.
Type	Array of `Index` objects
Example	This script displays the number of available indexes on the console. `console.println("Number of search indexes: " + search.indexes.length);`

markup

Description	Set this property to `true` to extend the scope of the search operations to annotations. The default is `false`.
Type	Boolean
Access	Read/Write
Example	Check the example provided with the `attachments` method.

Related concepts
attachments

matchCase

Description	Set this property to `true` to turn the search operation to case-sensitive. The default is `false`.
Type	Boolean
Access	Read/Write
Example	Check the example provided with the `attachments` method.

Related concepts
attachments

matchWholeWord

Description	Set this property to `true` to turn the search operation sensitive for full word matches only. That means, if you are searching for part, then particular nor participate will not be marked. The default is `false`.
Type	Boolean
Access	Read/Write

Example	Check the example provided with the `attachments` method.

Related concepts
attachments

maxDocs

Description	The maximum number of documents to include in the result set of a search query. The default is `100`.
Type	Integer
Access	Read/Write
Example	Check the example provided with the `attachments` method.

stem

Description	Set this property to `true` to include words with the same stemming in the search operation. That means, if you are searching for particle, then part will be found also. The default is `false`.
Type	Boolean
Access	Read/Write
Example	Check the example provided with the `attachments` method.

wordMatching

Description	This property determines how to deal with the individual words used in the search expression. This property relevant only in search queries with more than one word.
Values	• `MatchPhrase` • `MatchAllWords` • `MatchAnyWord` • `BooleanQuery` (default)
Type	String
Access	Read/Write
Example	Check the example provided with the `attachments` method.

search methods

addIndex

Description	Adds the specified index to the list of active indexes.
Parameters	`cDIPath` — The device-independent path to the index to add. `bSelect` — (optional) Set this to true to turn the index active.

Returns	`Index` object
Example	This script registers the `GeneralDocsIndex.zpi` as an active index. `search.addIndex("/C/MyIndexes/GeneralDocsIndex.zpi", true);`

getIndexForPath

Description	This method searches for the corresponding path in the list of active indexes and returns the matching index object.
Parameters	`cDIPath` — The device-independent path to the index file sought.
Returns	`Index` object

Example	This script looks for an index with the path and filename, then displays some of the index properties on the console.
	`var oID = search.getIndexForPath("/C/MyIndexes/GeneralDocsIndex.zpi");`
	`console.println(oID.available);`
	`console.println(oID.name);`
	`console.println(oID.path);`

query

Description	Searches an index, the active document or a folder with PDF files for the specified text and displays the results in a search window. The properties of the search object have an impact on the search results.
Parameters	`cQuery` — The text to search. `cWhere` — (optional) Sets the scope of the operation: • `ActiveDoc` • `ActiveIndexes` (default) • `Folder` • `Index` `cDDIPath` — (optional) This path to a folder or a catalog index should be specified only if the `cWhere` argument is `Folder` or `Index`.

Example	This script searches for the word `Continued` in files or catalogs within the specified folder.
	`if (typeof search != "undefined" && search.available) {`
	`  search.query("Continued", "Folder", "/C/Temp");`
	`}`

removeIndex

Description	Removes the specified index from the list of active indexes.
Parameters	`index` — The index object to remove.
Example	This script removes the first index from the list of active indexes. `search.removeIndex(search.indexes[0]);`

security

This static object provides tools related to PDF-security.

security constants

StandardHandler

Description	This property is out of use.
Type	String
Access	Read

PPKLiteHandler

Description	Use this `HandlerName` constant in the `handler` property of the `SecurityPolicy` object if it applies a PPKLite (certificate-based) security handler. Pass this value to `security.getHandler` to create a new security context.
Type	String
Access	Read

security properties

handlers

Description	This array contains the names of all available security handlers. You may use these handlers for signatures or encryption.
Type	Array
Access	Read

Example	This script lists the `handlers` array to the console.
	`for ( var i=0; i < security.handlers.length; i++ )`
	`console.println("#" + i + ": " + security.handlers[i]);`

Related concepts
Field.signatureSign
getHandler

security methods

getHandler

Description	Returns with a `SecurityHandler` object. You can reuse an existing handler or create a new instance.
Parameters	`cName` — The name of the handler. (For details, see the `handlers` property.) `bUIEngine` — (optional) Set this to `true` to reuse an existing handler instance. The default is `false`.
Returns	`SecurityHandler` object
Example	Check the example provided with the `Field.signatureSign` method.

Related concepts
Field.signatureSign

SecurityHandler

This object provides access to signatures, encryption, and identifiers. `SecurityHandler` objects are different, not necessarily implementing each property and method detailed in the following chapters.

You may obtain a `SecurityHandler` object using the `security.getHandler` method.

Related concepts
security.getHandler

SecurityHandler properties

appearences

Description	An array containing the names for each available user-configured appearances regarding the specified security handler.
Type	Array
Access	Read

digitalIDs

Description	The certificates associated with the active digital ID for this security handler.
Type	Object
Returns	Returns with a generic object with the following properties: `certs` — (Array of `Certificate` objects) This array of certificate objects corresponds to all digital IDs for this `SecurityHandler` object.
Access	Read

isLoggedIn

Description	Returns `true` if the login was successful and the timeout period is not expired yet for the current security handler.
Type	Boolean
Access	Read

Example	This script logs in the user then displays the login status on the console.
	`var oSH = security.getHandler( "Adobe.PPKLite" );`
	`var success = oSH.login({ cDIPath: "/C/IDs/JohnDoe.pfx",`
	`cPassword: "123456",`
	`bUI : true`
	`});`
	`console.println( "Is logged in = " + oSH.isLoggedIn );`

Related concepts

login
setPasswordTimeout

loginName

Description	This property holds the login name for the current digital ID on this security handler.
Type	String
Access	Read

Example	This script logs in the user then displays some properties on the console.
	`var oSH = security.getHandler( "Adobe.PPKLite" );`
	`var success = oSH.login({ cDIPath: "/C/IDs/JohnDoe.pfx",`
	`cPassword: "123456",`
	`bUI : true`
	`});`
	`console.println( "loginName: " + oSH.loginName );`
	`console.println( "loginPath: " + oSH.loginPath );`
	`console.println( "name: " + oSH.name );`
	`console.println( "signAuthor: " + oSH.signAuthor );`
	`console.println( "signVisible: " + oSH.signVisible );`
	`console.println( "signValidate: " + oSH.signValidate );`
	`console.println( "signInvisible: " + oSH.signInvisible );`

loginPath

Description	This property holds the device-independent path to the user's profile file, including the file name. Returns a null value if the user is not logged in, or if this property is not supported by the security handler, or it is irrelevant for the active user.
Type	String
Access	Read
Example	Check the example provided with the `loginName` property.

Related concepts
loginName

name

Description	This property holds the device-independent name of the security handler.
Type	String
Access	Read
Example	Check the example provided with the `loginName` property.

Related concepts
loginName

signAuthor

Description	This property indicates if the security handler is able to prepare certified documents. For modification prevention details see the `SignatureInfo` object and its `mdp` property.
Type	Boolean
Access	Read
Example	Check the example provided with the `loginName` property.

Related concepts
loginName
SignatureInfo.mdp
Related information
SignatureInfo object

signInvisible

Description	This property indicates if the security handler is able to prepare invisible signatures.
Type	Boolean
Access	Read
Example	Check the example provided with the `loginName` property.

Related concepts
loginName

signValidate

Description	This property indicates if the security handler is able to validate signatures.
Type	Boolean
Access	Read

Example	Check the example provided with the `loginName` property.

Related concepts
loginName

signVisible

Description	This property indicates if the security handler is able to prepare visible signatures.
Type	Boolean
Access	Read
Example	Check the example provided with the `loginName` property.

Related concepts
loginName

SecurityHandler methods

login

Description	Launches the login UI which provides access to the digital IDs. Parameters are varying by handler type.
Parameters	`cPassword` — (optional) The password for the digital ID. `cDIPath` — (optional) The device-independent path to the digital ID file. `bUI` — (optional) Set this to `true` to launch the login interface to ask for credentials.
Returns	`true` if the login was successful, otherwise `false`
Example	Check the example provided with the `Field.signatureSign` method.

Related concepts
Field.signatureSign
getHandler

logout

Description	Performs a logout on the `SecurityHandler` object.
Returns	`true` if the logout was successful, otherwise `false`

Related concepts
Field.signatureSign
getHandler
login

newUser

Description	Creates a new self-sign credential, enrolling with the `Adobe.PPKLite` security handler.
Parameters	`cPassword` — (optional) The password for the digital ID. `cDIPath` — (optional) The device-independent path to the digital ID file. `oRDN` — (optional) An `RDN` object, containing the issuer or subject name for the certificate. Only the common name (`RDN.cn`) is required. If there is a country code (`RDN.c`) provided, then it should match with *ISO 3166*. `oCPS` — (optional) Provide a generic object here, containing the certificate policy information to embed. `oCPS` has the following properties: • `oid` — Certificate Policy object identifier. • `url` — (optional) An URL pointing to the policy description information. • `notice` — (optional) A digested version of the certificate information. `bUI` — (optional) Set this to `true` to launch the login interface to ask for credentials. `cStore` — Specifies the credential store to use. If not provided, than `cDIPath` will be used instead.
Returns	`true` if successful

Related concepts

RDN object

setPasswordTimeout

Description	Set the length of the timeout period in seconds.
Parameters	`cPassword` — The password for the digital ID. `iTimeout` — The length of the timeout session in seconds. Set to `0` to expire immediately (default). Specify `0x7FFFFFFF` for unlimited session length (no expiration).
Returns	Raises an exception in case of failure (for example, the user is not logged in).

Example	This script logs in to an existing ID file then sets the timeout period to 30 seconds.
	`var oSH = security.getHandler( "Adobe.PPKLite" );`
	`var success = oSH.login({ cDIPath: "/C/IDs/JohnDoe.pfx",`
	`cPassword: "123456",`
	`bUI : true`
	`});`
	`console.println(success ? "Logged in" : "Login failed");`
	`var n = 30;`
	`if (success) {`
	`oSH.setPasswordTimeout( "123456", n );`
	`console.println("TimeOut is set to " + n + " seconds.")`
	`}`

SignatureInfo

This generic object contains the properties of a digital signature. The `Field.signatureValidate` method returns a `SignatureInfo` object. `Field.signatureSign` and `Field.signatureValidate` methods receive a `SignatureInfo` object as an argument.

Different handlers may support different properties.

SignatureInfo properties

date

Description	This property holds the name or the e-mail address of the user, who created this signature.
Type	String
Access	Read
Example	Check the example provided with the `name` property.

Related concepts

name

handlerName

Description	This property holds the name of the security handler used for this signature.
Type	String
Access	Read
Example	Check the example provided with the `name` property.

Related concepts

name

location

Description	This property holds the location, where this signature was signed. It is optional and specified by the signer.
Type	String
Access	Read/Write
Example	Check the example provided with the `name` property.

Related concepts

name

mdp

Description	This property holds the Modification Detection and Prevention (MDP) setting for signing. Valid values are: • `allowNone` — No changes allowed to the document without invalidating the signature. • `allowAll` — Allows all changes to the document without invalidating the signature. • `default` — Allows form field fill-in only, any other changes invalidate the document. • `defaultAndComments` — Allows form field fill-in and adding, deleting, or editing annotations. Any other change invalidates the document.
Type	String
Access	Read/Write
Example	Check the example provided with the `name` property.

Related concepts

name

name

Description	This property holds the name of the user, who created this signature.
Type	String
Access	Read

Example	Create a blank signature named `Signature1` as described at the `addField` method. Click `Signature1`, then provide Digital ID information in the first page of the Sign Document dialog. Click Next to move to the second page. Select the desired Appearance, then click Save to close the dialog. Now the signature is signed.
	Bind this code to the `Mouse Up` trigger of a button in the `Button Properties` dialog box. Click the button to run the script, which displays all `SignatureInfo` properties on the console.
	`// Get the signature info`
	`var sf = this.getField("Signature1");`
	`sf.signatureValidate();`
	`var si = sf.signatureInfo();`
	`console.println( "Signature Attributes:" );`
	`// Basic information`
	`console.println("date = " + si.date);`
	`console.println("handlerName = " + si.handlerName);`
	`console.println("location = " + si.location);`
	`console.println("mdp = " + si.mdp);`
	`console.println("name = " + si.name);`
	`console.println("numRevisions = " + si.numRevisions);`
	`console.println("reason = " + si.reason);`
	`console.println("revision = " + si.revision);`
	`console.println("sigValue = " + si.sigValue);`
	`console.println("status = " + si.status);`
	`console.println("statusText = " + si.statusText);`
	`console.println("subFilter = " + si.subFilter);`
	`console.println("byteRange = " + si.byteRange);`
	`console.println("idValidity = " + si.idValidity);`
	`// Additional signatureInfo properties from PPKLite`
	`console.println("contact info = " + si.contactInfo);`

Related concepts
Doc.addField

numRevisions

Description	This property holds the number of revisions. Used for signature fields only.
Type	Number
Access	Read
Example	Check the example provided with the `name` property.

Related concepts
name

reason

Description	The user may specify the reason of signing in this property.
Type	String
Access	Read/Write
Example	Check the example provided with the `name` property.

Related concepts
name

revision

Description	This property holds the signature revisions corresponding to this signature field. Used for signature fields only.
Type	Number
Access	Read
Example	Check the example provided with the `name` property.

Related concepts
name

sigValue

Description	This string contains the raw bytes of the signature in hex-encoded format.
Type	String
Access	Read
Example	Check the example provided with the `name` property.

Related concepts
name

status

Description	This property holds the status code of the last validation performed on this signature.
Values	Valid values are: `-1` — The referred field is not a signature field, so it cannot have a validation status code. `0` — The signature is not signed or blank. `1` — The status is not yet known, because the signature is not yet validated. This may happen if the download of the document is still in progress. `2` — The signature is invalid. Either the signature got corrupted, or the document was changed since signing. `3` — The signature is valid but the identity of the signer could not be verified. `4` — The signature is valid and the identity of the signer is verified.
Type	Number
Access	Read
Example	Check the example provided with the `name` property.

Related concepts
name
Field.signatureValidate

statusText

Description	Language-dependent status text, which interprets or complements the status code of the last validation performed on this signature. See the `status` property for details.
Type	String
Access	Read
Example	Check the example provided with the `name` property.

Related concepts
status
name
Field.signatureValidate

subFilter

Description	This string specifies the (sub)format to use for signing. For the complete list of values refer to the *PDF Reference version 1.7* document. Make sure the addressed signature handler supports this format.
Type	String
Access	Read/Write
Example	Check the example provided with the `name` property.

Related concepts
name

contactInfo

Description	The author may specify any contact information (such as phone number or e-mail address) in this property. This way recipients may contact the author personally.
Type	String
Access	Read/Write
Example	Check the example provided with the `name` property.

Related concepts
name

byteRange

Description	This property is an array of numbers defining the bytes covered by this signature.
Type	Array

Access	Read
Example	Check the example provided with the `name` property.

Related concepts
name

idValidity

Description	The validity of signer identity as a status code. The same kind of code is used to reflect the validity status of the signature field in the `status` property.
Type	Number
Access	Read
Example	Check the example provided with the `name` property.

Related concepts
name
status

util

This static object offers utility methods for date and string formatting and parsing.

util methods

printd

Description	This method converts and formats a date (and time) object according to the `cFormat` parameter.

Parameters	`cFormat` — (optional) The format string specifying the date output may work one of the following ways:
	• A composite formatting pattern, constructed from the following date and time place-holders:
	`mmmm` — The name of the month in words (for example: August)
	`mmm` — The name of the month in abbreviated form (for example: Aug)
	`mm` — The number of the month with leading zero (for example: 08)
	`m` — The number of the month with leading zero (for example: 8)
	`dddd` — The day of the week in words (for example: Saturday)
	`ddd` — The day of the week in words, abbreviated (for example: Sat)
	`dd` — The numeric date with leading zero (for example: 05)
	`d` — The numeric date without leading zero (for example: 5)
	`yyyy` — The year in four digits (for example: 2018)
	`yy` — The year in two digits (for example: 18)
	`HH` — 24-hour time with leading zero (for example: 08)
	`H` — 24-hour time with leading zero (for example: 8)
	`hh` — 12 hour time with leading zero (for example: 08)
	`h` — 12 hour time with leading zero (for example: 8)
	`MM` — Minutes with leading zero (for example: 02)
	`M` — Minutes without leading zero (for example: 2)
	`ss` — Seconds with leading zero (for example: 03)
	`s` — Seconds without leading zero (for example: 3)
	`tt` — Indication of am/pm (for example: pm)
	`t` — Single digit indication of am/pm (for example: pm)
	`jj` — Japanese Emperor Year
	`j` — Japanese Emperor Year, abbreviated
	`\` — Escape character
	• If `cFormat` is an integer value (0, 1, or 2), then it may specify the following:
	  • `0` — PDF date, such as D:20180719162912
	  • `1` — Universal date, such as 2018.07.19 16:28:51
	  • `2` — Localized string, such as 2018/07/19 16:27:47
	• If the `bXFAPicture` argument is `true`, then this parameter is interpreted according to the XFA Picture Clause format.
	
	`oDate` — The `Date` object to format.
	`bXFAPicture` — This boolean value determines, if the `cFormat` argument should be interpreted along the *XFA Picture Clause* format. For details, refer to the *XFA Specification, Version 2.2*.

Returns	The formatted date as `String`

Example	This script displays a date, converts it to a `Date` object, then displays it on the console again in reverse order.
	`var s = "2018/07/20";`
	`console.println("Original date: " + s);`
	`var d = util.scand("yyyy/mm/dd", s);`
	`console.println("Reverse date: " + util.printd("mm/dd/yyyy", d));`

printf

Description	This method formats and concatenates the arguments according to the format string (`cFormat`) provided as the first argument.
Parameters	`cFormat` — (optional) The format string, defining the structure of the output, consists of two kinds of objects: • Standard text, to be copied into the output. • Conversion specification codes, always with a leading % tag. The standard text may contain multiple % tags according to the following formula (brackets indicate optional sections): `%[nWidth][,nDecSep][cFlags][.nPrecision]cConvChar` `nWidth` — Specifies the minimal width of the field in characters, this includes separators and other non-numerical characters also. Values having fewer characters will be padded with leading spaces or zeros, as it determined in the `cFlags` section above. Values extending over the field width are truncated. `nDecSep` — A comma character followed by one of the following codes to determine decimal and thousand separator format: • `0` — Uses comma as thousand separator, period as decimal separator. • `1` — Does not use thousand separator, uses period as decimal separator. • `2` — Uses period as thousand separator, comma as decimal separator. • `3` — Does not use thousand separator, uses comma as decimal separator. `cFlags` — This code affects only numeric arguments, consisting of one or more of these tokens: • `+` — Adds the leading sign (unary operator) always, either `+` or `-`. • `space` — Uses a leading space for the number, if it does not have a sign. • `0` — Padding the field with leading zeros instead of spaces. • `#` — Specifies an alternate output form according to the `cConvChar` code below. If this is `f`, then the output always has a decimal point. `nPrecision` — A period character followed by a number, which determines the number of digits after the decimal separator in a floating point conversion. `cConvChar` — This single character code specifies how to display the argument:: • `d` — Integer • `f` — Floating point number • `s` — String • `x` — Integer in hexadecimal format `arguments` — (optional) Optional arguments (separated by commas), each containing a piece of data to be inserted in place of the next % tag specified in the `cFormat` parameter. The number of arguments should be equal to the number of % tags in the `cFormat` parameter.
Returns	The formatted `String`

Example	This script displays a floating point number in various ways.
	`var f = Math.sqrt(2) * 100;`
	`console.println(util.printf("Decimal: %d", f));`
	`console.println(util.printf("Hexadecimal: %x", f));`
	`console.println(util.printf("Floating point: %.2f", f));`
	`console.println(util.printf("String: %s", f));`

printx

Description	This method formats a string using a formatting pattern of commands.
Parameters	`cFormat` — The format string specifying the output pattern may contain one or more of the following command characters: • `?` — Copy next character. • `X` — Copy next alphanumeric character (letters and numbers), skipping others. • `A` — Copy next letter, skipping others like numbers and signs. • `9` — Copy next numeric character, skipping others. • `*` — Copy the rest of the source string from the current position. • `\` — Escape character. • `>` — Switches to uppercase mode. • `<` — Switches to lowercase mode.. • `=` — Preserve original casing from now on. `cSource` — The source string to process.
Returns	The formatted `String`

Example	This script displays both the source string, the formatting and the output formatted as a Hungarian phone number on the console.
	`var s = "blablabla123456789zzz";`
	`console.println("source:" + s);`
	`var f="+36 (99) 999\-9999";`
	`console.println("Mask   :" + f);`
	`o = util.printx(f, v);`
	`console.println("Output:" + o);`

scand

Description	This method converts string to a `Date` (and time) object according to the `cFormat` parameter.
Parameters	`cFormat` — (optional) The format string specifying the pattern how to interpret the string as a date. Use the `cFormat` argument as it is described in the `printd` method. `cDate` — The string with the date to convert.
Returns	The `Date` object or null in case of failure.

Example	This script displays the current date and time on the console in a year/month/day order and 24-hour time format.
	`var d = "2018/07/20 14:09";`
	`console.println(util.printd("yyyy/mm/dd HH:MM", d));`

stringFromStream

Description	Converts a `ReadStream` object to a string. Useful for processing of embedded data objects or files.
Parameters	`oStream` — The `ReadStream` object as the source of the conversion. `cCharSet` — (optional) The encoding scheme for the stream, one of the following: • `utf-8` (default) • `utf-16` • `Shift-JIS` • `BigFive` • `GBK` • `UHC`
Returns	String
Example	See the `Doc.getDataObjectContents` method.

Related concepts
getDataObjectContents

Chapter 2

JavaScript 3D

JavaScript objects for 3D in Power PDF

This section describes the supported JavaScript objects and related functions in alphabetical order.

Background

Description	Represents the background of a Canvas.
Properties	• image
Functions	• `getColor()` • `getImage()` • `setColor(topColor, bottomColor)` — If two colors are used, the background is a linear gradient from top to bottom. `bottomColor` is optional. • `setImage(image)`

Bone

Type of Node used to modify the shape of a Mesh, and is usually moved over time to create animated characters. See Node.

BoundingBox

Description	Represents an axis-aligned bounding box.
Properties	Coordinates with x, y, and z values: • center • max • min

Camera

Description	A Node that controls the projection from world space to screen space. In addition to the methods and properties below, the Node methods and properties are also applicable here.
Properties	• absoluteBindingScale • binding — Possible values are `"min"`, `"max"`, `"horizontal"`, and `"vertical"`. • BINDING_HORIZONTAL

	• BINDING_MAX • BINDING_MIN • BINDING_VERTICAL • far — Distance to the far clipping plane. A value of -1 for both near and far signifies to use auto clipping plane calculations. • fov — Size of the field of view in radians. • near — Distance to the far clipping plane. • position • positionLocal • projectionType — Possible values are `"perspective"` and `"orthographic"`. • roll — Roll angle in radians. • target • targetPosition • TYPE_ORTHOGRAPHIC • TYPE_PERSPECTIVE • up • upLocal • useAbsoluteBinding • viewPlaneSize — Measured in scene units.
Functions	• `getScreenFromPosition(position, canvasWidth, canvasHeight)` • `getDirectionFromScreen(x, y, canvasWidth, canvasHeight)` — All parameters are in pixels.

CameraEvent

Description	Describes the format of the object that is passed as an argument to the `onEvent` method of the CameraEventHandler object.
Properties	• binding — Possible values are `"min"`, `"max"`, `"horizontal"`, and `"vertical"`. • canvas • currentTool • far — Distance to the far clipping plane. A value of -1 for both near and far signifies to use auto clipping plane calculations. • fov — Size of the field of view in radians. • isNewCanvas • near — Distance to the far clipping plane. • projectionType — Possible values are `"perspective"` and `"orthographic"`. • targetDistance • transform • view • viewPlaneSize — Measured in scene units.

CameraEventHandler

Description	Exposes a callback mechanism that allows a function to be evaluated when an camera event occurs. Event handlers are registered with the `addEventHandler` method of Runtime.
Functions	• `new CameraEventHandler()` • `onEvent(event)`

Canvas

Description	Represents a rectangular region into which the Scene is rendered from the viewpoint of the attached Camera.
Properties	• background
Functions	• `getCamera()` • `setCamera(camera)`

ClippingPlane

Description	An object representing a plane, within the Scene, that clips all geometry on one side of it. It is created by invoking the `createClippingPlane` method of the Scene object. See also: Node.
Functions	• `remove()`

Color

Description	Object that represents a RGB encoded color.
Properties	• b — Blue component • g — Green component • r — Red component Possible values are from 0.0 to 1.0 for each.
Functions	• `new Color()` — Obtains black by default. • `new Color(r, g, b)` • `set(color)` • `set(r, g, b)` • `set3(r, g, b)`

Console

Description	Creates output to the JavaScript console.
Functions	• `print(string)` • `println(string)` — Prints a string with an accompanying newline.

Dummy

A Node object used as an empty placeholder or a group within a Scene. See Node.

HitInfo

Description	Object returned when a hit test occurs during a MouseEvent.
Properties	• distance • material • position • surfaceNormal • target • textureCoordinate

Host

Object that provides access to the JavaScript engine context and to pertinent objects within it. The variable host is a global reference to this object. It is a reference to the JavaScript Doc object in which the 3D annotation is contained.

Image

Description	Object that represents an image.
Properties	In pixels: • height • width
Functions	• `new Image(resource)`

KeyEvent

Description	Object that is passed as an argument to the `onEvent` method of the KeyEventHandler object.
Properties	• canvas • canvasPixelHeight • canvasPixelWidth • characterCode • ctrlKeyDown • currentTool • shiftKeyDown

KeyEventHandler

Description	Object that exposes a callback mechanism that allows a function to be evaluated when a key event occurs. Event handlers are registered with the `addEventHandler` method of Runtime.
Functions	• `new KeyEventHandler()` • `onEvent(event)`

Light

Description	A Node object that illuminates meshes in the Scene. See Node.

Properties	• attenuationA
	• attenuationB
	• attenuationC
	• attenuationType — Attenuation determines how fast the light intensity decreases with distance. Possible values:
	  • `"abc"` — Uses the equation 1 / max((a+bd+cdd), 1), where d is distance.
	  • `"none"`
	• ATTENUATION_ABC
	• ATTENUATION_NONE
	• brightness — 100% brightness is obtained by value 1.
	• color
	• direction
	• directionLocal
	• innerConeAngle — In radians; within this value, the light is of uniform full density (in radians).
	• innerRadius
	• outerConeAngle — In radians; beyond this value, the intensity of the light is zero.
	• outerRadius
	• position
	• positionLocal
	• type — Possible values are `"point"`, `"spot"`, and `"infinite"`.
	• TYPE_INFINITE
	• TYPE_POINT
	• TYPE_SPOT

Material

Description	A SceneObject that controls the appearance of materials using the fixed function shader. In addition to the properties below, it contains the same methods and properties as a SceneObject.
Properties	• ambientColor • bumpTexture • diffuseColor • diffuseTexture • emissiveColor • opacity • opacityTexture • reflectionTexture • specularColor

Matrix4x4

Description	Object commonly used for transformations.
Properties	• determinant • inverse • scaleComponent • diffuseTexture • translation • transpose
Functions	• `new Matrix4x4()` • `new Matrix4x4(matrix)` • `invertInPlace` • `isEqual(matrix)` • `multiply(matrix)` • `multiplyInPlace(matrix)` • `rotateWithQuaternion(quaternion)` • `rotateWithQuaternionInPlace(quaternion)` • `rotateAboutLine(angle, start, end)` • `rotateAboutLineInPlace(angle, start, end)` • `rotateAboutX(angle)` • `rotateAboutXInPlace(angle)` • `rotateAboutVector(angle, axis)` • `rotateAboutVectorInPlace(angle, axis)` • `rotateAboutY(angle)` • `rotateAboutYInPlace(angle)` • `rotateAboutZ(angle)` • `rotateAboutZInPlace(angle)`

	• `scale(x, y, z)` • `scaleInPlace(x, y, z)` • `set(matrix)` • `set(array)` • `set(v0, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15)` • `setIdentity()` • `setView(position, direction, up)` • `transformDirection(vector)` • `transformPosition(position)` • `translate(translation)` • `translateInPlace(translation)` • `transposeInPlace()`

MenuEvent

Description	Object that is passed as an argument to the `onEvent` method of the MenuEventHandler object.
Properties	• canvas • currentTool • menuItemChecked • menuItemName

MenuEventHandler

Description	Exposes a callback mechanism that allows a function to be evaluated when an event occurs. Event handlers are registered with the `addEventHandler` method of Runtime.
Functions	• `new MenuEventHandler()` • `onEvent(event)`

Mesh

Description	A Node object that contains geometry. A Mesh object with no geometry has children Node objects that can be transformed as a group. In addition to the methods and properties below, the Node methods and properties are also applicable here.

Properties	• faceCount
	• material
	• renderMode — Possible values are:
	  • `"default"`
	  • `"bounding box"`
	  • `"transparent bounding box"`
	  • `"transparent bounding box outline"`
	  • `"vertices"`
	  • `"shaded vertices"`
	  • `"wireframe"`
	  • `"shaded wireframe"`
	  • `"solid"`
	  • `"transparent"`
	  • `"solid wireframe"`
	  • `"transparent wireframe"`
	  • `"illustration"`
	  • `"solid outline"`
	  • `"shaded illustration"`
	  • `"hidden wireframe"`
	• menuItemName

Functions	• `computeBoundingBox()` • `setColor(color,faceIndex)`

MouseEvent

Description	An object that is passed as an argument to the `onEvent` method of the MouseEventHandler object.
Properties	• canvas • canvasPixelHeight • canvasPixelWidth • ctrlKeyDown • currentTool • hits • isDoubleClick • isMouseDown • isMouseHit • isMouseMove • isMouseOut • isMouseOver • isMouseUp • leftButtonDown • mouseX • mouseY • rightButtonDown • shiftKeyDown

MouseEventHandler

Description	Exposes a callback mechanism that allows a function to be evaluated when a mouse event occurs. The handler may be customized to filter out certain event types. Event handlers are registered with the `addEventHandler` method of Runtime.
Properties	• onMouseDoubleClick • onMouseDown • onMouseHit • onMouseMove • onMouseOut • onMouseOver • onMouseUp • reportAllTargets — When set to `false`, the following events are not reported because they depend on hit testing: `mouse-hit`, `mouse-move`, `mouse-out`, and `mouse-over`. • target
Functions	• `new MouseEventHandler()` • `onEvent(event)`

Node

Description	Object within the Scene hierarchy (a SceneObject) that has a 3D representation. The following objects are considered Node objects: • Bone • Camera • ClippingPlane • Dummy • Light • Mesh • Procedural
Properties	• firstChild • hitEnabled • info • metadataString • nextSibling • opacity value — Possible values are from 0 (transparent) to 1 (opaque). • parent • transform • wireframeColor • visible
Functions	• `console.println(myNode.constructor.name)`

Procedural

A Node object used to represent procedurally created geometry, such as constructive solid geometry (CSG) solids, procedural spheres, or NURB objects (a 3D curve or surface). For methods and properties applicable to this object, see Node.

Quaternion

Description	Represents a rotation in 3D space, and allows for smooth interpolation (blending) between orientations of subjects. A Quaternion is typically used for animating a Camera or Mesh over time, and can be converted to and from angles of rotation about the axes.
Functions	• `new Quaternion()` • `new Quaternion(matrix)` • `new Quaternion(quaternion)` • `interpolate(quaternion, a)` — Possible values are from 0 to 1, signifying the degree of interpolation. • `interpolateInPlace(quaternion, a)` • `normalize()`

RenderEvent

Description	Object that is passed as an argument to the `onEvent` method of the RenderEventHandler object.
Properties	• canvas • canvasPixelHeight • canvasPixelWidth • currentTool

RenderEventHandler

Description	Object that exposes a callback mechanism that allows a function to be evaluated when an event occurs. Event handlers are registered with the `addEventHandler` method of Runtime. It issues a callback just before each Canvas is rendered.
Functions	• `new RenderEventHandler()` • `onEvent(event)`

RenderOptions

Description	Object that describes the style with which to render Node objects in the Scene.
Properties	• boundingBoxColor • clippingPlaneColor • clippingPlaneIntersectionColor • defaultAmbientColor • defaultDiffuseColor • defaultEmissiveColor • defaultSpecularColor • illustrationRenderModeFaceColor • illustrationRenderModeLineColor • pointsRenderModeColor • shadedIllustrationRenderModeLineColor • solidGridColorEven • solidGridColorOdd • solidRenderModeLineColor • transparentBoundsRenderModeColor • transparentGridColorEven • transparentGridColorOdd • wireframeRenderModeColor • xAxisColor • yAxisColor • zAxisColor

Resource

Description	Object that creates an abstraction for loading behavior in files and streams.
Properties	• type — Possible values are `"image"`, `"model"` and `"unknown"`. • TYPE_IMAGE • TYPE_MODEL • TYPE_UNKNOWN

Runtime

Description	Object that represents the run-time instance of the player. Each Runtime object can have its own unique script engine and set of annotations. The variable runtime is a global reference to this object.
Properties	• BUTTON_TYPE_PUSH • BUTTON_TYPE_TOOL • canvasCount • ctrlKeyDown • eventHandlerCount • instances • MENU_ITEM_TYPE_CHECKED • MENU_ITEM_TYPE_DEFAULT • overrideFlyTool • overrideNavTools • overridePanTool • overrideRotateTool • overrideSelection • overrideSpinTool • overrideViewChange • overrideWalkTool • overrideScrollWheel • overrideZoomTool • scrollWheelSpeed • shiftKeyDown • speedThreshold • strafeSpeed • TOOL_NAME_FLY • TOOL_NAME_MEASURE • TOOL_NAME_PAN • TOOL_NAME_ROTATE • TOOL_NAME_SPIN • TOOL_NAME_WALK • TOOL_NAME_ZOOM • version

	• viewCount • walkSpeed
Functions	• `addCustomMenuItem(name, label, type, checkedState)` • `addCustomToolButton(name, label, type)` • `addEventHandler(eventHandler)` • `disableTool(toolName)` • `enableTool(toolName)` • `getEventHandler(index)` • `getRendererName()` • `getView(index)` • `getView(name)` • `pause()` • `play()` • `refresh()` • `removeEventHandler(handler)` • `removeCustomMenuItem(menuName)` • `removeCustomToolButton(toolName)` • `setCurrentTool(toolName)` • `setCustomMenuItemChecked(menuItemName, checkedState)` • `setView(index, animate)` • `setView(name, animate)`

Scene

Description	Object that represents the hierarchy of the 3D related content, including Light, Material, and Mesh objects. The variable scene is a global reference to this object.
Properties	• ambientIlluminationColor • cameras • defaultRenderOptions • gridMode — Possible values are `"off"`, `"wire"`, `"solid"`, and `"transparent"`. • GRID_MODE_OFF • GRID_MODE_SOLID • GRID_MODE_TRANSPARENT • GRID_MODE_WIRE • gridSize • lengthUnits • LIGHT_MODE_FILE • LIGHT_MODE_NONE • LIGHT_MODE_WHITE • LIGHT_MODE_DAY • LIGHT_MODE_BRIGHT • LIGHT_MODE_RGB • LIGHT_MODE_NIGHT

	• LIGHT_MODE_BLUE
	• LIGHT_MODE_RED
	• LIGHT_MODE_CUBE
	• LIGHT_MODE_CAD
	• LIGHT_MODE_HEADLAMP
	• lights
	• lightScaleFactor
	• lightScheme — Possible values are:
	  • `"file"`
	  • `"none"`
	  • `"white"`
	  • `"day"`
	  • `"bright"`
	  • `"rgb"`
	  • `"night"`
	  • `"blue"`
	  • `"red"`
	  • `"cube"`
	  • `"cad"`
	  • `"headlamp"`
	• materials
	• meshes
	• nodes
	• showGrid
	• renderDoubleSided
	• renderMode — Possible values are:
	  • `"default"`
	  • `"bounding box"`
	  • `"transparent bounding box"`
	  • `"transparent bounding box outline"`
	  • `"vertices"`
	  • `"shaded vertices"`
	  • `"wireframe"`
	  • `"shaded wireframe"`
	  • `"solid"`
	  • `"transparent"`
	  • `"solid wireframe"`
	  • `"transparent wireframe"`
	  • `"illustration"`
	  • `"solid outline"`
	  • `"shaded illustration"`
	  • `"hidden wireframe"`
	• RENDER_MODE_DEFAULT
	• RENDER_MODE_BOUNDING_BOX

	• RENDER_MODE_TRANSPARENT_BOUNDING_BOX • RENDER_MODE_TRANSPARENT_BOUNDING_BOX_OUTLINE • RENDER_MODE_VERTICES • RENDER_MODE_SHADED_VERTICES • RENDER_MODE_WIREFRAME • RENDER_MODE_SOLID • RENDER_MODE_TRANSPARENT • RENDER_MODE_SOLID_WIREFRAME • ENDER_MODE_TRANSPARENT_WIREFRAME • RENDER_MODE_ILLUSTRATION • RENDER_MODE_SOLID_OUTLINE • RENDER_MODE_SHADED_ILLUSTRATION • RENDER_MODE_HIDDEN_WIREFRAME • selectedNode • showAxes • showOrientationAxes • smoothing • smoothingOverride
Functions	• `addModel(modelRes)` • `createClippingPlane()` • `createLight()` • `createSquareMesh(sizeX, sizeY, name)` • `computeBoundingBox()` • `update()`

SceneObject

Description	Base type for objects within the Scene, including Material and Node objects.
Properties	• name • objectGUID • objectID

SceneObjectList

Description	Structure that contains references to several SceneObject objects.
Properties	• count
Functions	• `getByGUID(guid)` • `getByID(id)` • `getByIndex(index)` • `getByName(name)` • `removeAll()` • `removeByIndex(index)` • `removeItem(scene)`

ScrollWheelEvent

Description	Object that is passed as an argument to the `onEvent` method of the ScrollWheelEventHandler object. A ScrollWheelEvent object is created when the mouse scroll wheel is activated over an active 3D Canvas object.
Properties	• canvas • canvasPixelHeight • canvasPixelWidth • ctrlKeyDown • currentTool • deltaY • shiftKeyDown

ScrollWheelEventHandler

Description	Object that exposes a callback mechanism that allows a function to be evaluated when an event occurs. Event handlers are registered with the `addEventHandler` method of Runtime.
Functions	• `new ScrollWheelEventHandler()` • `onEvent(event)`

SelectionEvent

Description	Object that is passed as an argument to the `onEvent` method of the SelectionEventHandler object. A SelectionEvent object is created when an object is selected from an active 3D Canvas object or from a model tree. If the selection is made from a Canvas object, a MouseEvent is also created.
Properties	• node • selected

SelectionEventHandler

Description	Object that exposes a callback mechanism that allows a function to be evaluated when an event occurs. Event handlers are registered with the `addEventHandler` method of Runtime.
Functions	• `new SelectionEventHandler()` • `onEvent(event)`

StateEvent

Description	Object that is passed as an argument to the `onEvent` method of the StateEventHandler object. A StateEvent object is created when state data must be stored or loaded for the scene, such as when a new comment view is created or invoked for the annotation.
Properties	• stateString — If SaveEvent type is `"load"`, this property contains the state data that was stored as part of the corresponding `"save"` StateEvent. If SaveEvent type is `"save"`, stateString is undefined. • type — Possible values are `"default"` and `"save"`. • TYPE_LOAD • TYPE_SAVE

StateEventHandler

Description	Object that exposes a callback mechanism that allows a function to be evaluated when a state event occurs. Event handlers are registered with the `addEventHandler` method of Runtime.
Functions	• `new StateEventHandler()` • `onEvent(event)`

Texture

Description	Represents the mapping of a texture.
Properties	• angle • TYPE_LOAD • clampU • clampV • image • uOffset • uScale • vOffset • vScale
Functions	• `getImage()` • `setImage(image)`

TimeEvent

Description	Object that is passed as an argument to the `onEvent` method of the TimeEventHandler object.
Properties	• deltaTime • time

TimeEventHandler

Description	Object that exposes a callback mechanism that allows a function to be evaluated when an event occurs. Event handlers are registered with the `addEventHandler` method of Runtime.
Functions	• `new TimeEventHandler()` • `onEvent(event)`

ToolEvent

Description	Object that is passed as an argument to the `onEvent` method of the ToolEventHandler object.
Properties	• canvas • canvasPixelHeight • canvasPixelWidth • currentTool • toolName

ToolEventHandler

Description	Object that exposes a callback mechanism that allows a function to be evaluated when an event occurs. Event handlers are registered with the `addEventHandler` method of Runtime.
Functions	• `new ToolEventHandler()` • `onEvent(event)`

Vector3

Description	Object comprised of three values that represent a point in space or a direction and magnitude.
Properties	• x • y • z • length
Functions	• `new Vector3()` • `new Vector3(x, y, z)` • `add(offset)` • `addInPlace(offset)` • `addScaled(offset, scale)` • `addScaledInPlace(offset, scale)` • `blend(vec, blendFactor)` — blendFactor values are from 0.0 to 1.0. • `blendInPlace(vec, blendFactor)` • `cross(vec)` • `dot(vec)` • `normalize()` • `scale(scale)` • `scaleInPlace(scale)` • `set(vec)` • `set(x, y, z)` • `set3(x, y, z)` • `subtract(offset)` • `subtractInPlace(offset)`

View

Description	Represents a named view for the annotation. See the viewCount property and the `getView` and `setView` methods of the Runtime object.
Properties	• name