

# Adobe Acrobat 7.0



## Acrobat JavaScript Scripting Reference

June 27, 2005



Adobe Solutions Network — <http://partners.adobe.com>



© 2005 Adobe Systems Incorporated. All rights reserved.

#### Acrobat® JavaScript Scripting Reference

If this guide is distributed with software that includes an end user agreement, this guide, as well as the software described in it, is furnished under license and may be used or copied only in accordance with the terms of such license. Except as permitted by any such license, no part of this guide may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Adobe Systems Incorporated. Please note that the content in this guide is protected under copyright law even if it is not distributed with software that includes an end user license agreement.

The content of this guide is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies that may appear in the informational content contained in this guide.

Please remember that existing artwork or images that you may want to include in your project may be protected under copyright law. The unauthorized incorporation of such material into your new work could be a violation of the rights of the copyright owner. Please be sure to obtain any permission required from the copyright owner.

Any references to company names in sample templates are for demonstration purposes only and are not intended to refer to any actual organization.

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Acrobat Capture, Distiller, PostScript, the PostScript logo and Reader are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

Apple, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. PowerPC is a registered trademark of IBM Corporation in the United States. ActiveX, Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. UNIX is a registered trademark of The Open Group. All other trademarks are the property of their respective owners.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA. Notice to U.S. Government End Users. The Software and Documentation are "Commercial Items," as that term is defined at 48 C.F.R. §2.101, consisting of "Commercial Computer Software" and "Commercial Computer Software Documentation," as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial Items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

# Contents

<b>Preface</b> . . . . .	<b>.27</b>
Description . . . . .	27
Audience . . . . .	27
Resources . . . . .	28
Online Help . . . . .	28
References . . . . .	28
Document Conventions . . . . .	29
Font Conventions Used in This Book . . . . .	29
Quick Bars . . . . .	31
<b>Acrobat JavaScript Scripting Reference</b> . . . . .	<b>.35</b>
ADBC Object . . . . .	35
ADBC Properties . . . . .	36
SQL Types . . . . .	36
JavaScript Types . . . . .	37
ADBC Methods . . . . .	38
getDataSourceList . . . . .	38
newConnection . . . . .	38
Alerter Object . . . . .	40
Alerter Object Methods . . . . .	40
dispatch . . . . .	40
AlternatePresentation Object . . . . .	42
AlternatePresentation Properties . . . . .	43
active . . . . .	43
type . . . . .	43
AlternatePresentation Methods . . . . .	43
start . . . . .	43
stop . . . . .	44
Annot Object . . . . .	45
Annotation Types . . . . .	45
Annotation Access from JavaScript . . . . .	48
Annot Properties . . . . .	48
alignment . . . . .	49
AP . . . . .	49
arrowBegin . . . . .	50
arrowEnd . . . . .	50
attachIcon . . . . .	51
author . . . . .	51
borderEffectIntensity . . . . .	51

borderEffectStyle	51
callout	52
caretSymbol	52
contents	52
creationDate	53
dash	53
delay	53
doc	54
doCaption	54
fillColor	54
gestures	55
hidden	55
inReplyTo	55
intent	55
leaderExtend	56
leaderLength	56
lineEnding	56
lock	57
modDate	57
name	57
notelcon	58
noView	58
opacity	59
page	59
point	59
points	60
popupOpen	60
popupRect	60
print	61
quads	61
rect	61
readOnly	61
refType	62
richContents	62
richDefaults	63
rotate	63
seqNum	63
state	64
stateModel	64
strokeColor	64
style	65
subject	65
textFont	65
textSize	66
toggleNoView	66
type	66
soundlcon	67
vertices	67
width	67
Annot Methods	67
destroy	67

getProps	68
getStateInModel	69
setProps	70
transitionToState	70
Annot3D Object	72
Annot3D Properties	72
activated	72
context3D	72
innerRect	72
name	73
page	73
rect	73
App Object	73
App Properties	73
activeDocs	73
calculate	74
constants	75
focusRect	75
formsVersion	76
fromPDFConverters	76
fs	76
fullscreen	77
language	77
media	78
monitors	78
numPlugIns	79
openInPlace	79
platform	79
plugIns	79
printColorProfiles	80
printerNames	80
runtimeHighlight	80
runtimeHighlightColor	81
thermometer	81
toolbar	81
toolbarHorizontal	82
toolbarVertical	82
viewerType	82
viewerVariation	83
viewerVersion	83
App Methods	83
addMenuItem	83
addSubMenu	85
addToolButton	86
alert	88
beep	90
beginPriv	91
browseForDoc	91
clearInterval	93

clearTimeOut	93
endPriv	94
execDialog	94
execMenuItem	109
getNthPlugInName	111
getPath	111
goBack	112
goForward	113
hideMenuItem	113
hideToolBarButton	113
launchURL	114
listMenuItems	115
listToolBarButtons	116
mailGetAdrs	116
mailMsg	117
newDoc	118
newFDF	120
openDoc	121
openFDF	123
popupMenu	124
popupMenuEx	124
removeToolButton	126
response	127
setInterval	128
setTimeout	129
trustedFunction	130
trustPropagatorFunction	133
App.media Object	138
App.media Object Properties	138
align	138
canResize	139
closeReason	139
defaultVisible	140
ifOffScreen	140
layout	140
monitorType	141
openCode	141
over	142
pageEventNames	142
raiseCode	143
raiseSystem	143
renditionType	144
status	144
trace	145
version	145
windowType	145
App.media Object Methods	146
addStockEvents	146
alertFileNotFound	146
alertSelectFailed	147

argsDWIM	148
canPlayOrAlert	148
computeFloatWinRect	149
constrainRectToScreen	150
createPlayer	150
getAltTextData	153
getAltTextSettings	153
getAnnotStockEvents	155
getAnnotTraceEvents	155
getPlayers	155
getPlayerStockEvents	156
getPlayerTraceEvents	157
getRenditionSettings	157
getURLData	158
getURLSettings	158
getWindowBorderSize	160
openPlayer	160
removeStockEvents	162
startPlayer	162
Bookmark Object	163
Bookmark Properties	163
children	163
color	164
doc	164
name	164
open	165
parent	165
style	165
Bookmark Methods	165
addChild	165
execute	166
insertChild	167
remove	168
setAction	168
Catalog Object	169
Catalog Properties	169
isIdle	169
jobs	169
Catalog Methods	169
getIndex	169
remove	170
CatalogJob Generic Object	170
Certificate Object	171
Certificate Properties	171
binary	171
issuerDN	172
keyUsage	172
MD5Hash	172

SHA1Hash	.172
serialNumber	.173
subjectCN	.173
subjectDN	.173
ubRights	.173
usage	.175
Collab Object	.176
Collab Methods	.176
addStateModel	.176
removeStateModel	.177
Color Object	.178
Color Arrays	.178
Color Properties	.179
Color Methods	.180
convert	.180
equal	.181
Column Generic Object	.181
ColumnInfo Generic Object	.182
Connection Object	.182
Connection Methods	.183
close	.183
newStatement	.183
getTableList	.183
getColumnList	.184
Console Object	.185
Console Methods	.185
show	.185
hide	.186
println	.186
clear	.187
Data Object	.187
Data Properties	.187
creationDate	.187
modDate	.188
MIMEType	.188
name	.188
path	.188
size	.188
DataSourceInfo Generic Object	.189
Dbg Object	.189
Dbg Properties	.190
bps	.190
Dbg Methods	.191
c	.191
cb	.191



q	.191
sb	.192
si	.193
sn	.193
so	.194
sv	.194
Dialog Object	.194
Dialog Methods	.195
enable	.195
end	.195
load	.196
store	.196
Directory Object	.197
Directory Properties	.197
info	.197
Directory Methods	.200
connect	.200
DirConnection Object	.201
DirConnection Properties	.201
canList	.201
canDoCustomSearch	.201
canDoCustomUISearch	.202
canDoStandardSearch	.202
groups	.202
name	.202
uiName	.203
DirConnection Methods	.203
search	.203
setOutputFields	.205
Doc Object	.206
Doc Access from JavaScript	.206
Doc Properties	.207
alternatePresentations	.207
author	.208
baseURL	.208
bookmarkRoot	.208
calculate	.209
creationDate	.209
creator	.209
dataObjects	.209
delay	.210
dirty	.210
disclosed	.211
docID	.211
documentFileName	.212
dynamicXFAForm	.212
external	.212
filesize	.213

hidden	.213
icons	.214
info	.214
innerAppWindowRect	.215
innerDocWindowRect	.216
keywords	.216
layout	.216
media	.217
metadata	.217
modDate	.218
mouseX	.218
mouseY	.218
noautocomplete	.218
nocache	.219
numFields	.220
numPages	.220
numTemplates	.220
path	.221
outerAppWindowRect	.221
outerDocWindowRect	.221
pageNum	.222
pageWindowRect	.222
permStatusReady	.222
producer	.223
requiresFullSave	.223
securityHandler	.223
selectedAnnots	.224
sounds	.224
spellDictionaryOrder	.224
spellLanguageOrder	.225
subject	.225
templates	.225
title	.226
URL	.226
zoom	.226
zoomType	.227
Doc Methods	.227
addAnnot	.227
addField	.229
addIcon	.230
addLink	.231
addRecipientListCryptFilter	.233
addScript	.234
addThumbnails	.234
addWatermarkFromFile	.235
addWatermarkFromText	.237
addWeblinks	.239
bringToFront	.240
calculateNow	.241
closeDoc	.241
createDataObject	.242

createTemplate	.243
deletePages	.244
deleteSound	.245
embedDocAsDataObject	.245
encryptForRecipients	.246
encryptUsingPolicy	.248
exportAsText	.251
exportAsFDF	.252
exportAsXFDF	.254
exportDataObject	.255
exportXFADData	.257
extractPages	.259
flattenPages	.260
getAnnot	.261
getAnnot3D	.261
getAnnots	.261
getAnnots3D	.263
getDataObject	.263
getDataObjectContents	.263
getField	.265
getIcon	.266
getLegalWarnings	.267
getLinks	.268
getNthFieldName	.268
getNthTemplate	.269
getOCGs	.269
getOCGOrder	.270
getPageBox	.270
getPageLabel	.271
getPageNthWord	.271
getPageNthWordQuads	.272
getPageNumWords	.272
getPageRotation	.273
getPageTransition	.273
getPrintParams	.274
getSound	.274
getTemplate	.275
getURL	.275
gotoNamedDest	.276
importAnFDF	.277
importAnXFDF	.277
importDataObject	.278
importIcon	.279
importSound	.280
importTextData	.281
importXFADData	.283
insertPages	.283
mailDoc	.284
mailForm	.285
movePage	.286
newPage	.287

openDataObject	.288
print	.289
removeDataObject	.291
removeField	.291
removeIcon	.292
removeLinks	.292
removeScript	.293
removeTemplate	.293
removeThumbnails	.294
removeWeblinks	.294
replacePages	.295
resetForm	.296
saveAs	.297
scroll	.299
selectPageNthWord	.300
setAction	.300
setDataObjectContents	.301
setOCGOrder	.303
setPageAction	.303
setPageBoxes	.304
setPageLabels	.305
setPageRotations	.306
setPageTabOrder	.307
setPageTransitions	.307
spawnPageFromTemplate	.308
submitForm	.309
syncAnnotScan	.315
Doc.media Object	.316
Doc.media Object Properties	.316
canPlay	.316
Doc.media Object Methods	.318
deleteRendition	.318
getAnnot	.318
getAnnots	.319
getOpenPlayers	.320
getRendition	.321
newPlayer	.321
Error Objects	.322
Error Properties	.323
fileName	.323
lineNumber	.323
extMessage	.323
message	.324
name	.324
Error Methods	.324
toString	.324
Event Object	.324
Event Type/Name Combinations	.325

Document Event Processing . . . . .	.334
Form Event Processing. . . . .	.335
Multimedia Event Processing . . . . .	.335
Event Properties . . . . .	.336
change . . . . .	.336
changeEx . . . . .	.337
commitKey . . . . .	.338
fieldFull . . . . .	.338
keyDown . . . . .	.339
modifier. . . . .	.339
name . . . . .	.340
rc . . . . .	.340
richChange . . . . .	.340
richChangeEx . . . . .	.341
richValue . . . . .	.342
selEnd . . . . .	.342
selStart . . . . .	.343
shift . . . . .	.343
source . . . . .	.344
target . . . . .	.344
targetName . . . . .	.344
type . . . . .	.345
value . . . . .	.345
willCommit . . . . .	.346
Events Object . . . . .	.347
Events Object Methods . . . . .	.347
add . . . . .	.347
dispatch . . . . .	.348
remove . . . . .	.349
EventListener Object. . . . .	.350
EventListener Object Methods. . . . .	.351
afterBlur . . . . .	.351
afterClose . . . . .	.352
afterDestroy . . . . .	.352
afterDone . . . . .	.353
afterError . . . . .	.353
afterEscape . . . . .	.354
afterEveryEvent . . . . .	.354
afterFocus . . . . .	.355
afterPause . . . . .	.356
afterPlay . . . . .	.356
afterReady . . . . .	.357
afterScript . . . . .	.358
afterSeek . . . . .	.359
afterStatus . . . . .	.360
afterStop . . . . .	.361
onBlur . . . . .	.361
onClose . . . . .	.361
onDestroy . . . . .	.363

onDone	.363
onError	.363
onEscape	.364
onEveryEvent	.364
onFocus	.365
onGetRect	.365
onPause	.366
onPlay	.367
onReady	.367
onScript	.368
onSeek	.368
onStatus	.369
onStop	.369
FDf Object	.370
FDf Properties	.370
deleteOption	.370
isSigned	.370
numEmbeddedFiles	.371
FDf Methods	.371
addContact	.371
addEmbeddedFile	.372
addRequest	.373
close	.373
mail	.374
save	.375
signatureClear	.375
signatureSign	.376
signatureValidate	.377
Field Object	.378
Field Access from JavaScript	.378
Field Properties	.380
alignment	.380
borderStyle	.381
buttonAlignX	.382
buttonAlignY	.382
buttonFitBounds	.383
buttonPosition	.383
buttonScaleHow	.384
buttonScaleWhen	.384
calcOrderIndex	.384
charLimit	.385
comb	.385
commitOnSelChange	.386
currentValueIndices	.386
defaultStyle	.387
defaultValue	.388
doNotScroll	.389
doNotSpellCheck	.389
delay	.389

display	.390
doc	.390
editable	.391
exportValues	.391
fileSelect	.392
fillColor	.392
hidden	.393
highlight	.393
lineWidth	.394
multiline	.394
multipleSelection	.394
name	.395
numItems	.395
page	.395
password	.396
print	.396
radiosInUnison	.397
readonly	.397
rect	.397
required	.398
richText	.398
richValue	.399
rotation	.400
strokeColor	.401
style	.401
submitName	.402
textColor	.402
textFont	.402
textSize	.404
type	.404
userName	.405
value	.405
valueAsString	.406
Field Methods	.406
browseForFileToSubmit	.406
buttonGetCaption	.407
buttonGetIcon	.408
buttonImportIcon	.409
buttonSetCaption	.410
buttonSetIcon	.410
checkThisBox	.411
clearItems	.412
defaultIsChecked	.413
deleteItemAt	.413
getArray	.414
getItemAt	.414
getLock	.415
insertItemAt	.416
isBoxChecked	.416
isDefaultChecked	.417
setAction	.417

setFocus	.418
setItems	.419
setLock	.420
signatureGetModifications	.421
signatureGetSeedValue	.423
signatureInfo	.423
signatureSetSeedValue	.425
signatureSign	.428
signatureValidate	.430
FullScreen Object	.431
FullScreen Properties	.431
backgroundColor	.431
clickAdvances	.432
cursor	.432
defaultTransition	.432
escapeExits	.432
isFullScreen	.433
loop	.433
timeDelay	.433
transitions	.433
usePageTiming	.434
useTimer	.434
Global Object	.434
Creating Global Properties	.434
Deleting Global Properties	.435
Global Methods	.435
setPersistent	.435
subscribe	.436
Icon Generic Object	.437
Icon Stream Generic Object	.437
Identity Object	.437
Identity Properties	.438
corporation	.438
email	.438
loginName	.438
name	.438
Index Object	.439
Index Properties	.439
available	.439
name	.439
path	.439
selected	.440
Index Methods	.440
build	.440
Link Object	.441
Link Properties	.441



borderColor	.441
borderWidth	.441
highlightMode	.441
rect	.442
Link Methods	.442
setAction	.442
Marker Object	.442
Marker Object Properties	.443
frame	.443
index	.443
name	.443
time	.443
Markers Object	.444
Markers Object Properties	.444
player	.444
Markers Object Methods	.444
get	.444
MediaOffset Object	.445
MediaOffset Object Properties	.446
frame	.446
marker	.446
time	.446
MediaPlayer Object	.447
MediaPlayer Object Properties	.447
annot	.447
defaultSize	.447
doc	.448
events	.448
hasFocus	.448
id	.448
innerRect	.449
isOpen	.449
isPlaying	.450
markers	.450
outerRect	.450
page	.451
settings	.451
uiSize	.451
visible	.452
MediaPlayer Object Methods	.452
close	.452
open	.453
pause	.454
play	.455
seek	.455
setFocus	.457
stop	.457
triggerGetRect	.458

where	.458
MediaReject Object	.459
MediaReject Object Properties	.459
rendition	.459
MediaSelection Object	.460
MediaSelection Object Properties	.460
selectContext	.460
players	.461
rejects	.461
rendition	.461
MediaSettings Object	.462
MediaSettings Object Properties	.462
autoPlay	.462
baseUrl	.462
bgColor	.462
bgOpacity	.463
endAt	.463
data	.464
duration	.464
floating	.465
layout	.466
monitor	.466
monitorType	.467
page	.468
palindrome	.468
players	.468
rate	.469
repeat	.469
showUI	.470
startAt	.470
visible	.470
volume	.471
windowType	.471
Monitor Object	.472
Monitor Object Properties	.472
colorDepth	.472
isPrimary	.473
rect	.473
workRect	.473
Monitors Object	.474
Monitors Object Properties	.474
Monitors Object Methods	.475
bestColor	.475
bestFit	.475
desktop	.476
document	.476
filter	.477

largest	.478
leastOverlap	.478
mostOverlap	.479
nonDocument	.479
primary	.480
secondary	.480
select	.480
tallest	.481
widest	.482
OCG Object	.482
OCG Properties	.482
constants	.482
initState	.483
locked	.483
name	.484
state	.484
OCG Methods	.485
getIntent	.485
setAction	.485
setIntent	.486
PlayerInfo Object	.486
PlayerInfo Object Properties	.486
id	.486
mimeTypes	.487
name	.487
version	.487
PlayerInfo Object Methods	.488
canPlay	.488
honors	.488
canUseData	.493
PlayerInfoList Object	.493
PlayerInfoList Object Properties	.493
PlayerInfoList Object Methods	.494
select	.494
PlugIn Object	.495
PlugIn Properties	.495
certified	.495
loaded	.495
name	.495
path	.496
version	.496
printParams Object	.496
PrintParams Properties	.496
binaryOK	.496
bitmapDPI	.497
colorOverride	.497
colorProfile	.497

constants	.498
downloadFarEastFonts	.498
fileName	.499
firstPage	.499
flags	.500
fontPolicy	.502
gradientDPI	.502
interactive	.503
lastPage	.503
nUpAutoRotate	.504
nUpNumPagesH	.504
nUpNumPagesV	.505
nUpPageBorder	.505
nUpPageOrder	.505
pageHandling	.506
pageSubset	.507
printAsImage	.508
printContent	.508
printerName	.509
psLevel	.509
rasterFlags	.510
reversePages	.511
tileLabel	.511
tileMark	.511
tileOverlap	.512
tileScale	.512
transparencyLevel	.512
usePrinterCRD	.513
useT1Conversion	.513
Rendition Object	.514
Rendition Object Properties	.514
altText	.514
doc	.514
fileName	.515
type	.515
uiName	.515
Rendition Object Methods	.516
getPlaySettings	.516
select	.517
testCriteria	.518
RDN Generic Object	.518
Report Object	.518
Report Properties	.519
absIndent	.519
color	.519
size	.519
style	.520
Report Methods	.520
breakPage	.520

divide	.520
indent	.521
outdent	.521
open	.521
save	.522
mail	.523
Report	.523
writeText	.524
Row Generic Object	.525
ScreenAnnot Object	.525
ScreenAnnot Object Properties	.525
altText	.525
alwaysShowFocus	.526
display	.526
doc	.526
events	.526
extFocusRect	.527
innerDeviceRect	.527
noTrigger	.528
outerDeviceRect	.528
page	.528
player	.528
rect	.529
ScreenAnnot Object Methods	.529
hasFocus	.529
setFocus	.529
Search Object	.530
Search Properties	.530
attachments	.530
available	.531
docInfo	.531
docText	.531
docXMP	.531
bookmarks	.532
ignoreAccents	.532
ignoreAsianCharacterWidth	.532
indexes	.532
jpegExif	.533
legacySearch	.533
markup	.533
matchCase	.533
matchWholeWord	.533
maxDocs	.534
objectMetadata	.534
proximity	.534
proximityRange	.534
refine	.535
soundex	.535
stem	.535

thesaurus . . . . .	.535
wordMatching . . . . .	.536
Search Methods . . . . .	.536
addIndex . . . . .	.536
getIndexForPath . . . . .	.537
query . . . . .	.537
removeIndex . . . . .	.538
Security Object . . . . .	.538
Security Constants . . . . .	.539
Security Properties . . . . .	.540
handlers . . . . .	.540
validateSignaturesOnOpen . . . . .	.541
Security Methods . . . . .	.541
chooseRecipientsDialog . . . . .	.541
chooseSecurityPolicy . . . . .	.544
exportToFile . . . . .	.545
getHandler . . . . .	.545
getSecurityPolicies . . . . .	.546
importFromFile . . . . .	.548
SecurityPolicy Object . . . . .	.549
SecurityPolicy Properties . . . . .	.549
SecurityHandler Object . . . . .	.549
SecurityHandler Properties . . . . .	.550
appearances . . . . .	.550
digitalIDs . . . . .	.551
directories . . . . .	.552
directoryHandlers . . . . .	.552
isLoggedIn . . . . .	.552
loginName . . . . .	.553
loginPath . . . . .	.553
name . . . . .	.553
signAuthor . . . . .	.554
signFDF . . . . .	.554
signInvisible . . . . .	.554
signValidate . . . . .	.554
signVisible . . . . .	.554
uiName . . . . .	.555
SecurityHandler Methods . . . . .	.555
login . . . . .	.555
logout . . . . .	.558
newDirectory . . . . .	.559
newUser . . . . .	.559
setPasswordTimeout . . . . .	.561
SignatureInfo Object. . . . .	.561
SignatureInfo Object properties. . . . .	.562
SOAP Object . . . . .	.571
SOAP Properties . . . . .	.572

wireDump	.572
SOAP Methods	.572
connect	.572
queryServices	.574
resolveService	.577
request	.580
response	.590
streamDecode	.592
streamDigest	.592
streamEncode	.593
streamFromString	.593
stringFromStream	.594
Sound Object	.594
Sound Properties	.594
name	.594
Sound Methods	.595
play	.595
pause	.595
stop	.595
Span Object	.595
Span Properties	.596
alignment	.596
fontFamily	.596
fontStretch	.596
fontStyle	.597
fontWeight	.597
text	.597
textColor	.597
textSize	.597
strikethrough	.598
subscript	.598
superscript	.598
underline	.599
Spell Object	.599
Spell Properties	.599
available	.599
dictionaryNames	.600
dictionaryOrder	.600
domainNames	.600
languages	.601
languageOrder	.602
Spell Methods	.602
addDictionary	.602
addWord	.603
check	.603
checkText	.604
checkWord	.605
customDictionaryClose	.606

customDictionaryCreate	.607
customDictionaryDelete	.608
customDictionaryExport	.608
customDictionaryOpen	.609
ignoreAll	.610
removeDictionary	.611
removeWord	.611
userWords	.612
Statement Object	.613
Statement Properties	.613
columnCount	.613
rowCount	.613
Statement Methods	.613
execute	.613
getColumn	.614
getColumnArray	.615
getRow	.615
nextRow	.616
TableInfo Generic Object	.617
Template Object	.618
Template Properties	.618
hidden	.618
name	.618
Template Methods	.618
spawn	.618
Thermometer Object	.620
Thermometer Properties	.620
cancelled	.620
duration	.620
value	.620
text	.621
Thermometer Methods	.621
begin	.621
end	.621
TTS Object	.622
TTS Properties	.622
available	.622
numSpeakers	.622
pitch	.623
soundCues	.623
speaker	.623
speechCues	.623
speechRate	.623
volume	.623
TTS Methods	.624
getNthSpeakerName	.624
pause	.624



qSilence	.625
qSound	.625
qText	.625
reset	.626
resume	.626
stop	.626
talk	.626
this Object	.627
Variable and Function Name Conflicts.	.627
Util Object	.628
Util Methods	.628
iconStreamFromIcon	.628
printf	.629
printfd	.631
printx	.634
scand	.635
spansToXML	.636
streamFromString	.636
stringFromStream	.637
xmlToSpans	.637
XFAObject Object.	.638
XMLData Object	.638
XMLData Object Methods.	.639
applyXPath	.639
parse	.643

**New Features and Changes . . . . . 647**

Acrobat 7.0 Changes	.647
Introduced in Acrobat 7.0	.647
Modified in Acrobat 7.0	.651
Acrobat 6.0 Changes	.652
Introduced in Acrobat 6.0	.652
Modified in Acrobat 6.0	.660
Deprecated in Acrobat 6.0	.662
Introduced in Acrobat 6.0.2	.662
Acrobat 5.0 Changes	.669
Introduced in Acrobat 5.0	.669
Modified in Acrobat 5.0	.676
Deprecated in Acrobat 5.0	.677
Modified in Acrobat 5.05	.677
Modified in Adobe 5.1 Reader	.678

<b>Security and Technical Notes . . . . .</b>	<b>679</b>
Security Notes . . . . .	.679
Technical Notes . . . . .	.680



# Preface

---

## Description

JavaScript is the cross-platform scripting language of Adobe Acrobat®. Through its JavaScript extensions, Acrobat exposes much of the functionality of the viewer and its plug-ins to the document author/form designer/plugin developer. This functionality, which was originally designed for within-document processing of forms, has been expanded and extended in recent versions of Acrobat to include the use of JavaScript in batch processing of collections of PDF documents, for developing and maintaining an online collaboration scheme, for communicating with local databases through ADBC, and for controlling multimedia events. Acrobat JavaScript objects, properties and methods can also be accessed through Visual Basic to automate the processing of PDF documents.

The [Acrobat JavaScript Scripting Reference](#) describes in detail all objects, properties and methods within the Acrobat extension to JavaScript, and gives code examples. The section [New Features and Changes](#) summarizes the new features and changes introduced in this version of Adobe Acrobat and in earlier versions.

Please review the chapter. This chapter summarizes various security changes that may affect the way the JavaScript interpreter responds to your code.

**IMPORTANT:** Certain properties and methods that may be discoverable via JavaScript's introspection facilities are not documented here. These undocumented properties and methods should not be used. They are entirely unsupported and subject to change without notice at any time.

---

## Audience

This document is intended for users familiar with core JavaScript 1.5. The intended audience would include, but is not limited to, document authors who want to create interactive PDF documents, form designers intent on designing intelligent documents, and Acrobat plug-in developers.

A knowledge of the Acrobat user interface (UI) is essential; familiarity with the PDF file format is helpful.

Some of the features of Acrobat include ADBC, multimedia, SOAP, XML and various security protocols. Using Acrobat JavaScript to control any of these features requires a detailed knowledge of the corresponding technology.

---

## Resources

The following resources provide further information about the Acrobat JavaScript.

### Online Help

The Web offers a great many resources to help you with JavaScript in general as well as JavaScript for PDF. For example:

- <http://partners.adobe.com/asn/acrobat/>—A listing of Acrobat resources for developers. This listing includes the following:
  - <http://www.adobe.com/support/forums/main.html>—Adobe Systems, Inc. provides dedicated online support forums for all Adobe products, including Acrobat and Adobe Reader.
  - <http://www.adobe.com/support/products/acrobat.html>—In addition to the forums, Adobe maintains a searchable support database with answers to commonly asked questions.

### References

#### Core JavaScript 1.5 Documentation

Complete documentation for JavaScript 1.5, the version used by Acrobat 7.0, is available on the web at <http://partners.adobe.com/NSjscript/>.

*XML Path Language (XPath) Version 1.0*, W3C Recommendation 16 November 1999. XPath is a language for addressing parts of an XML document, designed to be used by both XSLT and XPointer. <http://www.w3.org/TR/xpath>

#### Adobe Web Documentation

*PDF Reference, Fifth Edition, Version 1.6*. The PDF Reference provides a description of the PDF file format and is intended primarily for application developers wishing to develop PDF producer applications that create PDF files directly. <http://partners.adobe.com/asn/>

*Acrobat JavaScript Scripting Guide*. Gives an overview and tutorial of Acrobat JavaScript. <http://partners.adobe.com/asn/acrobat/docs.jsp>

*Acrobat and PDF Library API Overview*. Gives an overview of the objects and methods provided by the plug-in API of the Acrobat viewer. This document is available with the Adobe Acrobat Plug-ins SDK CD-ROM or on the Adobe Web site <http://partners.adobe.com/asn/>.

*Acrobat and PDF Library API Reference.* Describes in detail the objects and methods provided by the Acrobat viewer's plug-in API. This document is available with the Adobe Acrobat Plug-ins SDK CD-ROM or on the Adobe Web site <http://partners.adobe.com/asn/>.

*Adobe Dialog Manager Programmer's Guide and Reference.* This document describes the Adobe Dialog Manager (ADM). ADM is a collection of APIs for displaying and controlling dialogs in a platform-independent way. <http://partners.adobe.com/asn/>

*Forms System Implementation Notes.* This document discusses the concepts of submitting form data as URL encoded, Forms Data Format (FDF) or XML Forms Data Format (XFDF). <http://partners.adobe.com/asn/>

*Programming Acrobat JavaScript using Visual Basic.* This document gives you the information you need to get started using the extended functionality of JavaScript from a Visual Basic programming environment. <http://partners.adobe.com/asn/>

*XFA-Picture Clause 2.0 Specification.* Describes the specific language for describing patterns utilized for formatting or parsing data. <http://partners.adobe.com/asn/>

*XFA-Picture Clause Version 2.2 – CCJK Addendum.* It extends numeric, date and time picture symbols to allow the parsing and formatting of the various Chinese, Chinese (Taiwan), Japanese, and Korean numeric, date and time values. <http://partners.adobe.com/asn/>

*XML Form Data Format Specification.* This document is the XFDF specification. <http://partners.adobe.com/asn/>

*Developing for Adobe Reader®* provides an introduction to those portions of the Adobe Acrobat Software Development Kit (SDK) that pertain to your development efforts for Adobe Reader.

---

## Document Conventions

This document uses font conventions common to all Acrobat reference documents, and also uses a *quick bar* for many methods and properties to summarize their availability and usage restrictions.

### Font Conventions Used in This Book

The Acrobat documentation uses text styles according to the following conventions.

Font	Used for	Examples
monospaced	Paths and filenames	C:\templates\mytmpl.fm
	Code examples set off from plain text	These are variable declarations: AVMenu commandMenu, helpMenu;

Font	Used for	Examples
monospaced bold	Code items within plain text	The <b>GetExtensionID</b> method ...
	Parameter names and literal values in reference documents	The enumeration terminates if <b>proc</b> returns <b>false</b> .
monospaced italic	Pseudocode	ACCB1 void ACCB2 ExeProc(void) { <i>do something</i> }
	Placeholders in code examples	AFSimple_Calculate( <i>cFunction</i> , <i>cFields</i> )
blue	Live links to Web pages	The Acrobat Solutions Network URL is: <a href="http://partners/adobe.com/asn/">http://partners/adobe.com/asn/</a>
	Live links to sections within this document	See <a href="#">Using the SDK</a> .
	Live links to other Acrobat SDK documents	See the <a href="#">Acrobat and PDF Library API Overview</a> .
	Live links to code items within this document	Test whether an <b>ASAtom</b> exists.
bold	PostScript language and PDF operators, keywords, dictionary key names	The <b>setpagedevice</b> operator
	User interface names	The <b>File</b> menu
italic	Document titles that are not live links	<i>Acrobat Core API Overview</i>
	New terms	<i>User space</i> specifies coordinates for...
	PostScript variables	<i>filename</i> <b>deletefile</b>

## Quick Bars

At the beginning of most property and method descriptions, a small table or *quick bar* provides a summary of the item’s availability and usage recommendations.

This sample illustrates a quick bar, with descriptive column headings that are not normally shown.

Acrobat Pro	Approval	Reader	Security	Save and Preferences Version or Deprecated
P	X	G	S	6.0

**NOTE:** Beginning with Acrobat 7.0, each icon within a quick bar has a link to the description of its meaning.

The following tables show the symbols that can appear in each column and their meanings

Column 1: Version or Deprecated	
#.#	<p>A number indicates the version of the software in which a property or method became available. If the number is specified, then the property or method is available only in versions of the Acrobat software greater than or equal to that number.</p> <p>For Adobe Acrobat 7.0, there are some compatibility issues with older versions. Before accessing this property or method, the script should check that the forms version is greater than or equal to that number to ensure backward compatibility. For example:</p> <pre>if (typeof app.formsVersion != "undefined" &amp;&amp; app.formsVersion &gt;= 7.0) {     // Perform version specific operations. }</pre> <p>If the first column is blank, no compatibility checking is necessary.</p> <p><b>HISTORICAL NOTE:</b> Acrobat JavaScript dates back to Adobe Exchange 3.01, JavaScript functionality was added to this version via the “Acrobat Forms Author Plug-in 3.5 Update”.</p>
ⓧ	<p>As the Acrobat JavaScript extensions have evolved, some properties and methods have been superseded by other more flexible or appropriate properties and methods. The use of these older methods are discouraged and marked by ⓧ in the version column.</p>

---

**Column 2: Save and Preferences**


---

- Ⓓ Using this property or method dirties (modifies) the PDF document. If the document is subsequently saved, the effects of this method are saved as well.
- Ⓔ The preferences symbol indicates that even though this property does not change the document, it can permanently change a user's application preferences.
- 

---

**Column 3: Security**


---

- Ⓔ This property or method may only be available during certain events for security reasons (for example, batch processing, application start, or execution within the console). See the [Event Object](#) for details of the various Acrobat events.
- Beginning with Acrobat 7.0, to execute a security restricted method (Ⓔ) through a menu event, one of the following must be true:
1. Under **Edit > Preferences > General > JavaScript**, the item labeled "Enable menu items JavaScript execution privileges" must be checked.
  2. The method must be executed through a trusted function. For details and examples, see `app.trustedFunction()`.
- Please review the paragraph [Privileged versus Non-privileged Context](#) on how these restricted methods can be executed in a non-privileged context.
- NOTE:** (Version 6.0 or later) If the document has been Certified by an author who is trusted for embedded JavaScript, methods marked with a Ⓔ in the third column of its quick bar will execute without restriction, provided any other limitations, as set out in the quick bar fields, are met.
- 

---

**Column 4: Availability in Adobe Reader**


---

- If the column is blank, the property or method is allowed in any version of the Adobe Reader.
- ⓧ The property or method is not allowed in any version of the Adobe Reader.
- Ⓐ The property or method is allowed only in version 5.1 or later, of the Adobe Reader, not in versions 5.05 or below.
- Ⓕ The property or method can be accessed only in the Adobe Reader 5.1 or later depending on additional usage rights.
- Ⓕ Requires Advanced Forms Features rights.
  - Ⓒ Requires the right to manipulate Comments.
  - Ⓔ Requires document Save rights.
  - Ⓓ Requires file attachment rights.
-



---

**Column 5: Availability in Adobe Acrobat Approval**

---

If the column is blank, the property or method is allowed in Acrobat Approval.

- 
- X** The property or method is not allowed in Acrobat Approval.
- 

---

**Column 6: Availability in Adobe Acrobat**

---

If the column is blank, the property or method is allowed in Acrobat Standard and Acrobat Professional.

- 
- P** The property or method is available only in Acrobat Professional.
-



# Acrobat JavaScript Scripting Reference

Many of the JavaScript methods provided by Acrobat accept either a list of arguments as is customary in JavaScript, or alternatively, a single object argument with properties that contain the arguments. For example, these two calls are equivalent:

```
app.alert( "Acrobat Multimedia", 3);  
  
app.alert({ cMsg: "Acrobat Multimedia", nIcon: 3});
```

It is important to note that the JavaScript methods defined in support of multimedia *do not* accept either argument format interchangeably. Use the exact argument format described for each method, whether it is a list of arguments or a single object argument containing various properties.

## Parameter Help

For Acrobat Professional users, if you give an Acrobat JavaScript method an argument of **acrohelp** and execute that method in the **JavaScript Debugger** console (or any internal JavaScript editor), the method will return a list of its own arguments; for example, enter the following code in the console window:

```
app.response(acrohelp)
```

While the cursor is still on the line just entered, press either Ctrl-Enter or the Enter key on the numeric pad. The output to the console is seen to be

```
HelpError: Help.  
app.response:1:Console undefined:Exec  
====> [cQuestion: string]  
====> [cTitle: string]  
====> [cDefault: string]  
====> [bPassword: boolean]  
====> [cLabel: string]
```

Parameters listed in square brackets indicate optional parameters.

**NOTE:** The parameter help just described is not implemented for every Acrobat JavaScript method, for example, for those methods defined in the App JavaScript folder.

---


## ADBC Object

5.0			X	
-----	--	--	---	--

The Acrobat Database Connectivity (ADBC) plug-in allows JavaScripts inside of PDF documents to access databases through a consistent object model. The object model is based on general principles used in the object models for the ODBC and JDBC APIs. Like

ODBC and JDBC, ADBC is a means of communicating with a database though SQL (Structured Query Language).

ADBC is a Windows-only feature and requires ODBC (Open Database Connectivity from Microsoft Corporation) to be installed on the client machine.

**NOTE:** (Security 

The **ADBC** object, described here, is a global object whose methods allow a JavaScript to create database connection contexts or connections. Related objects used in database access are described separately:

Object	Brief Description
<a href="#">ADBC Object</a>	An object through which a list of accessible databases can be obtained and a connection can be made to one of them.
<a href="#">Connection Object</a>	An object through which a list of tables in the connected database can be obtained.
<a href="#">Statement Object</a>	An object through which SQL statements can be executed and rows retrieved based on the query.

## ADBC Properties

### SQL Types

5.0				
-----	--	--	---	--


The **ADBC** object has the following constant properties representing various SQL Types:

Constant property name	value	version
<b>SQLT_BIGINT</b>	0	
<b>SQLT_BINARY</b>	1	
<b>SQLT_BIT</b>	2	
<b>SQLT_CHAR</b>	3	
<b>SQLT_DATE</b>	4	
<b>SQLT_DECIMAL</b>	5	
<b>SQLT_DOUBLE</b>	6	

Constant property name	value	version
<code>SQLT_FLOAT</code>	7	
<code>SQLT_INTEGER</code>	8	
<code>SQLT_LONGVARIABLE</code>	9	
<code>SQLT_LONGVARCHAR</code>	10	
<code>SQLT_NUMERIC</code>	11	
<code>SQLT_REAL</code>	12	
<code>SQLT_SMALLINT</code>	13	
<code>SQLT_TIME</code>	14	
<code>SQLT_TIMESTAMP</code>	15	
<code>SQLT_TINYINT</code>	16	
<code>SQLT_VARIABLE</code>	17	
<code>SQLT_VARCHAR</code>	18	
<code>SQLT_NCHAR</code>	19	6.0
<code>SQLT_NVARCHAR</code>	20	6.0
<code>SQLT_NTEXT</code>	21	6.0

The `type` properties of the [Column Generic Object](#) and [ColumnInfo Generic Object](#) use these properties.

## JavaScript Types

5.0				
-----	--	--	---	--

The ADBC object has the following constant properties representing various JavaScript data types.

Constant Property Name	value
<code>Numeric</code>	0
<code>String</code>	1
<code>Binary</code>	2
<code>Boolean</code>	3

Constant Property Name	value
<b>Time</b>	4
<b>Date</b>	5
<b>TimeStamp</b>	6

The methods `statement.getColumn` and `statement.getColumnArray` use these types.

## ADBC Methods

### getDataSourceList

5.0			ⓧ	
-----	--	--	---	--

Obtains information about the databases accessible from a given system.

#### Parameters

None

#### Returns

An array containing a [DataSourceInfo Generic Object](#) for each accessible database on the system. The method never fails but may return a zero-length array.

#### Example

See [newConnection](#) for an example.

### newConnection

5.0		Ⓢ	ⓧ	
-----	--	---	---	--

Creates a [Connection Object](#) associated with the specified database. Optionally, you can supply a user ID and a password.

**NOTES:** (SecurityⓈ, version 6.0) It is possible to connect to a database using a connection string with no DSN, but this is only permitted, beginning with Acrobat 6.0, during a console, batch or menu event. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Parameters**

<b>cDSN</b>	The data source name (DSN) of the database.
<b>cUID</b>	(optional) User ID.
<b>cPWD</b>	(optional) Password.

**Returns**

A [Connection Object](#), or **null** on failure.

**Example**

```

/* First, get the array of DataSourceInfo Objects available on the
system */
var aList = ADBC.getDataSourceList();
console.show(); console.clear();

try {
  /* now display them, while searching for the one named
  "q32000data". */
  var DB = "", msg = "";
  if (aList != null) {
    for (var i=0; i < aList.length; i++) {
      console.println("Name: "+aList[i].name);
      console.println("Description: "+aList[i].description);
      // and choose one of interest
      if (aList[i].name=="q32000data")
        DB = aList[i].name;
    }
  }

  // did we find the database?
  if (DB != "") {
    // yes, establish a connection.
    console.println("The requested database has been found!");
    var Connection = ADBC.newConnection(DB);
    if (Connection == null) throw "Not Connected!";
  } else
    // no, display message to console.
    throw "Could not find the requested database.";
} catch (e) {
  console.println(e);
}

// alternatively, we could simple connect directly.
var Connection = ADBC.newConnection("q32000data");

```

---

## Alerter Object

7.0				
-----	--	--	--	--

Acrobat's multimedia plug-in displays error alerts under various conditions such as a missing media file. JavaScript code can customize these alerts, either for an entire document or for an individual media player.

In an alert situation, code in `media.js` calls an internal function `app.media.alert()` with parameters containing information about the alert. The `app.media.alert()` methods handles the alert by looking for alerter objects and calling their `dispatch()` methods, in this order:

```
args.alerter
doc.media.alerter
doc.media.stockAlerter
```

To handle alerts for a specific player, provide an alerter object in `args.alerter` when you call `app.media.createPlayer()` or `app.media.openPlayer()`.

To handle alerts for an entire document, set `doc.media.alerter` to an alerter object.

All alerts can be suppressed for a player or document by setting `args.alerter` or `doc.media.alerter` to `null`.

`doc.media.stockAlerter` provides the default alerts that are used if a custom alerter is not specified. This property is initialized automatically by `app.media.alert()`. Normally, `doc.media.stockAlerter` would not be reference in developer code.

The `app.media.alert()` method is implemented in JavaScript code in `media.js` and is only called from elsewhere in `media.js`. This function is not designed to be called from PDF applications, but it can be instructive to review its source code to see how the custom alert processing works.

---

## Alerter Object Methods

### dispatch

A custom alerter object has a single method, `dispatch()`, which `app.media.alert` calls to handle an alert situation.

#### Parameters

<code>alert</code>	An alert object, see the <a href="#">The alert object</a> below.
--------------------	--

#### Returns

Boolean, `true` to stop further alert processing, `false` to continue processing.



## The alert object

Properties	type	Description
<b>type</b>	String	All alert types
<b>doc</b>	<a href="#">Doc Object</a>	All alert types
<b>fromUser</b>	Boolean	All alert types
<b>error</b>	Object	Available for the "Exception" type alert. The error object has a <b>message</b> property: <b>error: { message: String }</b>
<b>errorText</b>	String	Available for the "PlayerError" type alert.
<b>fileName</b>	String	Available for the "FileNotFound" type alert.
<b>selection</b>	<a href="#">MediaSelection Object</a>	Available for the "SelectFailed" type alert.

### Example

Open a media player and suppress all alerts for this player.

```
app.media.openPlayer({ alerter: null });

// A more elaborate way to do the same thing
app.media.openPlayer(
{
  alerter:
  {
    dispatch() { return true; }
  }
});
```

### Example

For all players in this document, log any alerts to a text field and allow the normal alert box to be displayed.

```
function logAlerts( doc )
{
  count = 0;
  doc.alerter =
  {
    dispatch( alert )
    {
      doc.getField("AlertLog").value += "Alert #"
      + ++count + ": " + alert.type + "\n";
    }
  }
}
```

```
    }
    logAlerts( this );

    // Another way to keep the counter
    function logAlerts( doc )
    {
        doc.alerter =
        {
            count = 0,
            dispatch( alert )
            {
                doc.getField("AlertLog").value += "Alert #"
                    + ++this.count + ": " + alert.type + "\n";
            }
        }
    }
    logAlerts( this );
```

### Example

Handle the PlayerError alert here, with defaults for other alerts.

```
this.media.alerter =
{
    dispatch( alert )
    {
        switch( alert.type )
        {
            case "PlayerError":
                app.alert( "Player error: " + alert.errorText );
                return true;
        }
    }
}
```

---

## AlternatePresentation Object

This object provides an interface to the document's particular alternate presentation. Use `doc.alternatePresentations` to acquire an `alternatePresentation` object.

See the [PDF Reference](#), Section 9.4, for additional details on alternate presentations.

---

## AlternatePresentation Properties

### active

6.0			
-----	--	--	--

This property is **true** if presentation is currently active and **false** otherwise. When a presentation is active it controls how the document that owns it is displayed on the screen.

*Type: Boolean*

*Access: R.*

### Example

See [start](#) for an example.

### type

6.0			
-----	--	--	--

The type of the alternate presentation. Currently, the only supported type is "SlideShow".

*Type: String*

*Access: R.*

---

## AlternatePresentation Methods

### start

6.0			
-----	--	--	--

Switches document view into the alternate presentation mode and makes this **AlternatePresentation** object **active**. An exception is thrown if this method is called if any (this or another) alternate presentation is already active.

**Parameters**

<b>cOnStop</b>	(optional) Expression to be evaluated by Acrobat when presentation completes for any reason (as a result of a call to <a href="#">stop</a> , an explicit user action, or presentation logic itself).
<b>cCommand</b>	(optional) Command or script to pass to the alternate presentation. This command is presentation-specific (not an Acrobat JavaScript expression).

**Returns**

Nothing

**Example**

Assume there is a named presentation, "MySlideShow", within the document.

```
// oMySlideShow is an AlternatePresentation object
oMySlideShow = this.alternatePresentations.MySlideShow;
if (!oMySlideShow.active) oMySlideShow.start();
```

Note **this.alternatePresentations**, used to access the specified presentation by property name.

**stop**

6.0			
-----	--	--	--

Stops the presentation and switches document into the normal (PDF) presentation. An exception is thrown if this method is called when this presentation is not active.

**Parameters**

None

**Returns**

Nothing

**Example**

Assume **oMySlideShow** is an **AlternatePresentations** object. See [start](#) for a related example.

```
// stop the show if already active
if (oMySlideShow.active) oMySlideShow.stop();
```

## Annot Object

The functionality of the Acrobat Annotation Plug-in is exposed to JavaScript methods through the **annot** object. An **annot** object represents a particular Acrobat annotation; that is, an annotation created using the Acrobat annotation tool, or by using **doc.addAnnot**. See also **doc.getAnnot** and **doc.getAnnots**.

The user interface in Acrobat refers to annotations as comments.

### Annotation Types

Annotations are of different types, as reflected in the **type** property. The each type is listed in the table below, along with all documented properties returned by the **getProps** method.

Annotation Type	Properties
<b>Text</b>	<b>author</b> , <b>borderEffectIntensity</b> , <b>borderEffectStyle</b> , <b>contents</b> , <b>creationDate</b> , <b>delay</b> , <b>hidden</b> , <b>inReplyTo</b> , <b>intent</b> , <b>lock</b> , <b>modDate</b> , <b>name</b> , <b>noView</b> , <b>noteIcon</b> , <b>opacity</b> , <b>page</b> , <b>point</b> , <b>popupOpen</b> , <b>popupRect</b> , <b>print</b> , <b>readOnly</b> , <b>rect</b> , <b>refType</b> , <b>richContents</b> , <b>rotate</b> , <b>seqNum</b> , <b>state</b> , <b>stateModel</b> , <b>strokeColor</b> , <b>style</b> , <b>subject</b> , <b>toggleNoView</b> , <b>type</b> , <b>width</b>
<b>FreeText</b>	<b>alignment</b> , <b>author</b> , <b>borderEffectIntensity</b> , <b>borderEffectStyle</b> , <b>callout</b> , <b>contents</b> , <b>creationDate</b> , <b>dash</b> , <b>delay</b> , <b>fillColor</b> , <b>hidden</b> , <b>inReplyTo</b> , <b>intent</b> , <b>lineEnding</b> , <b>lock</b> , <b>modDate</b> , <b>name</b> , <b>noView</b> , <b>opacity</b> , <b>page</b> , <b>print</b> , <b>readOnly</b> , <b>rect</b> , <b>refType</b> , <b>richContents</b> , <b>richDefaults</b> , <b>rotate</b> , <b>seqNum</b> , <b>strokeColor</b> , <b>style</b> , <b>subject</b> , <b>textFont</b> , <b>textSize</b> , <b>toggleNoView</b> , <b>type</b> , <b>width</b>
<b>Line</b>	<b>arrowBegin</b> , <b>arrowEnd</b> , <b>author</b> , <b>borderEffectIntensity</b> , <b>borderEffectStyle</b> , <b>contents</b> , <b>creationDate</b> , <b>dash</b> , <b>delay</b> , <b>doCaption</b> , <b>fillColor</b> , <b>hidden</b> , <b>inReplyTo</b> , <b>intent</b> , <b>leaderExtend</b> , <b>leaderLength</b> , <b>lock</b> , <b>modDate</b> , <b>name</b> , <b>noView</b> , <b>opacity</b> , <b>page</b> , <b>points</b> , <b>popupOpen</b> , <b>popupRect</b> , <b>print</b> , <b>readOnly</b> , <b>rect</b> , <b>refType</b> , <b>richContents</b> , <b>rotate</b> , <b>seqNum</b> , <b>strokeColor</b> , <b>style</b> , <b>subject</b> , <b>toggleNoView</b> , <b>type</b> , <b>width</b>

<b>Annotation Type</b>	<b>Properties</b>
<b>Square</b>	<code>author, borderEffectIntensity, borderEffectStyle, contents, creationDate, dash, delay, fillColor, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, width</code>
<b>Circle</b>	<code>author, borderEffectIntensity, borderEffectStyle, contents, creationDate, dash, delay, fillColor, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, width</code>
<b>Polygon</b>	<code>author, borderEffectIntensity, borderEffectStyle, contents, creationDate, dash, delay, fillColor, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, vertices, width</code>
<b>PolyLine</b>	<code>arrowBegin, arrowEnd, author, borderEffectIntensity, borderEffectStyle, contents, creationDate, dash, delay, fillColor, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, vertices, width</code>
<b>Highlight</b>	<code>author, borderEffectIntensity, borderEffectStyle, contents, creationDate, delay, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, quads, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, width</code>
<b>Underline</b>	<code>author, borderEffectIntensity, borderEffectStyle, contents, creationDate, delay, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, quads, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, width</code>

<b>Annotation Type</b>	<b>Properties</b>
<b>Squiggly</b>	<code>author, borderEffectIntensity, borderEffectStyle, contents, creationDate, delay, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, quads, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, width</code>
<b>StrikeOut</b>	<code>author, borderEffectIntensity, borderEffectStyle, contents, creationDate, delay, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, quads, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, width</code>
<b>Stamp</b>	<code>AP, author, borderEffectIntensity, borderEffectStyle, contents, creationDate, delay, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, rotate, seqNum, strokeColor, style, subject, toggleNoView, type</code>
<b>Caret</b>	<code>author, borderEffectIntensity, borderEffectStyle, caretSymbol, contents, creationDate, delay, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, width</code>
<b>Ink</b>	<code>author, borderEffectIntensity, borderEffectStyle, contents, creationDate, dash, delay, gestures, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, popupOpen, popupRect, print, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, width</code>
<b>FileAttachment</b>	<code>attachIcon, author, borderEffectIntensity, borderEffectStyle, contents, creationDate, delay, hidden, inReplyTo, intent, lock, modDate, name, noView, opacity, page, point, print, readOnly, rect, refType, richContents, rotate, seqNum, strokeColor, style, subject, toggleNoView, type, width</code>

Annotation Type	Properties
Sound	<a href="#">author</a> , <a href="#">borderEffectIntensity</a> , <a href="#">borderEffectStyle</a> , <a href="#">contents</a> , <a href="#">creationDate</a> , <a href="#">delay</a> , <a href="#">hidden</a> , <a href="#">inReplyTo</a> , <a href="#">intent</a> , <a href="#">lock</a> , <a href="#">modDate</a> , <a href="#">name</a> , <a href="#">noView</a> , <a href="#">opacity</a> , <a href="#">page</a> , <a href="#">point</a> , <a href="#">print</a> , <a href="#">readOnly</a> , <a href="#">rect</a> , <a href="#">refType</a> , <a href="#">richContents</a> , <a href="#">rotate</a> , <a href="#">seqNum</a> , <a href="#">soundIcon</a> , <a href="#">strokeColor</a> , <a href="#">style</a> , <a href="#">subject</a> , <a href="#">toggleNoView</a> , <a href="#">type</a> , <a href="#">width</a>

## Annotation Access from JavaScript


Before an annotation can be accessed, it must be “bound” to a JavaScript variable through a method in the [Doc Object](#):

```
var a = this.getAnnot(0, "Important");
```

This example allows the script to now manipulate the annotation named “Important” on page 1 (0-based page numbering system) via the variable **a**. For example, the following code first stores the type of annotation in the variable **thetype**, then changes the author to “John Q. Public”.

```
var thetype = a.type;           // read property
a.author = "John Q. Public"; // write property
```

Another way of accessing the annot object is through the **doc.getAnnots()** method.

**NOTE:** In Adobe Reader 5.1 or later, you can get the value of any **annot** property except **contents**. The ability to set these properties depends on Comments document rights, as indicated by the  icon.

## Annot Properties

**NOTE:** Some property values are stored in the PDF document as names (see section 3.2.4 on name objects in the [PDF Reference](#)), while others are stored as strings (see section 3.2.3 on string objects in the [PDF Reference](#)). For a property that is stored as a name, there is a 127 character limit on the length of the string.

Examples of properties that have a 127 character limit include **AP**, **beginArrow**, **endArrow**, **attachIcon**, **noteIcon** and **soundIcon**. The [PDF Reference](#) documents all Annotation properties as well as how they are stored.



## alignment

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Controls the alignment of the text for a **FreeText** annotation.

Alignment	Value
Left aligned	0
Centered	1
Right aligned	2

Type: Number

Access: R/W

Annots: FreeText.

## AP

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The named appearance of the stamp to be used in displaying a stamp annotation. The names of the standard stamp annotations are given below:

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

Type: String

Access: R/W

Annots: Stamp.

### Example

```
var annot = this.addAnnot({
  page: 0,
  type: "Stamp",
  author: "A. C. Robot",
  name: "myStamp",
  rect: [400, 400, 550, 500],
  contents: "Try it again, this time with order and method!",
  AP: "NotApproved"
```

```
});
```

**NOTE:** The name of a particular stamp can be found by opening the PDF file in the Stamps folder that contains the stamp in question. For a list of stamp names currently in use in the document, see `doc.icons`.

## arrowBegin

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Determines the line cap style which specifies the shape to be used at the beginning of a **Line annot**. Permissible values are listed below:

```
None (default)
OpenArrow
ClosedArrow
ROpenArrow // version 6.0
RClosedArrow // version 6.0
Butt // version 6.0
Diamond
Circle
Square
Slash // version 7.0
```

Type: String

Access: R/W

Annots: Line, PolyLine.

### Example

See [setProps](#).

## arrowEnd

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Determines the line cap style which specifies the shape to be used at the end of a **Line annot**. Allowed values follows:

```
None (default)
OpenArrow
ClosedArrow
ROpenArrow // version 6.0
RClosedArrow // version 6.0
Butt // version 6.0
Diamond
Circle
Square
Slash // version 7.0
```

Type: String

Access: R/W

Annots: Line, PolyLine.

**Example**See [setProps](#).**attachIcon**

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The name of an icon to be used in displaying the annotation. Recognized values are listed below:

Paperclip  
 PushPin (default)  
 Graph  
 Tag

*Type: String**Access: R/W**Annots: FileAttachment.***author**

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Gets or sets the author of the annotation.

*Type: String**Access: R/W**Annots: all.***Example**See [contents](#).**borderEffectIntensity**

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The intensity of the border effect, if any. This represents how cloudy a cloudy rectangle, polygon or oval is.

*Type: Number**Access: R/W**Annots: all.***borderEffectStyle**

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

If non-empty, the name of a border effect style. Currently, the only supported border effects are the empty string (nothing) or "C" for cloudy.

*Type: String**Access: R/W**Annots: all.*

## callout

7.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

An array of four or six numbers specifying a callout line attached to the free text annotation. See Table 8.21 in the [PDF Reference](#) for additional details.

*Type: Array**Access: R/W**Annots: FreeText.*

## caretSymbol

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The symbol associated with a Caret annotation. Valid values are "" (nothing), "P" (paragraph symbol) or "S" (space symbol).

*Type: String**Access: R/W**Annots: Caret.*

## contents

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Accesses the contents of any annotation having a popup. In the case of **Sound** and **FileAttachment** annotations, specifies the text to be displayed as the description of the sound or file attachment.

**NOTE:** (Ⓒ) Getting and setting of this property in Adobe Reader 5.1 or later depends on Comments document rights.

*Type: String**Access: R/W**Annots: all.*

### Example

```
var annot = this.addAnnot({
    page: 0,
    type: "Text",
    point: [400,500],
    author: "A. C. Robot",
    contents: "Call Smith to get help on this paragraph.",
    noteIcon: "Help"
});
```

See also [addAnnot](#).

## creationDate

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The date and time when the annotation was created.

Type: Date

Access: R

Annots: all.

## dash

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

A dash array defining a pattern of dashes and gaps to be used in drawing a dashed border. For example, A value of [3, 2] specifies a border drawn with 3-point dashes alternating with 2-point gaps.

To set the dash array, the **style** property has to be set to "D".

Type: Array

Access: R/W

Annots: FreeText, Line, PolyLine, Polygon, Circle, Square, Ink.

### Example

Assume **annot** is an annot object, the code below changes the border to dashed.

```
annot.setProps({ style: "D", dash: [3,2] });
```

See also the example following **annot.delay**, below.

## delay

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

When **true**, property changes to the annot are queued up and then executed when **delay** is set back to **false**. (Similar to **Field.delay**.)

Type: Boolean

Access: R/W

Annots: all.

### Example

Assume **annot** is an annot object, the code below changes the border to dashed.

```
annot.delay=true;
annot.style = "D";
annot.dash = [4,3];
annot.delay = false;
```

## doc

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Returns the [Doc Object](#) of the document in which the annotation resides.

Type: **doc object**

Access: *R*

Annots: *all*.

### Example

```
var inch = 72;
var annot = this.addAnnot({
    page: 0,
    type: "Square",
    rect: [1*inch, 3*inch, 2*inch, 3.5*inch]
});
/* displays, for example,, "file:///C:/Adobe/Annots/myDoc.pdf" */
console.println(annot.doc.URL);
```

## doCaption

7.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

A boolean property, when **true** draws the rich contents in the line appearance itself. In the UI, this property corresponds to “Show Text in Line” on the property dialog.

Type: *Boolean*

Access: *R/W*

Annots: *Line*.

## fillColor

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Sets the background color for the **Circle**, **Square**, **Line**, **Polygon**, **PolyLine** and **FreeText** annotations. Values are defined by using **transparent**, **gray**, **RGB** or **CMYK** color. See [Color Arrays](#) for information on defining color arrays and how values are used with this property.

Type: *Color*

Access: *R/W*

Annots: *Circle, Square, Line, Polygon, PolyLine, FreeText*.

### Example

```
var annot = this.addAnnot(
{
    type: "Circle",
    page: 0,
    rect: [200,200,400,300],
    author: "A. C. Robot",
```

```

name: "myCircle",
popupOpen: true,
popupRect: [200,100,400,200],
contents: "Hi World!",
strokeColor: color.red,
fillColor: ["RGB",1,1,.855]
});
    
```

## gestures

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

An array of arrays, each representing a stroked path. Each array is a series of alternating **x** and **y** coordinates in *Default User Space*, specifying points along the path. When drawn, the points are connected by straight lines or curves in an implementation-dependent way. See “Ink Annotations” in the [PDF Reference](#) for more details.

Type: Array

Access: R/W

Annots: Ink.

## hidden

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

If **true**, the annotation is not shown and there is no user interaction, display or printing of the annotation.

Type: Boolean

Access: R/W

Annots: all.

## inReplyTo

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

If non-empty, the **name** value of the **annot** that this **annot** is in reply to.

Type: String

Access: R/W

Annots: all.

## intent

7.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Markup **intent** is a property by which a single markup annotation type may behave differently in a viewing application depending on the intended end-user use of the markup. For example, the **Callout Tool** is a **FreeText** annotation with **intent** set to **FreeTextCallout**.

Though this property is defined for all annotations, currently, only **FreeText**, **Polygon** and **Line** type annots have non-empty values for **intent**.

Type: String

Access: R/W

Annots: all.

The table below indicates the various tools available through the UI for creating annots with special appearances. These tools are built on standard annots.

UI	Annot Type	intent
Callout Tool	<b>FreeText</b>	<b>FreeTextCallout</b>
Cloud Tool	<b>Polygon</b>	<b>PolygonCloud</b>
Arrow Tool	<b>Line</b>	<b>LineArrow</b>
Dimensioning Tool	<b>Line</b>	<b>LineDimension</b>

### leaderExtend

7.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Specifies the length of *leader line extensions* that extend from each endpoint of the line perpendicular to the line itself. These lines extend from the line proper 180 degrees from the leader lines themselves. The value should always be greater than or equal to zero.

The default is zero which implies no leader line extension.

Type: Number

Access: R/W

Annots: Line.

### leaderLength

7.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Specifies the length of *leader lines* that extend from each endpoint of the line perpendicular to the line itself. The value may be negative to specify an alternate orientation of the leader lines.

The default is 0 which implies no leader line.

Type: Number

Access: R/W

Annots: Line.

### lineEnding

7.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

This property determines how the end of the callout line is stroked. Recognized values are



```

None (default)
OpenArrow
ClosedArrow
ROpenArrow // version 6.0
RClosedArrow // version 6.0
Butt // version 6.0
Diamond
Circle
Square
Slash // version 7.0
    
```

This property is relevant only when the [intent](#) of the FreeText annot is **FreeTextCallout**

*Type: String*                      *Access: R/W*                      *Annots: FreeText.*

## lock

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

When **true**, the annot is “locked”, which is equivalent to **readOnly** except that the annot is accessible through the UI (properties dialog).

*Type: Boolean*                      *Access: R/W*                      *Annots: all.*

## modDate

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Returns the last modification date for the annotation.

*Type: Date*                      *Access: R*                      *Annots: all.*

### Example

```

// This example prints the modification date to the console
console.println(util.printd("mmm dd, yyyy", annot.modDate));
    
```

## name

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The name of an annotation. This value can be used by `doc.getAnnot` to find and access the properties and methods of the annotation.

*Type: String*                      *Access: R/W*                      *Annots: all.*

**Example**

```
// This code locates the annotation named "myNote"
// and appends a comment.
var gannot = this.getAnnot(0, "myNote");
gannot.contents += "\r\rDon't forget to check with Smith";
```

**notelcon**

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The name of an icon to be used in displaying the annotation. Recognized values are given below:

- Check
- Circle
- Comment
- Cross
- Help
- Insert
- Key
- NewParagraph
- Note (default)
- Paragraph
- RightArrow
- RightPointer
- Star
- UpArrow
- UpLeftArrow

Type: String

Access: R/W

Annots: Text.

**Example**

See [contents](#).

**noView**

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

If **true**, the annotation is hidden, but if the annotation has an appearance, that appearance should be used for printing only.

Type: Boolean

Access: R/W

Annots: all.

**Example**

See [toggleNoView](#).

## opacity

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The constant opacity value to be used in painting the annotation. This value applies to all visible elements of the annotation in its closed state (including its background and border), but not to the popup window that appears when the annotation is opened. Permissible values are 0.0 - 1.0. A value of 0.5 makes the annot semi-transparent.

*Type: Number*

*Access: R/W*

*Annots: all.*

## page

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The page on which the annotation resides.

*Type: Integer*

*Access: R/W*

*Annots: all.*

### Example

The following code moves the Annot object, **annot**, from its current page to page 3 (0-based page numbering system).

```
annot.page = 2;
```

## point

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

An array of two numbers,  $[x_{ul}, y_{ul}]$  which specifies the upper left-hand corner in default, user's space, of an annotation's **Text**, **Sound**, or **FileAttachment** icon.

*Type: Array*

*Access: R/W*

*Annots: Text, Sound, FileAttachment.*

### Example

```
var annot = this.addAnnot({
  page: 0,
  type: "Text",
  point: [400,500],
  contents: "Call Smith to get help on this paragraph.",
  popupRect: [400,400,550,500],
  popupOpen: true,
  noteIcon: "Help"
});
```

See also [addAnnot](#) and [noteIcon](#).

## points

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

An array of two points,  $[[x_1, y_1], [x_2, y_2]]$ , specifying the starting and ending coordinates of the line in *default user space*.

Type: Array

Access: R/W

Annots: Line.

### Example

```
var annot = this.addAnnot({
  type: "Line",
  page: 0,
  author: "A. C. Robot",
  contents: "Look at this again!",
  points: [[10,40], [200,200]],
});
```

See [addAnnot](#), [arrowBegin](#), [arrowEnd](#) and [setProps](#).

## popupOpen

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

If **true** the popup text note will appear open when the page is displayed.

Type: Boolean

Access: R/W

Annots: all except FreeText, Sound, FileAttachment.

### Example

See the [print](#).

## popupRect

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

An array of four numbers  $[x_{ll}, y_{ll}, x_{ur}, y_{ur}]$  specifying the lower-left  $x$ , lower-left  $y$ , upper-right  $x$  and upper-right  $y$  coordinates—in *default user space*—of the rectangle of the *popup annotation* associated with a parent annotation and defines the location of the popup annotation on the page.

Type: Array

Access: R/W

Annots: all except FreeText, Sound, FileAttachment.

### Example

See the [print](#).

## print

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Indicates whether the annotation should be printed. When set to **true**, the annotation will be printed.

Type: Boolean

Access: R/W

Annots: all.

## quads

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

An array of  $8 \times n$  numbers specifying the coordinates of  $n$  quadrilaterals in *default user space*. Each quadrilateral encompasses a word or group of contiguous words in the text underlying the annotation. See Table 7.19, page 414 of the [PDF Reference](#) for more details. The **quads** for a word can be obtained through calls to the [getPageNthWordQuads](#).

Type: Array

Access: R/W

Annots: Highlight, StrikeOut, Underline, Squiggly.

### Example

See [getPageNthWordQuads](#) for an example.

## rect

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The **rect** array consists of four numbers  $[x_{ll}, y_{ll}, x_{ur}, y_{ur}]$  specifying the lower-left  $x$ , lower-left  $y$ , upper-right  $x$  and upper-right  $y$  coordinates—in *default user space*—of the rectangle defining the location of the annotation on the page. See also [popupRect](#).

Type: Array

Access: R/W

Annots: all.

## readOnly

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

When **true**, indicates that the annotation should display, but not interact with the user.

Type: Boolean

Access: R/W

Annots: all.

## refType

7.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The reference type of the annotation. The property is used to distinguish whether [inReplyTo](#) is indicating a plain threaded discussion relationship or a "group" relationship. Recognized values are "R" and "Group". See Table 8.17 of the [PDF Reference](#) for additional details.

*Type: String*

*Access: R/W*

*Annots: all.*

## richContents

6.0	Ⓓ		Ⓒ	
-----	---	--	---	--

This property gets the text contents and formatting of an annot. The rich text contents are represented as an array of [Span Objects](#) containing the text contents and formatting of the annot.

*Type: Array of  
[Span Objects](#)*

*Access: R/W*

*Annots: all except Sound,  
FileAttachment.*

### Example

Create a text annot, and give it some rich contents.

```
var annot = this.addAnnot({
    page: 0,
    type: "Text",
    point: [72,500],
    popupRect: [72, 500,6*72,500-2*72],
    popupOpen: true,
    noteIcon: "Help"
});

var spans = new Array();
spans[0] = new Object();
spans[0].text = "Attention:\r";
spans[0].textColor = color.blue;
spans[0].textSize = 18;

spans[1] = new Object();
spans[1].text = "Adobe Acrobat 6.0\r";
spans[1].textColor = color.red;
spans[1].textSize = 20;
spans[1].alignment = "center";

spans[2] = new Object();
spans[2].text = "will soon be here!";
```

```
spans[2].textColor = color.green;
spans[2].fontStyle = "italic";
spans[2].underline = true;
spans[2].alignment = "right";

// now give the rich field a rich value
annot.richContents = spans;
```

See also **field.richValue**, **event.richValue** (and **richChange**, **richChangeEx**) for additional examples of using the **Span Object** object.

### richDefaults

6.0	Ⓓ		Ⓒ	
-----	---	--	---	--

This property defines the default style attributes for a **FreeText** annot. See the description of **field.defaultStyle** for additional details.

Type: *Span Object*

Access: *R/W*

Fields: *FreeText.*

### rotate

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The number of degrees (0, 90, 180, 270) the annotation is rotated counter-clockwise relative to the page. The Icon based annotations do not rotate, this property is only significant for **FreeText** annotations.

Type: *Integer*

Access: *R/W*

Annots: *FreeText.*

### seqNum

5.0			Ⓒ	ⓧ
-----	--	--	---	---

Arad only sequence number for the annot on the page.

Type: *Integer*

Access: *R*

Annots: *all.*

**state**

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The state of the text annotation. The values of this property depend on the stateModel. For a state model of **Marked**, values are **Marked** and **Unmarked**; for a Review state model, the values are **Accepted**, **Rejected**, **Cancelled**, **Completed** and **None**.

Type: *String*

Access: *R/W*

Annots: *Text*.

**stateModel**

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Beginning with Acrobat 6.0, annotations may have author-specific state associated with them. The state is not specified in the annotation itself, but in a separate text annotation that refers to the original annotation by means of its **IRT** (**inReplyTo**) entry. There are two types of state models, "Marked" and "Review".

Type: *String*

Access: *R/W*

Annots: *Text*.

See also the annot method [getStateInModel](#).

**strokeColor**

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Sets the appearance color of the annotation. Values are defined by using **transparent**, **gray**, **RGB** or **CMYK** color. In the case of a **FreeText** annotation, **strokeColor** sets the border and text colors. Refer to the [Color Arrays](#) section for information on defining color arrays and how values are used with this property.

Type: *Color*

Access: *R/W*

Annots: *all*.

**Example**

```
// Make a text note red
var annot = this.addAnnot({type: "Text"});
annot.strokeColor = color.red;
```



## style

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

This property gets and sets the border style. Recognized values are **"S"** (solid) and **"D"** (dashed). The style property is defined for all annot types, but is only relevant for **Line**, **FreeText**, **Circle**, **Square**, **PolyLine**, **Polygon** and **Ink**.

*Type: String*

*Access: R/W*

*Annots: all.*

See `annot.dash` for an example.

## subject

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Text representing a short description of the subject being addressed by the annotation. The text appears in the title bar of the popup, if there is one, or the properties dialog.

*Type: String*

*Access: R/W*

*Annots: all.*

## textFont

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Determines the font that is used when laying out text in a **FreeText** annotation. Valid fonts are defined as properties of the **font** object, as listed in `field.textFont`.

An arbitrary font can be used when laying out a **FreeText** annotation by setting the value of **textFont** equal to a string that represents the PostScript name of the font.

*Type: String*

*Access: R/W*

*Annots: FreeText.*

### Example

The following example illustrates the use of this property and the font object.

```
// Create FreeText annotation with Helvetica
var annot = this.addAnnot({
  page: 0,
  type: "FreeText",
  textFont: font.Helv, // or, textFont: "Viva-Regular",
  textSize: 10,
  rect: [200, 300, 200+150, 300+3*12], // height for three lines
  width: 1,
  alignment: 1
});
```

**textSize**

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Determines the text size (in points) that is used in a **FreeText** annotation. Valid text sizes range from 0 to 32767 inclusive. A text size of zero means that the largest point size that will allow all the text data to still fit in the annotations's rectangle should be used.

Valid text sizes include zero and the range from 4 to 144 inclusive.

Type: Number

Access: R/W

Annots: FreeText.

**Example**

See [textFont](#).

**toggleNoView**

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

If **toggleNoView** is **true**, the **noView** flag is toggled when the mouse hovers over the annot or the annot is selected. The flag reflects a new flag in the PDF language.

If an annot has both the **noView** and **toggleNoView** flags set, the annot will generally be invisible; however, when the mouse is over it or it is selected, it will become visible.

Type: Boolean

Access: R/W

Annots: all.

**type**

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Reflects the type of annotation. The type of the annotation can only be set within the object-literal argument of the **doc.addAnnot** method. The valid values are:

- Text
- FreeText
- Line
- Square
- Circle
- Polygon
- PolyLine
- Highlight
- Underline
- Squiggly
- StrikeOut
- Stamp
- Caret
- Ink

FileAttachment  
Sound

Type: String

Access: R

Annots: all.

## soundIcon

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The name of an icon to be used in displaying the annotation. A value of "Speaker" is recognized.

Type: String

Access: R/W

Annots: Sound.

## vertices

6.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

An array of coordinate arrays representing the alternating horizontal and vertical coordinates, respectively, of each vertex, in default user space of a polygon or polyline annotation. See Table 8.25 of the [PDF Reference](#) for details.

Type: Array of arrays

Access: R/W

Annots: Polygon, PolyLine.

## width

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

The border width in points. If this value is 0, no border is drawn. The default value is 1.

Type: Number

Access: R/W

Annots: Square, Circle, Line, Ink, FreeText.

---

## Annot Methods

### destroy

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Destroys the **annot**, removing it from the page. The object becomes invalid.

**Parameters**

None

**Returns**

Nothing

**Example**

```
// remove all "FreeText" annotations on page 0
var annots = this.getAnnots({ nPage:0 });
for (var i = 0; i < annots.length; i++)
    if (annots[i].type == "FreeText") annots[i].destroy();
```

**getProps**

5.0	Ⓓ		Ⓐ	ⓧ
-----	---	--	---	---

Get the collected properties of an **annot**. Can be used to copy an annotation.

**Parameters**

None

**Returns**

This method returns an object literal of the properties of the annotation. The object literal is just like the one passed to [addAnnot](#).

**Example 1**

```
var annot = this.addAnnot({
    page: 0,
    type: "Text",
    rect: [40, 40, 140, 140]
});

// Make a copy of the properties of annot
var copy_props = annot.getProps();

// Now create a new annot with the same properties on every page
var numpages = this.numPages;
for (var i=0; i < numpages; i++) {
    var copy_annot = this.addAnnot(copy_props);
    // but move it to page i
    copy_annot.page=i;
}
```

**Example 2**

Display all properties and values of an annot.

```
var a = this.getAnnots(0);    // get all annots on page 0
if ( a != null ) {
```

```

var p = a[0].getProps(); // get the properties of first one
for ( o in p ) console.println( o + " : " + p[o] );
}

```

## getStateInModel

6.0	Ⓓ			
-----	---	--	--	--

Gets the current state of the **annot** in the context of a state model. See also [transitionToState](#).

### Parameters

<b>cStateModel</b>	The state model to determine the state of the <b>annot</b> .
--------------------	--

### Returns

The result is an array of the identifiers for the current state of the **annot**.

- If the state model was defined to be exclusive then there will only be a single state (or no states if the state has not been set).
- If the state model is non-exclusive then there may be multiple states. The array will have no entries if the state has not been set and there is no default.

### Exceptions

None

### Example

Report on the status of all annots on all pages of this document.

```

annots = this.getAnnots()
for ( var i= 0; i< annots.length; i++) {
    states = annots[i].getStateInModel("Review");
    if ( states.length > 0 ) {
        for(j = 0; j < states.length; j++)
        {
            var d = util.printd(2, states[j].modDate);
            var s = states[j].state;
            var a = states[j].author;

            console.println(annots[i].type + ": " + a + " "
                + s + " " + d + "on page "
                + (annots[i].page+1) );
        }
    }
}

```

## setProps

5.0	Ⓓ		Ⓒ	ⓧ
-----	---	--	---	---

Sets many properties of the annotation simultaneously.

### Parameters

---

<b>objectLiteral</b>	A generic object, which specifies the <b>properties</b> of the <b>annot</b> object annotation, such as <b>type</b> , <b>rect</b> , and <b>page</b> , to be created. (This is the same as the parameter of <b>doc.addAnnot</b> .)
----------------------	--

---

### Returns

The **annot** object

### Example

```
var annot = this.addAnnot({type: "Line"})
annot.setProps({
  page: 0,
  points: [[10,40], [200,200]],
  strokeColor: color.red,
  author: "A. C. Robat",
  contents: "Check with Jones on this point.",
  popupOpen: true,
  popupRect: [200, 100, 400, 200], // place rect at tip of the arrow
  arrowBegin: "Diamond",
  arrowEnd: "OpenArrow"
});
```

## transitionToState

6.0	Ⓓ			
-----	---	--	--	--

Makes the state of the Annot **cState** by performing a state transition. The state transition is recorded in the audit trail of the Annot.

See also [getStateInModel](#).

**NOTE:** For the states to work correctly in a multi-user environment, all users need to have the same state model definitions; therefore, it is best to place state model definitions in a folder-level JavaScript file which can be distributed to all users, or installed on all systems.

**Parameters**

<b>cStateModel</b>	The state model in which to perform the state transition. <b>cStateModel</b> must have been previously added by calling <a href="#">addStateModel</a> .
<b>cState</b>	A valid state in the state model to transition to.

**Returns**

Nothing

**Exceptions**

None

**Example**

```

try {
    // Create a document
    var myDoc = app.newDoc();
    // Create an annot
    var myAnnot = myDoc.addAnnot
    ({
        page: 0,
        type: "Text",
        point: [300,400],
        name: "myAnnot",
    });

    // Create the state model
    var myStates = new Object();
    myStates["initial"] = {cUIName: "Haven't reviewed it"};
    myStates["approved"] = {cUIName: "I approve"};
    myStates["rejected"] = {cUIName: "Forget it"};
    myStates["resubmit"] = {cUIName: "Make some changes"};
    Collab.addStateModel({
        cName: "ReviewStates",
        cUIName: "My Review",
        oStates: myStates,
        cDefault: "initial"
    });
} catch(e) { console.println(e); }

// Change the states
myAnnot.transitionToState("ReviewStates", "resubmit");
myAnnot.transitionToState("ReviewStates", "approved");

```

## Annot3D Object

An Annot3D object represents a particular Acrobat 3D annotation; that is, an annotation created using the Acrobat **3D Tool**. The Annot3D object can be acquired from the Doc object methods [getAnnot3D](#) and [getAnnots3D](#).

## Annot3D Properties

### activated

7.0				
-----	--	--	--	--

A boolean that indicates whether the annot is displaying the 3D artwork (**true**), or just the posterboard picture (**false**).

See [context3D](#) below.

Type: Boolean

Access: R/W.

### context3D

7.0				
-----	--	--	--	--

If **Annot3D.activated** is **true**, this property returns the context of the 3D Annot (a SceneContext3d object) containing the 3D scene, and returns **undefined** if **activated** is **false**.

Type: SceneContext3d object

Access: R.

### innerRect

7.0				
-----	--	--	--	--

The **innerRect** property returns an array of four numbers  $[x_{ll}, y_{ll}, x_{ur}, y_{ur}]$  specifying the lower-left x, lower-left y, upper-right x and upper-right y coordinates, in the coordinate system of the annotation (lower-left is  $[0, 0]$ , top right is  $[width, height]$ ), of the 3D annot's 3DB box, where the 3D artwork will be rendered.

Type: Array

Access: R.



**name**

7.0				
-----	--	--	--	--

The name of the annotation.

Type: String                      Access: R.

**page**

7.0				
-----	--	--	--	--

The **page** property is the 0-based page number of the page the annot occurs on.

Type: Integer                      Access: R.

**rect**

7.0				
-----	--	--	--	--

The rect property returns an array of four numbers [x<sub>ll</sub>, y<sub>ll</sub>, x<sub>ur</sub>, y<sub>ur</sub>] specifying the lower-left x, lower-left y, upper-right x and upper-right y coordinates, in default user space, of the rectangle defining the location of the annotation on the page. (Array, RW)

Type: Array                      Access: R.

**App Object**

A static JavaScript object that defines a number of Acrobat specific functions plus a variety of utility routines and convenience functions.


**App Properties**

**activeDocs**

5.0		Ⓢ	
-----	--	---	--

Returns an array containing the [Doc Object](#) for each active document open in the viewer, see note below. If no documents are active, **activeDocs** returns nothing, or has the same behavior as **d = new Array (0)** in core JavaScript.

Beginning with Acrobat 7.0, when the script `d = app.activeDocs` is executed in the console, there is no `toString()` value that is output to the console. (In previous versions of Acrobat, executing this script returned `[object Global]` to the console.)

**NOTES:** (Security 

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

The array returned by `app.activeDocs` will include any documents opened by `app.openDoc` with the `bHidden` parameter set to `true`, subject to the security restrictions described above.

Type: Array

Access: R.

### Example

This example searches among the open documents for the document with a title of "myDoc", then it inserts a button in that document using `addField`. Whether the documents need to be `disclosed` depends on the version of Acrobat executing this code, and on the placement of the code (for example, console versus MouseUp action).

```
var d = app.activeDocs;
for (var i=0; i < d.length; i++)
if (d[i].info.Title == "myDoc") {
    var f = d[i].addField("myButton", "button", 0 , [20, 100, 100, 20]);
    f.setAction("MouseUp", "app.beep(0)");
    f.fillColor=color.gray;
}
```

### calculate



If set to `true`, allows calculations to be performed. If set to `false`, prevents all calculations in all documents from occurring. Its default value is `true`.

See `doc.calculate` which supersedes this property in later versions.

Type: Boolean

Access: R/W.

## constants

7.0				
-----	--	--	--	--

Each instance of an [App Object](#) inherits the **constants** property, which is a wrapper object for holding various constant values. The **constants** property returns an object with a single property, **align**. **app.constants.align** is an object which has the following properties. The values stored in this (**align**) object can be used to specify alignment, such as when adding a watermark.

---

**app.constants.align Object**

Property	Description
left	Designates left alignment
center	Designates center alignment
right	Designates right alignment
top	Designates top alignment
bottom	Designates bottom alignment

---

Type: Object

Access: R.

### Example

See `doc.addWatermarkFromFile` and `doc.addWatermarkFromText` for examples.

## focusRect

4.05	Ⓟ		
------	---	--	--

Turns the focus rectangle on and off. The focus rectangle is the faint dotted line around buttons, check boxes, radio buttons, and signatures to indicate that the form field has the keyboard focus. A value of **true** turns on the focus rectangle.

Type: Boolean

Access: R/W.

### Example

```
app.focusRect = false; // don't want faint dotted lines around fields
```

## formsVersion

4.0			
-----	--	--	--

The version number of the forms software running inside the viewer. Use this method to determine whether objects, properties, or methods in newer versions of the software are available if you wish to maintain backwards compatibility in your scripts.

*Type: Number*

*Access: R.*

### Example

```
if (typeof app.formsVersion != "undefined" && app.formsVersion >= 5.0)
{
    // Perform version specific operations here.
    // For example, toggle full screen mode
    app.fs.cursor = cursor.visible;
    app.fs.defaultTransition = "";
    app.fs.useTimer = false;
    app.fs.isFullScreen = !app.fs.isFullScreen;
}
else app.fullscreen = !app.fullscreen;
```

## fromPDFConverters

6.0				
-----	--	--	--	--

Returns an array of file type conversion ID strings. A conversion ID string is passed to `doc.saveAs`.

*Type: Array*

*Access: R.*

### Example

List all currently supported conversion ID strings for `doc.saveAs`.

```
for ( var i = 0; i < app.fromPDFConverters.length; i++)
    console.println(app.fromPDFConverters[i]);
```

## fs

5.0	Ⓟ		
-----	---	--	--

Returns the [FullScreen Object](#), which can be used to access the fullscreen properties.

*Type: object*

*Access: R.*

### Example

```
// This code puts the viewer into fullscreen (presentation) mode.
```

```
app.fs.isFullScreen = true;
```

See also `fullScreenObject.isFullScreen`.

## fullscreen

ⓧ			
---	--	--	--

Puts the Acrobat viewer in fullscreen mode vs. regular viewing mode.

See `fullScreenObject.isFullScreen`; this property supersedes this property in later versions. See also `fs`, which returns a `FullScreen Object` which can be used to access the fullscreen properties.

**NOTE:** A PDF document being viewed from within a web browser cannot be put into fullscreen mode. Fullscreen mode can, however, be initiated from within the browser, but will not occur unless there is a document open in the Acrobat viewer application; in this case, the document open in the viewer will appear in fullscreen, not the PDF document open in the web browser.

Type: Boolean

Access: R/W.

### Example

```
// on mouse up, set to fullscreen mode
app.fullscreen = true;
```

In the above example, the Adobe Acrobat viewer is set to fullscreen mode when `app.fullscreen` is set to `true`. If `app.fullscreen` was `false` then the default viewing mode would be set. The default viewing mode is defined as the original mode the Acrobat application was in before full screen mode was initiated.

## language

3.01			
------	--	--	--

Defines the language of the running Acrobat Viewer. It returns the following strings:

String	Language
CHS	Chinese Simplified
CHT	Chinese Traditional
DAN	Danish
DEU	German
ENU	English
ESP	Spanish

String	Language
FRA	French
ITA	Italian
KOR	Korean
JPN	Japanese
NLD	Dutch
NOR	Norwegian
PTB	Brazilian Portuguese
SUO	Finnish
SVE	Swedish

Type: String

Access: R.

## media

6.0				
-----	--	--	--	--

The global [App.media Object](#) defines an extensive number of properties and methods useful for setting up and controlling multimedia player.

See the section on the [App.media Object](#) for a listing of the properties and methods of this object, as well as numerous examples of use.

Type: Object

Access: R/W.

## monitors

6.0				
-----	--	--	--	--

The `app.monitors` property returns a [Monitors Object](#), which is an array containing one or more Monitor objects representing each of the display monitors connected to the user's system. Each access to `app.monitors` returns a new, up to date copy of this array.

A Monitors object also has several methods that can be used to select a display monitor, or JavaScript code can look through the array explicitly. See the [Monitors Object](#) for details.

Type: [Monitors Object](#)

Access: R.

### Example

Count the number of display monitors connected to the user's system.

```
var monitors = app.monitors;
console.println("There are " + monitors.length
    + " monitor(s) connected to this system.");
```

## numPlugIns

ⓧ			
---	--	--	--

Indicates the number of plug-ins that have been loaded by Acrobat. See [plugIns](#) which supersedes this property in later versions.

*Type: Number*

*Access: R.*

## openInPlace

4.0	Ⓟ		
-----	---	--	--

Determines whether cross-document links are opened in the same window or opened in a new window.

*Type: Boolean*

*Access: R/W.*

### Example

```
app.openInPlace = true;
```

## platform

4.0			
-----	--	--	--

Returns the platform that the script is currently executing on. Valid values are

WIN  
MAC  
UNIX

*Type: String*

*Access: R.*

## plugIns

5.0			
-----	--	--	--

Determines which plug-ins are currently installed in the viewer. Returns an array of [PlayerInfo Objects](#).

*Type: Array*

*Access: R.*

**Example**

```
// Get array of PlugIn Objects
var aPlugins = app.plugins;
// Get number of plugins
var nPlugins = aPlugins.length;
// Enumerate names of all plugins
for ( var i = 0; i < nPlugins; i++)
    console.println("Plugin \#" + i + " is " + aPlugins[i].name);
```

**printColorProfiles**

6.0			X	
-----	--	--	---	--

Returns a list of available printer color spaces. Each of these values is suitable to use as the value of the **printParams.colorProfile**.

Type: Array of Strings      Access: R.

**Example**

Print out a listing of available printer color spaces.

```
var l = app.printColorProfiles.length
for ( var i = 0; i < l; i++)
    console.println("(" + (i+1) + ") " + app.printColorProfiles[i]);
```

**printerNames**

6.0				
-----	--	--	--	--

Returns a list of available printers. Each of these values is suitable to use in **printParams.printerName**. If no printers are installed on the system an empty array is returned.

Type: Array of Strings      Access: R.

**Example**

Print out a listing of available printer color spaces.

```
var l = app.printerNames.length
for ( var i = 0; i < l; i++)
    console.println("(" + (i+1) + ") " + app.printerNames[i]);
```

**runtimeHighlight**

6.0	P			
-----	---	--	--	--

If **true**, the background color and hover color for form fields are shown.



Type: Boolean

Access: R/W.

**Example**

If runtime highlighting is off (**false**) do nothing, else, change the preferences.

```
if (!app.runtimeHighlight)
{
    app.runtimeHighlight = true;
    app.runtimeHighlightColor = color.red;
}
```

**runtimeHighlightColor**

6.0	Ⓟ			
-----	---	--	--	--

Sets the color for runtime highlighting of form fields.

The value of **runtimeHighlightColor** is a color array, see the [Color Object](#) for details.

Type: A color array

Access: R/W.

**Example**

See the example following [runtimeHighlight](#).

**thermometer**

6.0			
-----	--	--	--

Returns a [Thermometer Object](#). The **thermometer** object is a combined status window/progress bar that indicates to the user that a lengthy operation is in progress.

Type: object

Access: R.

**Example**

See the [Thermometer Object](#) for an example.

**toolbar**

3.01	Ⓟ		
------	---	--	--

Allows a script to show or hide both the horizontal and vertical Acrobat tool bars. It does not hide the tool bar in external windows (that is, in an Acrobat window within a Web browser).

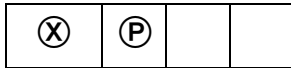
Type: Boolean

Access: R/W.

**Example**

```
// Opened the document, now remove the toolbar.
app.toolbar = false;
```

**toolbarHorizontal**



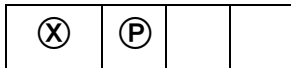
Allows a script to show or hide the Acrobat horizontal tool bar. It does not hide the tool bar in external windows (that is, in an Acrobat window within a Web browser).

**NOTE:** Acrobat 5.0 drastically changed the notion of what a toolbar is and where it can live within the frame of the application. This property has therefore been deprecated. If accessed, it acts like [toolbar](#).

Type: Boolean

Access: R/W.

**toolbarVertical**



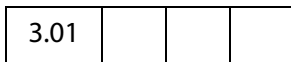
Allows a script to show or hide the Acrobat vertical tool bar. It does not hide the tool bar in external windows (that is, in an Acrobat window within a Web browser).

**NOTE:** Acrobat 5.0 drastically changed the notion of what a toolbar is and where it can live within the frame of the application. This property has therefore been deprecated. If accessed, it acts like [toolbar](#).

Type: Boolean

Access: R/W.

**viewerType**



This property returns a string that indicates the viewer the application that is running. The values are given in the table below.

Value	Description
Reader	Acrobat Reader 5.0 or earlier / Adobe Reader 5.1 or later
Exchange	Adobe Acrobat prior to 6.0 / Acrobat Standard 6.0 or later
Exchange-Pro	Acrobat Professional 6.0 or later

Type: String

Access: R.

### viewerVariation

5.0			
-----	--	--	--

Indicates the packaging of the running Acrobat Viewer. Values are:

- Reader
- Fill-In
- Business Tools
- Full

Type: String

Access: R.

### viewerVersion

4.0			
-----	--	--	--

Indicates the version number of the current viewer.

Type: Number

Access: R.

## App Methods

### addMenuItem

5.0		Ⓢ	
-----	--	---	--

Adds a menu item to the application.

**NOTE:** (SecurityⓈ): This method can only be executed during application initialization or console events. See the [Event Object](#) for a discussion of Acrobat JavaScript events.

See also the [addSubMenu](#), [execMenuItem](#), [hideMenuItem](#), and [listMenuItems](#).

#### Parameters

<b>cName</b>	The language independent name of the menu item. This language independent name is used to access the menu item for other methods (for example, <a href="#">hideMenuItem</a> ).
<b>cUser</b>	(optional) The user string (language dependent name) to display as the menu item name. If <b>cUser</b> is not specified then <b>cName</b> is used

---

<b>cParent</b>	<p>The name of the parent menu item. Its submenu will have the new menu item added to it. If <b>cParent</b> has no submenu then an exception is thrown.</p> <p>Menu item names can be discovered with <a href="#">listMenuItems</a>.</p>
<b>nPos</b>	<p>(optional) The position within the submenu to locate the new menu item. The default behavior is to append to the end of the submenu. Specifying <b>nPos</b> as 0 will add to the top of the submenu. Beginning with Acrobat 6.0, the value of <b>nPos</b> can also be the language independent name of a menu item.</p> <p>(Version 6.0) If the value <b>nPos</b> is a string, this string is interpreted as a named item in the menu (a language independent name of a menu item). The named item determines the position at which the new menu item is to be inserted. See <b>bPrepend</b> for additional details.</p> <p><b>NOTE:</b> The <b>nPos</b> parameter is ignored in certain menus that are alphabetized. The alphabetized menus are</p> <ul style="list-style-type: none"><li>● The first section of <b>View &gt; Navigational Tabs</b>.</li><li>● The first section of <b>View &gt; Toolbars</b>.</li><li>● The first section of the <b>Advanced</b> submenu.</li></ul> <p><b>NOTE:</b> When <b>nPos</b> is a number, <b>nPos</b> is not obeyed in the <b>Tools</b> menu. A menu item introduced into the <b>Tools</b> menu comes in at the top of the menu. <b>nPos</b> will be obeyed when <b>nPos</b> is a string referencing another user-defined menu item.</p>
<b>cExec</b>	<p>An expression string to evaluate when the menu item is selected by the user.</p> <p><b>NOTE:</b> Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged. See the note <a href="#">“Introduced in Acrobat 7.0” on page 647</a> for more details.</p>
<b>cEnable</b>	<p>(optional) An expression string that determines whether or not to enable the menu item. The default is that the menu item is always enabled. This expression should set <b>event.rc</b> to <b>false</b> to disable the menu item.</p>
<b>cMarked</b>	<p>(optional) An expression string that determines whether or not the menu item has a check mark next to it. Default is that the menu item is not marked. This expression should set <b>event.rc</b> to <b>false</b> to uncheck the menu item and true to check it.</p>

---

---

<b>bPrepend</b>	<p>(optional, version 6.0) Determines the position of the new menu item relative to the position specified by <b>nPos</b>. The default value is <b>false</b>. If <b>bPrepend</b> is <b>true</b>, the rules for insertion are as follows: If <b>nPos</b> is a string, the new item is placed before the named item; if <b>nPos</b> is a number, the new item is placed before the numbered item; if the named item can't be found or <b>nPos</b> is not between zero and the number of items in the list, inclusive, then the new item is inserted as the first item in the menu (rather than at the end of the menu).</p> <p><b>bPrepend</b> is useful when the named item is the first item in a group.</p>
-----------------	--

---

**Returns**

Nothing

**Example 1**

```
// This example adds a menu item to the top of the file submenu that
// puts up an alert dialog displaying the active document title.
// This menu is only enabled if a document is opened.
app.addItem({ cName: "Hello", cParent: "File",
              cExec: "app.alert(event.target.info.title, 3);",
              cEnable: "event.rc = (event.target != null);",
              nPos: 0
            });
```

**Example 2 (version 6.0)**

Place a two menu items in the "File" menu, one *before* the "Close" item, and the other *after* the "Close" item.

```
// insert after the "Close" item (the default behavior)
app.addItem( { cName: "myItem1", cUser: "My Item 1", cParent:
              "File", cExec: "_myProc1()", nPos: "Close"});
// insert before the "Close" item, set bPrepend to true.
app.addItem( { cName: "myItem2", cUser: "My Item 2", cParent:
              "File", cExec: "_myProc2()", nPos: "Close", bPrepend: true });
```

**addSubMenu**

5.0		Ⓢ	
-----	--	---	--

Adds a menu item with a submenu to the application.

See also the [addItem](#), [execMenuItem](#), [hideMenuItem](#), and [listMenuItems](#).

**NOTE:** (SecurityⓈ): This method can only be executed during application initialization or console events. See the [Event Object](#) for a discussion of Acrobat JavaScript events.

**Parameters**

<b>cName</b>	The language independent name of the menu item. This language independent name is used to access the menu item for <a href="#">hideMenuItem</a> , for example.
<b>cUser</b>	(optional) The user string (language dependent name) to display as the menu item name. If <b>cUser</b> is not specified then <b>cName</b> is used.
<b>cParent</b>	The name of the parent menu item to receive the new submenu. Menu item names can be discovered with <a href="#">listMenuItems</a> .
<b>nPos</b>	(optional) The position within the parent's submenu to locate the new submenu. Default is to append to the end of the parent's submenu. Specifying <b>nPos</b> as <b>0</b> will add to the top of the parent's submenu.  <b>NOTE:</b> The <b>nPos</b> parameter is ignored in certain menus that are alphabetized. The alphabetized menus are <ul style="list-style-type: none"> <li>• The first section of <b>View &gt; Navigational Tabs</b>.</li> <li>• The first section of <b>View &gt; Toolbars</b>.</li> <li>• The first section of the <b>Advanced</b> submenu.</li> </ul> <b>NOTE:</b> When <b>nPos</b> is a number, <b>nPos</b> is not obeyed in the <b>Tools</b> menu. A menu item introduced into the <b>Tools</b> menu comes in at the top of the menu. <b>nPos</b> will be obeyed when <b>nPos</b> is a string referencing another user defined menu item.

**Returns**

Nothing

**Example**See [newDoc](#).**addToolButton**

6.0				
-----	--	--	--	--

Adds a tool button to the "Add-on" toolbar of Acrobat.

**NOTES:** (Version 7.0) A number of changes have been made with regard to the secure use of this method. Execution of `addToolButton` in the console and app initialization is

considered *privileged* execution and is trusted. The tool button will be added to the “Add-on” toolbar, which is dockable.

If this method is called from *nonprivileged* script, the warning “JavaScript Window” will appear on the add-on toolbar, which will *not be dockable*. (See [Privileged versus Non-privileged Context](#).)

If there is an active document, `docA.pdf`, for example, open in Acrobat when this method is called to add a tool button, Acrobat will remove the tool button when `docA.pdf` is either no longer active, or is closed; in the former case, the tool button will be automatically added to the tool bar should `docA.pdf` become the active document again.

The icon size is restricted to 20 by 20 pixels. If an icon of larger dimensions is used, an exception is thrown.

See also [removeToolButton](#).

## Parameters

<b>cName</b>	A unique language independent identifier for the tool button. The language independent name is used to access the toolbutton for other methods (for example, <a href="#">removeToolButton</a> ).  <b>NOTE:</b> The value of <b>cName</b> must be unique. To avoid a name conflict, check <a href="#">listToolbarButtons</a> , which lists all toolbar button names currently installed.
<b>oIcon</b>	A <a href="#">Icon Stream Generic Object</a> . (version 7.0) Beginning with version 7.0, the <b>oIcon</b> parameter is optional if a <b>cLabel</b> is provided.
<b>cExec</b>	The expression string to evaluate when the tool button is selected.
<b>cEnable</b>	(optional) An expression string that determines whether or not to enable the tool button. The default is that the tool button is always enabled. This expression should set <b>event.rc</b> to <b>false</b> to disable the toolbutton.
<b>cMarked</b>	(optional) An expression string that determines whether or not the tool button is marked. The default is that the tool button is not marked. This expression should set <b>event.rc</b> to <b>true</b> to mark the toolbutton.
<b>cTooltext</b>	(optional) The text to display in the toolbutton help text when the user mouses over the toolbutton. The default is to not have a tool tip.  <b>NOTE:</b> Avoid the use of extended characters in the <b>cTooltext</b> string as the string may be truncated.

<b>nPos</b>	(optional) The Toolbutton number to place the added Toolbutton before in the Toolbar. If <b>nPos</b> is <b>-1</b> (the default) then the Toolbutton is appended to the Toolbar.
<b>cLabel</b>	(optional, version 7.0) A text label to be displayed on the button to the right of the icon. The default is to not have a label.

**Returns**

An integer.

**Exceptions**

None

**Example**

In this example, a toolbutton is created from a icon graphic on the user's hard drive. This script is executed from the console.

```
// Create a document
var myDoc = app.newDoc();

// import icon (20x20 pixels) from the file specified
myDoc.importIcon("myIcon", "/C/myIcon.jpg", 0);

// convert the icon to a stream.
oIcon = util.iconStreamFromIcon(myDoc.getIcon("myIcon"));

// close the doc now that we have grabbed the icon stream
myDoc.closeDoc(true);

// add a tool button
app.addToolButton({
    cName: "myToolButton",
    oIcon: oIcon,
    cExec: "app.alert('Someone pressed me!')",
    cTooltext: "Push Me!",
    cEnable: true,
    nPos: 0
});

app.removeToolButton("myToolButton")
```

See also [util.iconStreamFromIcon](#), and the example that follows.

**alert**

3.01			
------	--	--	--

Displays an alert dialog on the screen.



## Parameters

<b>cMsg</b>	A string containing the message to be displayed.
<b>nIcon</b>	(optional) An icon type. Values are associated with icons as follows: 0: Error (default) 1: Warning 2: Question 3: Status  <b>NOTE:</b> The Macintosh OS does not distinguish between warnings and questions, so it only has three different types of icons.
<b>nType</b>	(optional) A button group type. Values are associated with button groups as follows: 0: OK (default) 1: OK, Cancel 2: Yes, No 3: Yes, No, Cancel
<b>cTitle</b>	(optional, version 6.0) A title of the dialog. If not specified the title "Adobe Acrobat" is used.
<b>oDoc</b>	(optional, version 6.0) The <a href="#">Doc Object</a> that the alert should be associated with.
<b>oCheckbox</b>	(optional, version 6.0) If this parameter is passed, a checkbox is created in the lower left region of the alert box. <b>oCheckbox</b> is a generic JS object having three properties. The first two property values are passed to the <b>alert ()</b> method, the third property returns a boolean. <ul style="list-style-type: none"> <li>● <b>cMsg</b> (optional): A string to display with the checkbox. If not specified, the default string is "Do not show this message again".</li> <li>● <b>bInitialValue</b> (optional): If <b>true</b>, the initial state of the checkbox is checked. Default is <b>false</b>.</li> <li>● <b>bAfterValue</b>: When the alert method exits, contains the state of the checkbox when the dialog closed. If <b>true</b>, the checkbox was checked when the alert box is closed.</li> </ul>

## Returns

**nButton**, the type of the button that was pressed by the user:

- 1: OK
- 2: Cancel
- 3: No
- 4: Yes

**Example 1**

A simple alert box notifying the user.

```
app.alert({
  cMsg: "Error! Try again!",
  cTitle: "Acme Testing Service"
});
```

**Example 2**

Close the document with the user's permission

```
// A MouseUp action
var nButton = app.alert({
  cMsg: "Do you want to close this document?",
  cTitle: "A message from A. C. Robot",
  nIcon: 2, nType: 2
});
if ( nButton == 4 ) this.closeDoc();
```

**Example 3 (Version 6.0)**

One doc creates an alert box in another doc. Suppose there are two documents, DocA and DocB. One document is open in a browser and other in the viewer.

```
// The following is a declaration at the document level in DocA
var myAlertBoxes = new Object;
myAlertBoxes.oMyCheckbox = {
  cMsg: "Care to see this message again?",
  bAfterValue: false
}
```

The following is a MouseUp action in DocA. The variable theOtherDoc is the Doc object of DocB. The alert box ask the user if the user wants to see this alert box again. If the user clicks on the check box provided, the alert does not appear again.

```
if ( !myAlertBoxes.oMyCheckbox.bAfterValue )
{
  app.alert({
    cMsg: "This is a message from the DocA?",
    cTitle: "A message from A. C. Robot",
    oDoc:theOtherDoc,
    oCheckbox: myAlertBoxes.oMyCheckbox
  });
}
```

**beep**

3.01			
------	--	--	--

Causes the system to play a sound.

**NOTE:** On Apple Macintosh and UNIX systems the beep type is ignored.

**Parameters**


---

<b>nType</b>	(optional) The sound type. Values are associated with sounds as follows: 0: Error 1: Warning 2: Question 3: Status 4: Default (default value)
--------------	--

---

**Returns**

Nothing

**beginPriv**

7.0				
-----	--	--	--	--

If allowed, raises the execution privilege of the current stack frame such that methods marked “secure” will execute without security exceptions. This method will only succeed if there is a frame on the stack representing the execution of a trusted function and any and all frames (including the frame making the call) between the currently executing frame and that frame represent the execution of trust propagator functions.

Use `app.endPriv` to revoke privilege. The method `app.trustedFunction` can be used to create a trusted function, and `app.trustPropagatorFunction` to create a trust propagator function. The term “stack frame” is discussed following the description of `app.trustedFunction`.

**Parameters**

None

**Returns**Returns `undefined` on success, exception on failure.**Example**

For examples of usage, see `trustedFunction` and `trustPropagatorFunction`.

**browseForDoc**

7.0		Ⓢ		
-----	--	---	--	--

This method presents a file system browser and returns an object containing information concerning the user’s response.

**NOTE:** (SecurityⓈ): This method can only be executed during batch or console events. See the `Event Object` for a discussion of Acrobat JavaScript events.

**Parameters**

<b>bSave</b>	(optional) A boolean that, if <b>true</b> , specifies that the file system browser should be presented as it would for a save operation. The default is <b>false</b> .
<b>cFilenameInit</b>	(optional) A string that specifies the default file name for the file system browser to be populated with.
<b>cFSInit</b>	(optional) A string that specifies the file system that the file system browser will be operating on initially. Two values are supported: "" (the empty string) representing the default file system and "CHTTP". The default is the default file system. This parameter is only relevant if the web server supports WebDAV.

**Returns**

On success, returns an object which has three properties:

Property	Description
<b>cFS</b>	A string containing the resulting file system name for the chosen file.
<b>cPath</b>	A string containing the resulting path for the chosen file.
<b>cURL</b>	A string containing the resulting URL for the chosen file

If the user cancels, the return value is **undefined**. On error, throws an exception.

**Example 1**

Browse for a document and report back the results to console.

```
var oRetn = app.browseForDoc({
    cFilenameInit: "myComDoc.pdf",
    cFSInit: "CHTTP",
});
if ( typeof oRetn != "undefined" )
    for ( var o in oRetn )
        console.println( "oRetn." + o + "=" + oRetn[o] );
else console.println("User cancelled!");
```

If the user selects a file on a WebDAV server, a possible output of this code is given below:

```
oRetn.cFS=CHTTP
oRetn.cPath=http://www.myCom.com/WebDAV/myComDoc.pdf
oRetn.cURL=http://www.myCom.com/WebDAV/myComDoc.pdf
```

Should the user select a file in the default file system, a typical output of this code is given below:

```
oRetn.cFS=DOS
```

```
oRetn.cPath=/C/temp/myComDoc.pdf
oRetn.cURL=file:///C:/temp/myComDoc.pdf
```

The script can go on to open the selected file using `app.openDoc`.

```
var myURL = (oRetn.cFS=="CHTTP") ? encodeURIComponent(oRetn.cPath) : oRetn.cPath;
var oDoc = app.openDoc({cPath: myURL, cFS: oRetn.cFS});
```

**NOTE:** `app.openDoc` requires `cPath` to be an escaped string when retrieving a file from a WebDAV server. See `app.openDoc` for a brief description and example.

## Example 2

Browse and save a document.

```
var oRetn = app.browseForDoc({
  bSave: true,
  cFilenameInit: "myComDoc.pdf",
  cFSInit: "CHTTP",
});
if ( typeof oRetn != "undefined" ) this.saveAs({
  cFS: oRetn.cFS, cPath: oRetn.cPath, bPromptToOverwrite: false});
```

## clearInterval

5.0			
-----	--	--	--

Cancels a previously registered interval, `oInterval`, initially set by `setInterval`.

See also `setTimeout` and `clearTimeout`.

### Parameters

<code>oInterval</code>	The registered interval to cancel.
------------------------	------------------------------------

### Returns

Nothing

### Example

See `setTimeout`.

## clearTimeout

5.0			
-----	--	--	--

Cancels a previously registered time-out interval, `oTime`. Such an interval is initially set by `setTimeout`.

See also `setInterval` and `clearInterval`.

**Parameters**


---

<code>oTime</code>	The previously registered time-out interval to cancel.
--------------------	--

---

**Returns**

Nothing

**Example**See [setTimeout](#).**endPriv**

7.0				
-----	--	--	--	--

Revokes any privilege bestowed upon the current stack frame by `app.beginPriv`. Will not revoke privilege bestowed by the current event.

Related methods are `app.trustedFunction`, `app.trustPropagatorFunction` and `app.beginPriv`.

**Parameters**

None

**Returns**Returns `undefined` on success, exception on failure.**Example**For examples of usage, see [trustedFunction](#) and [trustPropagatorFunction](#).**execDialog**

7.0				
-----	--	--	--	--

This method presents a modal dialog to the user. Modal dialogs require the user to dismiss the dialog before the host application can be directly used again.

The dialog description is provided by the `monitor` parameter, a generic object literal as described below. This object literal *dialog descriptor* consists of a set of handler functions which are called to handle various events for the dialog and a set of properties which describe the contents of the dialog.

Dialog items are identified by an *ItemID*, which is a unique 4 character string. An *ItemID* for an element is only necessary if it will be necessary to refer to that element elsewhere in the dialog description (e.g. to set/get a value for the element, to add a handler for the element, to set a tab order including the element).

**NOTE:** To distinguish a dialog from an Acrobat dialog from JavaScript, dialogs that are added at the document level will have a title of "JavaScript Dialog" and will display the text "Warning: JavaScript Dialog" at the bottom.

*Adobe Dialog Manager Programmer's Guide and Reference* explains in detail the modal dialog and its properties. See the section [Adobe Web Documentation](#) for a link to this document.

## Parameters

<b>monitor</b>	An object literal that is the dialog descriptor.
<b>inheritDialog</b>	(optional) This is a <a href="#">Dialog Object</a> which should be reused when displaying this dialog. This is useful for displaying a series of dialogs (such as a "Wizard") to prevent the previous dialog from disappearing before the new dialog is displayed. The default is to not reuse a dialog.
<b>parentDoc</b>	(optional) A <a href="#">Doc Object</a> to use as the parent for this dialog. The default is to parent the dialog to the Acrobat application.

## Returns

A string, which is the *ItemID* of the element that caused the dialog to be dismissed. The return value is "ok" or "cancel" if the dismissed is dismissed by the **ok** or **cancel** button.

**NOTE:** The `app.execDialog()` creates a modal dialog, and, as a consequence, the JavaScript debugger does not work while the dialog is active. Debugging is disabled.

## Dialog Handlers

The dialog handlers allow the caller to be notified when specific events occur for the dialog. Each handler is optional and is passed a [Dialog Object](#) that can be used to query or set values in the dialog. The supported Dialog handlers are listed in the table that follows.

Dialog Handlers	
Handler	Description
<b>initialize</b>	This method is called when the dialog is being initialized.
<b>validate</b>	This method is called when a field is modified to determine if the value is acceptable (by returning <b>true</b> ) or unacceptable (by returning <b>false</b> ).
<b>commit</b>	This method is called when the OK button of the dialog is hit.
<b>destroy</b>	This method is called when the dialog is being destroyed.

---

## Dialog Handlers

Handler	Description
<i>ItemID</i>	<p>This method will be called when the Dialog element <i>ItemID</i> is modified. In the case of a text box, this is when the text box loses focus. For other controls, it is when the selection changes.</p> <p>If <i>ItemID</i> is not a JavaScript identifier, then the name needs to be enclosed in double quotes when the method is defined, e.g.,</p> <pre>"bt:1": function () { .... }</pre> <p>The double quotes are optional in the case <i>ItemID</i> is a JavaScript identifier, e.g.,</p> <pre>butn": function () { .... }</pre> <pre>butn: function () { .... }</pre> <p>are both correct.</p>

---

## Description Property

The dialog description consists of a set of nested properties. The dialog is described in the **description** property, with sub-elements as an array of child object literals that form a tree.

The dialog properties which are set at the root level of the **description** property are listed in the table that follows.

---

### description Properties

Property	Type	Description
<b>name</b>	String	The title bar of the Dialog which should be localized.
<b>first_tab</b>	String	An <i>ItemID</i> for the dialog item which should be first in the tab order. This dialog item will also be active when the dialog is created. This property is required for setting up a tabbing order. See the <b>next_tab</b> property defined below.
<b>width</b>	Numeric	Specifies the width of the Dialog in pixels. If no width is specified, the combined width of the contents is used.
<b>height</b>	Numeric	Specifies the height of the Dialog in pixels. If no height is specified, the combined height of the contents is used.

---



---

**description Properties**

Property	Type	Description
<code>char_width</code>	Numeric	Specifies the width of the Dialog in characters. If no width is specified, the combined width of the contents is used.
<code>char_height</code>	Numeric	Specifies the height of the Dialog in characters. If no height is specified, the combined height of the contents is used.
<code>align_children</code>	String	Sets the alignment for all descendents. Must be one of the following values: <ul style="list-style-type: none"> <li>● "align_left": Left Aligned</li> <li>● "align_center": Center Aligned</li> <li>● "align_right": Right Aligned</li> <li>● "align_top": Top Aligned</li> <li>● "align_fill": Align to fill the parents width, may widen objects.</li> <li>● "align_distribute": Distribute the contents over the parents width.</li> <li>● "align_row": Distribute the contents over the parents width with a consistent baseline.</li> <li>● "align_offscreen": Align items one on top of another.</li> </ul>
<code>elements</code>	Array	An array of <i>object literals</i> which describe the dialog elements contained within this dialog (see <a href="#">Dialog elements</a> )

**Dialog elements**

The dialog `elements` are object literals with the following set of properties.

---

**Dialog elements Properties**

Property	Type	Description
<code>name</code>	String	The displayed name of the dialog element which should be localized.  <b>NOTE:</b> The <code>name</code> property is ignored for the <code>edit_text</code> type.
<code>item_id</code>	String	An <i>ItemID</i> for this dialog, a unique 4 character string.

Dialog elements Properties		
Property	Type	Description
<code>type</code>	String	The type of this dialog element. It must be one of the following: <ul style="list-style-type: none"> <li>● "button" - A push button.</li> <li>● "check_box" - A check box.</li> <li>● "radio" - A radio button.</li> <li>● "list_box" - A list box.</li> <li>● "heir_list_box" - A heirarchical list box.</li> <li>● "static_text" - A static text box.</li> <li>● "edit_text" - An editable text box.</li> <li>● "popup" - A popup control.</li> <li>● "ok" - An OK button.</li> <li>● "ok_cancel" - An OK and Cancel Button.</li> <li>● "ok_cancel_other" - An OK, Cancel and Other button.</li> <li>● "view" - A container for a set controls.</li> <li>● "cluster" - A frame for a set of controls.</li> <li>● "gap" - A place holder.</li> </ul>
<code>next_tab</code>	String	An <i>ItemID</i> for the next dialog item in the tab order.  <b>NOTE:</b> Tabbing does not stop at any dialog item that is not the target of the <code>next_tab</code> (or <code>first_tab</code> ) property. Tabbing should form a circular linked list.
<code>width</code>	Numeric	Specifies the width of the element in pixels. If no width is specified, the combined width of the contents is used.
<code>height</code>	Numeric	Specifies the height of the element in pixels. If no height is specified, the combined height of the contents is used.
<code>char_width</code>	Numeric	Specifies the width of the element in characters. If no width is specified, the combined width of the contents is used.
<code>char_height</code>	Numeric	Specifies the height of the element in characters. If no height is specified, the combined height of the contents is used.

<b>Dialog elements Properties</b>		
<b>Property</b>	<b>Type</b>	<b>Description</b>
<b>font</b>	String	The font to use for this element. Must be one of the following: <ul style="list-style-type: none"> <li>● "default" - Default Font</li> <li>● "dialog" - Dialog Font</li> <li>● "palette" - Palette (small) Font</li> </ul>
<b>bold</b>	Boolean	Specify if the font is bold.
<b>italic</b>	Boolean	Specify if the font is italic.
<b>alignment</b>	String	Sets the alignment for this element. Must be one of the following values: <ul style="list-style-type: none"> <li>● "align_left": Left Aligned</li> <li>● "align_center": Center Aligned</li> <li>● "align_right": Right Aligned</li> <li>● "align_top": Top Aligned</li> <li>● "align_fill": Align to fill the parents width, may widen objects.</li> <li>● "align_distribute": Distribute the contents over the parents width.</li> <li>● "align_row": Distribute the contents over the parents width with a consistent baseline.</li> <li>● "align_offscreen": Align items one on top of another.</li> </ul>
<b>align_children</b>	String	Sets the alignment for all descendents. Must be one of the following values: <ul style="list-style-type: none"> <li>● "align_left": Left Aligned</li> <li>● "align_center": Center Aligned</li> <li>● "align_right": Right Aligned</li> <li>● "align_top": Top Aligned</li> <li>● "align_fill": Align to fill the parents width, may widen objects.</li> <li>● "align_distribute": Distribute the contents over the parents width.</li> <li>● "align_row": Distribute the contents over the parents width with a consistent baseline.</li> <li>● "align_offscreen": Align items one on top of another.</li> </ul>

**Dialog elements Properties**

Property	Type	Description
<code>elements</code>	Array	An array of <i>object literals</i> which describe the <a href="#">Dialog elements</a> contained within this dialog element.

**Additional Attributes of some Dialog elements**

Some of the `element` types have additional attributes, as listed below.

**Dialog elements Properties**

element type	Property	Type	Description
<code>static_text</code>	<code>multiline</code>	Boolean	If <code>true</code> , this static text element is multiline.
<code>edit_text</code>	<code>multiline</code>	Boolean	If <code>true</code> , this static text element is multiline.
	<code>readonly</code>	Boolean	If <code>true</code> , this text element is readonly. <b>NOTE:</b> This property is ignored when <code>password</code> is set to <code>true</code> .
	<code>password</code>	Boolean	If <code>true</code> , this text element is a password field.
	<code>PopupEdit</code>	Boolean	If <code>true</code> , this is a popup edit text element.
<code>radio</code>	<code>SpinEdit</code>	Boolean	If <code>true</code> , this is a spin edit text element.
	<code>group_id</code>	String	The group name to which this radio button belongs.
	<code>ok_name</code>	String	The name for the OK button
<code>ok, ok_cancel, ok_cancel_other</code>	<code>cancel_name</code>	String	The name for the cancel button
	<code>other_name</code>	String	The name for the other button

**Example 1**

The following dialog descriptor can be a document level or folder level JavaScript.

```
var dialog1 = {
```

```

initialize: function (dialog) {
    // create a static text containing the current date.
    var todayDate = dialog.store()["date"];
    todayDate = "Date: " + util.printd("mmmm dd, yyyy", new Date());
    dialog.load({ "date": todayDate });
},
commit: function (dialog) { // called when OK pressed
    var results = dialog.store();
    // now do something with the data collected, for example,
    console.println("Your name is " + results["fnam"]
        + " " + results["lnam"] );
},
description:
{
    name: "Personal Data", // dialog title
    align_children: "align_left",
    width: 350,
    height: 200,
    elements:
    [
        {
            type: "cluster",
            name: "Your Name",
            align_children: "align_left",
            elements:
            [
                {
                    type: "view",
                    align_children: "align_row",
                    elements:
                    [
                        {
                            type: "static_text",
                            name: "First Name: "
                        },
                        {
                            item_id: "fnam",
                            type: "edit_text",
                            alignment: "align_fill",
                            width: 300,
                            height: 20
                        }
                    ]
                }
            ]
        },
        {
            type: "view",
            align_children: "align_row",
            elements:
            [
                {

```

```

        type: "static_text",
        name: "Last Name: "
    },
    {
        item_id: "lnam",
        type: "edit_text",
        alignment: "align_fill",
        width: 300,
        height: 20
    }
]
},
{
    type: "static_text",
    name: "Date: ",
    char_width: 25,
    item_id: "date"
},
]
},
{
    alignment: "align_right",
    type: "ok_cancel",
    ok_name: "Ok",
    cancel_name: "Cancel"
}
]
}
};

```

Now, the following line can be executed from a mouse up action of a button, a menu action, and so on.

```
app.execDialog(dialog1);
```

### Example 2

The following example uses a check box and a radio button field. This code might be a document level JavaScript.

```

var dialog2 =
{
    initialize: function(dialog) {
        // set a default value for radio button field
        dialog.load({"rd01": true });
        this.hasPet = false;
        // disable radio button field
        dialog.enable({
            "rd01" : this.hasPet,
            "rd02" : this.hasPet,
            "rd03" : this.hasPet
        });
    },
},

```

```

commit: function(dialog) {
    // when user presses "Ok", this handler will execute first
    console.println("commit");
    var results = dialog.store();
    // do something with the data, for example,
    var hasPet = (this.hasPet) ? "have" : "don't have";
    console.println("You " + hasPet + " a pet.");
    if (this.hasPet)
        console.println("You have " + this.getNumPets(results)
            + " pet(s).");
},
getNumPets: function (results) {
    for ( var i=1; i<=3; i++) {
        if ( results["rd0"+i] ) {
            switch (i) {
                case 1:
                    var nPets = "one";
                    break;
                case 2:
                    var nPets = "two";
                    break;
                case 3:
                    var nPets = "three or more";
            }
        }
    }
};
return nPets;
},
ok: function(dialog) {
    // the handler for the Ok button will be handed after commit
    console.println("Ok!");
},
ckbx: function (dialog) {
    // process the checkbox, if user has pet, turn on radios
    this.hasPet = !this.hasPet;
    dialog.enable({
        "rd01" : this.hasPet,
        "rd02" : this.hasPet,
        "rd03" : this.hasPet
    });
},
cancel: function(dialog) { // handle handle the cancel button
    console.println("Cancel!");
},
other: function(dialog){ // handle the other button
    app.alert("Thanks for pressing me!");
    dialog.end("other"); // end the dialog, return "other"!
},
// The Dialog Description
description:
{

```

```
name: "More Personal Information",
elements:
[
  {
    type: "view",
    align_children: "align_left",
    elements:
    [
      {
        type: "static_text",
        name: "Personal Information",
        bold: true,
        font: "dialog",
        char_width: 30,
        height: 20
      },
      {
        type: "check_box",
        item_id: "ckbx",
        name: "Pet Owner"
      },
      {
        type: "view",
        align_children: "align_row",
        elements:
        [
          {
            type: "static_text",
            name: "Number of pets: "
          },
          {
            type: "radio",
            item_id: "rd01",
            group_id: "rado",
            name: "One"
          },
          {
            type: "radio",
            item_id: "rd02",
            group_id: "rado",
            name: "Two",
          },
          {
            type: "radio",
            item_id: "rd03",
            group_id: "rado",
            name: "Three or more",
          }
        ]
      }
    ]
  }
]
```



```

    ]
  },
  {
    type: "gap", //add a small vertical gap between
    height: 10 //..radio fields and buttons
  },
  {
    type: "ok_cancel_other",
    ok_name: "Ok",
    cancel_name: "Cancel",
    other_name: "Press Me"
  }
]
}
};

```

Now, the following line can be executed from a mouse up action of a button, a menu action, and so on.

```
var retn = app.execDialog(dialog2);
```

The value of **retn** will be "ok" if the ok button was pressed, "cancel" if the cancel button was pressed, and "other" if the other button was pressed, the one labeled "Press Me".

### Example 3

This example uses a list box.

```

var dialog3 = {
  // This dialog gets called when the dialog is created
  initialize: function(dialog) {
    this.loadDefaults(dialog);
  },
  // This dialog gets called when the OK button is hit.
  commit: function(dialog) {
    // See the Dialog Object for a description of how dialog.load
    // and dialog.store work.
    var elements = dialog.store()["subl"];
    // do something with the data.
  },
  // Callback for when the button "butn" is hit.
  butn: function(dialog) {
    var elements = dialog.store()["subl"]
    for(var i in elements) {
      if ( elements[i] > 0 ) {
        app.alert("You chose \" + i
          + "\", which has a value of " + elements[i] );
      }
    }
  },
  loadDefaults: function (dialog) {
    dialog.load({
      subl:

```



We then execute

```
app.execDialog(dialog3);
```

In the above example, if the line **type: "list\_box"** is replaced by **type: "popup"** and the **height** specification is removed, the example above will run with a popup control rather than a list box.

#### Example 4

In this example, the **hier\_list\_box** is illustrated. After the dialog is opened, a hierarchical list is presented. After a selection is made, and the user clicks on the Select buttons, the PDF jumps to destination chosen by the user. The doc object is passed to the dialog by making it a property of the dialog.

```
var dialog4 = {
  initialize: function(dialog) {
    dialog.load({
      subl:
        {
          "Chapter 1":
            {
              "Section 1":
                {
                  "SubSection 1": -1,
                  "SubSection 2": -2,
                },
              "Section 2":
                {
                  "SubSection 1": -3,
                  "SubSection 2": -4,
                }
            },
          "Chapter 3": -5,
          "Chapter 4": -6
        }
    })
  },
  subl: function(dialog) {
    console.println("Selection Box Hit");
  },
  getHierChoice: function (e)
  {
    if (typeof e == "object") {
      for ( var i in e ) {
        if ( typeof e[i] == "object" ) {
          var retn = this.getHierChoice(e[i]);
          if ( retn ) {
            retn.label = i + ", " + retn.label;
            return retn;
          }
        }
        // if e[i] > 0, then we've found the selected item
      } else if ( e[i] > 0 ) return { label:i, value: e[i] };
    }
  }
}
```

```

    }
  } else {
    if ( e[i] > 0 ) return e[i];
  }
},
butn: function (dialog)
{
  var element = dialog.store()["sub1"]
  var retn = this.getHierChoice(element);
  if ( retn ) {
    // write to the console the full name of the item selected
    console.println("The selection you've chosen is \"
      + retn.label + "\", its value is " + retn.value );
    dialog.end("ok");
    // this.doc is the doc object of this document
    this.doc.gotoNamedDest("dest"+retn.value);
  }
  else app.alert("Please make a selection, or cancel");
},
cncl: function (dialog) { dialog.end("cancel") },
// Dialog Description
description:
{
  name: "My Novel",
  elements:
  [
    {
      type: "view",
      align_children: "align_left",
      elements:
      [
        {
          type: "cluster",
          name: "Book Headings",
          elements:
          [
            {
              type: "static_text",
              name: "Make a selection",
            },
            {
              type: "hier_list_box",
              item_id: "sub1",
              char_width: 20,
              height: 200
            }
          ]
        }
      ],
    },
    {
      type: "view",
      align_children: "align_row",
    }
  ]
}

```

```

elements:
  [
    {
      type: "button",
      item_id: "cncl",
      name: "Cancel"
    },
    {
      item_id: "butn",
      type: "button",
      name: "Select"
    }
  ]
}
]
}
];

```

This function attaches the doc object to the dialog, then passes the dialog to the **app.execDialog()** method. The **dialog4** object and this function can be at the document level.

```

function dotheDialog(dialog,doc)
{
  dialog.doc = doc;
  var retn = app.execDialog( dialog )
}

```

Finally, we can execute the script below from a mouse up action, for example.

```
dotheDialog( dialog4, this );
```

### Example 5

See [Example 2](#) following **app.trustPropagatorFunction()**. This example illustrates how to executed privileged code from a non-privileged context.

## execMenuItem


4.0			
-----	--	--	--

Executes the specified menu item.

See also [addMenuItem](#), [addSubMenu](#), [hideMenuItem](#). Use [listMenuItems](#) to list the names of all menu items to the console.

Beginning with version 5.0, **app.execMenuItem("SaveAs")** can be called, subject to the restrictions described below. This saves the current file to the user's hard drive; a "SaveAs" dialog opens to ask the user to select a folder and file name. Executing the "SaveAs" menu item saves the current file as a linearized file, provided "Save As creates Fast

View Adobe PDF files" is checked in the **General** category of the **Edit > Preferences > General** dialog.

**NOTE:** (Security , version 7.0) In previous versions of Acrobat,

```
app.execMenuItem("SaveAs");
```

could only be executed during batch, console or menu events. Version 7.0 removes this restriction, **app.execMenuItem("SaveAs")** can be executed during a mouse up event, for example.

**NOTES:** If the user preferences are set to "Save As creates Fast View Adobe PDF files", do not expect a form object to survive a "SaveAs"; **Field Objects** are no longer valid, and an exception may be thrown when trying to access a field object immediately after a "SaveAs". See examples that follow.

For security reasons, scripts are not allowed to execute the **Quit** menu item. Beginning with Acrobat 6.0, scripts are not allowed to execute the **Paste** menu item.

## Parameters

<b>cMenuItem</b>	The menu item to execute. Menu item names can be discovered with <a href="#">listMenuItems</a> .
<b>oDoc</b>	(optional, version 7.0) <b>oDoc</b> is the document object of a document that is not hidden (see <a href="#">doc.hidden</a> ). If this parameter is present, <b>execMenuItem</b> executes the menu item in the document's context.

## Returns

Nothing

## Example 1

This example executes the **File > Open** menu item. It will display a dialog to the user asking for the file to be opened.

```
app.execMenuItem("Open");
```

## Example 2 (Acrobat 5.0)

```
var f = this.getField("myField");
// Assume preferences set to save linearized
app.execMenuItem("SaveAs");
// exception thrown, field not updated
f.value = 3;
```

## Example 3 (Acrobat 5.0)

```
var f = this.getField("myField");
// Assume preferences set to save linearized
app.execMenuItem("SaveAs");
// get the field again after the linear save
var f = getField("myField");
// field updated to a value of 3
```

```
f.value = 3;
```

## getNthPlugInName

ⓧ			
---	--	--	--

Obtains the name of the *n*th plug-in that has been loaded by the viewer. See also [numPlugIns](#).

See [plugIns](#) which supersedes this property in later versions.

### Parameters

---

<b>nIndex</b>	The <i>n</i> th plug-in loaded by the viewer.
---------------	---

---

### Returns

**cName**, the plug-in name that corresponds to **nIndex**.

## getPath

6.0		Ⓢ	
-----	--	---	--

This method returns the path to folders created during installation. A distinction is made between application folders and user folders. The method will throw a **GeneralError** exception (see [Error Objects](#)) if the path does not exist.

**NOTE:** (Security Ⓢ, version 7.0) This method can only be executed during batch or console events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

### Parameters

---

<b>cCategory</b>	(optional) Use this parameter to indicate the category of folder sought. The two values of <b>cCategory</b> are app user The default is <b>app</b> .
<b>cFolder</b>	(optional) A platform independent string that indicates the folder. The values of <b>cFolder</b> are root, eBooks, preferences, sequences, documents javascript, stamps, dictionaries, plugIns, spPlugIns help, temp, messages, resource, update The default is <b>root</b> .

---

**Returns**

The path to the folder determined by the parameters. An exception is thrown if the folder does not exist.

**Example 1**

Find the path to the user's Sequences folder

```
try {
    var userBatch = app.getPath("user", "sequences");
} catch(e) {
    var userBatch = "User has not defined any custom batch sequences";
}
console.println(userBatch);
```

**Example 2**

Create and save a document to My Documents on a windows platform.

```
var myDoc = app.newDoc();
var myPath = app.getPath("user", "documents") + "/myDoc.pdf"
myDoc.saveAs(myPath);
myDoc.closeDoc();
```

**goBack**

3.01			
------	--	--	--

Go to the previous view on the view stack. This is equivalent to pressing the go back button on the Acrobat tool bar.

**Parameters**

None

**Returns**

Nothing

**Example**

Create a go back button. This code could be part of a batch sequence, for example, to place navigational buttons on the selected PDF documents.

```
var aRect = this.getPageBox();
var width = aRect[2] - aRect[0];
// rectangle is 12 points high and 18 points width, centered at bottom
rect = [width/2-8, 10, width/2+8, 22];
f = this.addField("goBack", "button", 0, rect);
f.textFont="Wingdings";
f.textSize=0;
f.buttonSetCaption("\u00E7"); // left pointing arrow
f.setAction("MouseUp", "app.goBack()"); // add an action
```



## goForward

3.01			
------	--	--	--

Go to the next view on the view stack. This is equivalent to pressing the go forward button on the Acrobat tool bar.

### Parameters

None

### Returns

Nothing

### Example

See the example following `app.goBack`.

## hideMenuItem

4.0		Ⓢ	
-----	--	---	--

Removes a specified menu item.

See also [addItem](#), [addSubMenu](#), [execMenuItem](#), and [listMenuItems](#).

**NOTE:** (SecurityⓈ): This method can only be executed during application initialization or console events. See the [Event Object](#) for a discussion of Acrobat JavaScript events.

### Parameters

---

<b>cName</b>	The menu item name to remove. Menu item names can be discovered with <a href="#">listMenuItems</a> .
--------------	---

---

### Returns

Nothing

## hideToolbarButton

4.0		Ⓢ	
-----	--	---	--

Removes a specified toolbar button.

**NOTE:** (SecurityⓈ): This method can only be executed during application initialization or console events. See the [Event Object](#) for a discussion of Acrobat JavaScript events.

**Parameters**


---

<b>cName</b>	The name of the toolbar button to remove. Toolbar item names can be discovered with <a href="#">listToolBarButtons</a> .
--------------	---

---

**Returns**

Nothing

**Example**

A file named, `myConfig.js`, containing the following script is placed in one of the Folder Level JavaScripts folders.

```
app.hideToolBarButton("Hand");
```

When the Acrobat viewer is started, the "Hand" icon does not appear.

**launchURL**

7.0			
-----	--	--	--

The **launchURL** method launches in a browser window the URL passed to it by the **cURL** parameter.

**Parameters**


---

<b>cURL</b>	<b>cURL</b> is a string that specifies the URL to launch.
<b>bNewFrame</b>	(optional) If <b>true</b> , this method launches the URL in a new window of the browser application. The default is <b>false</b> .

---

**Returns**

The value **undefined** is returned on success. An exception is thrown on failure.

**Example 1**

```
app.launchURL("http://www.adobe.com/", true);
```

**Example 2**

Add an online help item to the menu system. This code should be placed in a folder level JavaScript file, or executed from the JavaScript Debugger console.

```
app.addItem({
  cName: "myHelp", cUser: "Online myHelp",
  cParent: "Help",
  cExec: "app.launchURL('www.myhelp.com/myhelp.html');",
  nPos: 0
});
```

Related methods are `doc.getURL()` and `app.openDoc()`

## listMenuItems

5.0			
-----	--	--	--

Prior to Acrobat 6.0, this method returned a list of menu item names to the console. This method has changed significantly.

Beginning with version 6.0, returns an array of **treeItem** objects, which describes a menu hierarchy.

See also [addItem](#), [addSubMenu](#), [execMenuItem](#), and [hideMenuItem](#).

### Parameters

None

### Returns

Array of [TreeItem Generic Objects](#).

### TreeItem Generic Object

This generic JS Object represents a menu or toolbar item hierarchy. An array of these objects is returned by `app.listMenuItems` and `app.listToolbarButtons` (starting in Acrobat 6.0). It contains the following properties:

<b>cName</b>	The name of a menu item or toolbar button.
<b>oChildren</b>	(optional) An array of <b>treeItem</b> objects containing the submenus or flyout buttons.

### Example 1

List all menu item names to the console.

```
var menuItems = app.listMenuItems()
for( var i in menuItems)
    console.println(menuItems[i] + "\n")
```

### Example 2

List all menu items to console, fancy format.

```
function FancyMenuList(m, nLevel)
{
    var s = "";
    for (var i = 0; i < nLevel; i++) s += " ";
    console.println(s + "+-" + m.cName);
    if ( m.oChildren != null )
        for ( var i = 0; i < m.oChildren.length; i++ )
            FancyMenuList(m.oChildren[i], nLevel + 1);
}
```

```
var m = app.listMenuItems();
for ( var i=0; i < m.length; i++ ) FancyMenuList(m[i], 0);
```

## listToolBarButtons

5.0			
-----	--	--	--

Prior to Acrobat 6.0, this method returned a list of toolbar button names to the console. This method has changed significantly.

Beginning with version 6.0, returns an array of **treeItem** objects which describes a toolbar hierarchy (with flyout toolbars).

### Parameters

None

### Returns

Array of [TreeItem Generic Objects](#).

### Example

List all toolbar names to the console.

```
var toolbarItems = app.listToolBarButtons()
for( var i in toolbarItems)
    console.println(toolbarItems[i] + "\n")
```

See also the [hideToolBarButton](#).

## mailGetAdrrs

6.0		Ⓢ	ⓧ	
-----	--	---	---	--

Pops up an address book dialog to let one choose e-mail recipients. The dialog will be optionally pre-populated with the semi-colon separated lists of addressees in the **cTo**, **cCc**, and **cBcc** strings. The **bCc** and **bBcc** booleans control whether the dialog should allow the user to choose CC and BCC recipients.

See also [app.mailMsg](#), [doc.mailDoc](#), [doc.mailForm](#), [fdf.mail](#) and [report.mail](#).

**NOTES:** The **mailGetAdrrs** is a windows-only feature.

(Security Ⓢ, version 7.0) This method can only be executed during batch or console events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

### Parameters

---

<b>cTo</b>	(optional) A semicolon separated list of "To" addressees to use.
------------	--

---

<b>cCc</b>	(optional) A semicolon separated list of CC addressees to use.
<b>cBcc</b>	(optional) A semicolon separated list of BCC addressees to use.
<b>cCaption</b>	(optional) A string to appear on the caption bar of the address dialog.
<b>bCc</b>	(optional) A boolean to indicate whether the user can choose CC recipients.
<b>bBcc</b>	(optional) A boolean to indicate whether the user can choose BCC recipients. This boolean should only be used when <b>bCc</b> is <b>true</b> ; otherwise, the method fails (and returns <b>undefined</b> ).

### Returns

On failure (the user cancelled), returns undefined. On success, returns an array of three strings for To, CC, BCC.

### Example

```
var attempts = 2;
while (attempts > 0)
{
    var recipients = app.mailGetAddrs
    ({
        cCaption: "Select Recipients, Please",
        bBcc: false
    })
    if (typeof recipients == "undefined" ) {
        if (--attempts == 1)
            app.alert("You did not choose any recipients,"
                + " try again");
        } else break;
    }
    if (attempts == 0)
        app.alert("Cancelling the mail message");
    else {
        JavaScript statements to send mail
    }
}
```

## mailMsg

4.0				
-----	--	--	---	--

Sends out an e-mail message with or without user interaction.

See also [doc.mailDoc](#), [doc.mailForm](#), [fdf.mail](#) and [report.mail](#).

**NOTE:** On Windows: The client machine must have its default mail program configured to be MAPI enabled in order to use this method.

**Parameters**

<b>bUI</b>	Indicates whether user interaction is required. If <b>true</b> , the remaining parameters are used to seed the compose-new-message window that is displayed to the user. If <b>false</b> , the <b>cTo</b> parameter is required and others are optional.
<b>cTo</b>	A semicolon-separated list of addressees.
<b>cCc</b>	(optional) A semicolon-separated list of CC addressees.
<b>cBcc</b>	(optional) A semicolon-separated list of BCC addressees.
<b>cSubject</b>	(optional) Subject line text. The length limit is 64k bytes.
<b>cMsg</b>	(optional) Mail message text. The length limit is 64k bytes.

**Returns**

Nothing

**Example**

This will pop up the compose new message window

```
app.mailMsg(true);
```

This will send out the mail to **fun1@fun.com** and **fun2@fun.com**

```
app.mailMsg(false, "fun1@fun.com; fun2@fun.com", "", "",
"This is the subject", "This is the body of the mail.");
```

It is possible to compose a message containing form data in it.

```
var cMyMsg = "Below are the current budget figures:\n\n";
cMyMsg += "Date Compiled: " + this.getField("date").value + "\n";
cMyMsg += "Current Estimate: " + this.getField("budget").value + "\n";
app.mailMsg({
  bUI: true,
  cTo: "myBoss@greatCo.com",
  cSubject: "The latest budget figures",
  cMsg: cMyMsg
});
```

**newDoc**

5.0		Ⓢ	ⓧ	
-----	--	---	---	--

Creates a new document in the Acrobat Viewer and returns the **doc** object. The optional parameters specify the media box dimensions of the document in points.

**NOTES:** (Security<sup>Ⓢ</sup>): This method can only be executed during batch, console or menu events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

## Parameters

<b>nWidth</b>	(optional) The width (in points) for the new document. The default value is 612.
<b>nHeight</b>	(optional) The height (in points) for the new document. The default value is 792.

## Returns

Returns the [Doc Object](#) of the newly created document

## Example

Add a "New" item to the Acrobat File menu. Within "New", there are three menu items: "Letter", "A4" and "Custom". This script should go in a Folder Level JavaScripts .js file.

```
app.addSubMenu({ cName: "New", cParent: "File", nPos: 0 })
app.addMenuItem({ cName: "Letter", cParent: "New", cExec:
  "app.newDoc();" });
app.addMenuItem({ cName: "A4", cParent: "New", cExec:
  "app.newDoc(420,595)"});
app.addMenuItem({ cName: "Custom...", cParent: "New", cExec:
  "var nWidth = app.response({ cQuestion:'Enter Width in Points',\
    cTitle: 'Custom Page Size'});"
  +"if (nWidth == null) nWidth = 612;"
  +"var nHeight = app.response({ cQuestion:'Enter Height in Points',\
    cTitle: 'Custom Page Size'});"
  +"if (nHeight == null) nHeight = 792;"
  +"app.newDoc({ nWidth: nWidth, nHeight: nHeight })"});
```

The script above will work for versions of Acrobat prior to 7.0, for version 7.0, it will work correctly if the user has checked **Enable menu items JavaScript execution privileges** item under the menu **Edit > Preferences > General > JavaScript**.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged (unless the preferences item **Enable menu items JavaScript execution privileges** is checked, as noted above), so `app.newDoc()` needs to be executed through a [trustedFunction](#). See the technical note [JavaScript Execution through the Menu](#).

The same example can be worked as follows:

```
trustedNewDoc = app.trustedFunction( function (nWidth, nHeight)
{
  app.beginPriv();
  switch( arguments.length ) {
    case 2:
```

```

        app.newDoc( nWidth, nHeight );
        break;
    case 1:
        app.newDoc( nWidth );
        break;
    default:
        app.newDoc();
    }
    app.endPriv();
})
app.addSubMenu({ cName: "New", cParent: "File", nPos: 0 })
app.addItem({ cName: "Letter", cParent: "New", cExec:
    "trustedNewDoc();" });
app.addItem({ cName: "A4", cParent: "New", cExec:
    "trustedNewDoc(420,595)"});
app.addItem({ cName: "Custom...", cParent: "New", cExec:
    "var nWidth = app.response({ cQuestion:'Enter Width in Points',\
    cTitle: 'Custom Page Size'});"
    +"if (nWidth == null) nWidth = 612;"
    +"var nHeight = app.response({ cQuestion:'Enter Height in Points',\
    cTitle: 'Custom Page Size'});"
    +"if (nHeight == null) nHeight = 792;"
    +"trustedNewDoc(nWidth, nHeight) "});

```

The code is a little incomplete. In the case of the "Custom" menu item, additional lines can be inserted to prevent the user from entering the empty string, or a value too small or too large. See the "General Implementation Limits" in the [PDF Reference](#) for the current limitations.

### Example

Create a blank document and acquire the **doc** object, then insert a watermark.

```

var myNewDoc = app.newDoc();
myNewDoc.addWatermarkFromText("Confidential",0,font.Helv,24,color.red);

```

This example uses the **doc.addWatermarkFromText** method.

## newFDF

6.0		Ⓢ	ⓧ	
-----	--	---	---	--

Create a new [FDF Object](#) that contains no data.

**NOTE:** (Security Ⓢ): This method is available only during batch, console, application initialization and menu events. Not available in the Adobe Reader.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

### Parameters

None



**Returns**

A new [FDf Object](#).

**Example**

Create a FDF with an embedded PDF file.

```
var fdf = app.newFDF();
fdf.addEmbeddedFile( "/c/myPDFs/myFile.pdf", 1);
fdf.save( "/c/myPDFs/myFile.fdf" );
```

This example continues following the description of [app.openFDF](#).

**openDoc**

5.0			
-----	--	--	--

Opens a specified PDF document and returns the **doc** object. The returned **doc** object can be used by the script to call methods, or to get or set properties in the newly opened document.

**NOTE:** When a batch sequence is running, a modal dialog is open, which prevents user interference while processing; consequently, this method cannot be executed through a batch sequence.


**NOTE:** An exception is thrown and an invalid [Doc Object](#) is returned when an `html` document is opened using this method. Enclose `app.openDoc` in a `try/catch` construct to catch the exception. See **Example 2** below.

**Parameters**

<b>cPath</b>	A device-independent path to the document to be opened. The path can be relative to <b>oDoc</b> , if passed. The target document must be accessible in the default file system.  <b>NOTE:</b> When <b>cFS</b> is set to "CHTTP", the <b>cPath</b> string should be escaped, perhaps using the core JavaScript global function <code>encodeURIComponent()</code> . See <b>Example 5</b> below.
<b>oDoc</b>	(optional) A <a href="#">Doc Object</a> to use as a base to resolve a relative <b>cPath</b> . Must be accessible in the default file system.
<b>cFS</b>	(optional, version 7.0) A string that specifies the source file system name. Two values are supported: "" (the empty string) representing the default file system and "CHTTP". The default is the default file system. This parameter is only relevant if the web server supports WebDAV.
<b>bHidden</b>	(optional, version 7.0) A boolean, which if <b>true</b> , opens the PDF file with its window hidden. The default is <b>false</b> .

---

**bUseConv** (optional, version 7.0) This parameter is used when **cPath** references a non-PDF file. The **bUseConv** is a boolean, which if **true**, the method will try to convert the non-PDF file to a PDF document. The default for this parameter is **false**.

**NOTE:** (Security , version 7.0) **bUseConv** can only be set to **true** during console and batch events. See also [Privileged versus Non-privileged Context](#).

---

## Returns

The [Doc Object](#), or **null**

**NOTE:** For version 5.0, this method returns a **Doc Object**. In version 5.0.5, the method returns the **Doc Object**, or **null** if the target document does not have the **doc.disclosed** property set to **true**. The “Acrobat 5.0.5 Accessibility and Forms Patch” changed this behavior—this is the behavior of **openDoc** in Acrobat 6.0 or later—as follows: During a batch, console or menu event, **openDoc** ignores the **disclosed** property and returns the **Doc Object** of the file specified by **cPath**; during any other event, **openDoc** returns the **Doc Object**, if **disclosed** is **true**, and **null**, otherwise.

## Example 1

This example opens another document, inserts a prompting message into a text field, sets the focus in the field, then closes the current document.

```
var otherDoc = app.openDoc("/c/temp/myDoc.pdf");
otherDoc.getField("name").value="Enter your name here: ";
otherDoc.getField("name").setFocus();
this.closeDoc();
```

Same example as above, but a relative path is given.

```
var otherDoc = app.openDoc("myDoc.pdf", this);
otherDoc.getField("name").value="Enter your name here: ";
otherDoc.getField("name").setFocus();
this.closeDoc();
```

This example uses **doc.closeDoc** and **field.setFocus**.

## Example 2

Open an html document on hard drive and convert to PDF.

```
try {
    app.openDoc("/c/myWeb/myHomePage.html");
} catch (e) {};
```

## Example 3 (Acrobat 7.0)

Open a hidden PDF document, extract information from it, and close it.

```
oDoc = app.openDoc({
    cPath: "/C/myDocs/myInfo.pdf",
    bHidden: true
```

```
});
var v = oDoc.getField("myTextField").value;
this.getField("yourTextField").value = v;
oDoc.closeDoc();
```

**Example 4 (Acrobat 7.0)**

Open a non-PDF file by converting it to a PDF document. The following script can be executed successfully from the console.

```
app.openDoc({
  cPath: "/c/temp/myPic.jpg",
  bUseConv: true
})
```

**Example 5 (Acrobat 7.0)**

Open a file from a WebDAV server. The `app.openDoc()` method requires the path to the file to be escaped.

```
var myURL = encodeURI("http://www.myCom.com/My Folder/Com Doc.pdf");
app.openDoc({cPath: myURL, cFS: "HTTP"});
```

See also `app.browseForDoc()`.

**openFDF**

6.0		Ⓢ	ⓧ	
-----	--	---	---	--

This method creates a new [FDF Object](#) by opening the specified file. The **FDF** object has methods and properties that can be used on the data that this file contains.

**NOTES:** (Security Ⓢ): This method is available only during batch, console, application initialization and menu events. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

Not available in the Adobe Reader.

**Parameters**

---

<b>cDIPath</b>	The device-independent path to the file to be opened.
----------------	---

---

**Returns**

The [FDF Object](#) for the FDF file that is opened.

**Example**

Create a FDF with an embedded PDF file.

```
var fdf = app.newFDF();
fdf.addEmbeddedFile( "/c/myPDFs/myFile.pdf", 1);
```

```
fdf.save( "/c/myFDFs/myFile.fdf" ); // save and close this FDF

// now open the fdf and embed another PDF doc.
var fdf = app.openFDF( "/c/myFDFs/myFile.fdf" );
fdf.addEmbeddedFile( "/c/myPDFs/myOtherFile.pdf", 1);
fdf.save( "/c/myFDFs/myFile.fdf" ); // save and close this FDF
```

See [fdf.signatureSign](#) for another example of usage.

## popUpMenu

5.0			
-----	--	--	--

Creates a pop-up menu at the current mouse position, containing the specified items.

See also the preferred method [popUpMenuEx](#).

### Parameters

<b>cItem</b>	(optional) If the argument is a string, then it is listed in the menu as a menu item. The menu item name "-" is reserved to draw a separator line in the menu.
<b>Array</b>	(optional) If the argument is an array then it appears as a submenu where the first element in the array is the parent menu item. This array can contain further submenus if desired.

### Returns

Returns the name of the menu item that was selected, or **null** if no item was selected.

### Example

```
var cChoice = app.popUpMenu("Introduction", "-", "Chapter 1",
    [ "Chapter 2", "Chapter 2 Start", "Chapter 2 Middle",
      ["Chapter 2 End", "The End"]]);
app.alert("You chose the \" + cChoice + "\" menu item");
```

## popUpMenuEx

6.0			
-----	--	--	--

Creates a pop-up menu at the current mouse position, containing the specified items.

Each of the one or more parameters is a [MenuItem Generic Object](#) that describes a menu item to be included in the pop up menu.

The [popUpMenuEx](#) method is preferred over the use of [popUpMenu](#).

## Parameters

---

One or more [MenuItem Generic Objects](#).

---

## Returns

The **cReturn** value of the menu item that was selected, or its **cName**, if **cReturn** was not specified for that item. The method returns **null** if no selection was made.

## MenuItem Generic Object

This generic JS object represents a menu item passed to **app.popUpMenuEx**. It has the following properties:

<b>cName</b>	The name of the menu item. This is the string to appear on the menu item to be created. The value of "-" is reserved to draw a separator line in the menu.
<b>bMarked</b>	(optional) Whether the item is to be marked with a check. The default is <b>false</b> (not marked).
<b>bEnabled</b>	(optional) Whether the item is to appear enabled or grayed out. The default is <b>true</b> (enabled).
<b>cReturn</b>	(optional) A string to be returned when the menu item is selected. The default is the value of <b>cName</b> .
<b>oSubMenu</b>	(optional) A <a href="#">MenuItem Generic Object</a> representing a submenu item, or an array of submenu items, each represented by a <a href="#">MenuItem Generic Object</a> .

## Example 1

The following example illustrates all the features of the **popUpMenuEx ()** method.

```
var cChoice = app.popUpMenuEx
(
  {cName: "Item 1", bMarked:true, bEnabled:false},
  {cName: "-"},
  {cName: "Item 2", oSubMenu:
    [ {cName: "Item 2, Submenu 1"},
      {
        cName: "Item 2, Submenu 2",
        oSubMenu: {cName:"Item 2, Submenu 2, Subsubmenu 1",
          cReturn: "0"}
      }
    ]
  },
  {cName: "Item 3"},
  {cName: "Item 4", bMarked:true, cReturn: "1"}
)
app.alert("You chose the \" + cChoice + "\" menu item");
```

**Example 2**

The `app.popupMenuEx` actually takes a list of [MenuItem Generic Objects](#), as a consequence of this, its parameters cannot be passed to it as a JavaScript variable. The following example gives a workabout: Create an array of menu items and use the Function object method `apply` from core JavaScript. This methods allows arguments to be passed as an array.

```
// Declare pop-up menu properties as an array.
var aParams = [
    {cName: "Adobe Web Page", cReturn: "www.adobe.com"},
    {cName: "-"},
    {cName: "The Adobe Acrobat family",
     cReturn: "http://www.adobe.com/products/acrobat/main.html"},
    {cName: "Adobe Reader",
     cReturn: "http://www.adobe.com/products/acrobat/readstep2.html"}
];
// apply the function app.popupMenuEx to the app object, with an array
// of parameters aParams
var cChoice = app.popupMenuEx.apply( app, aParams );
if ( cChoice != null ) app.launchURL(cChoice);
```

**removeToolButton**

6.0				
-----	--	--	--	--

Removes a previously added button from the toolbar.

**NOTES:** (Version 7.0) To remove a toolbutton added by the [addToolButton](#) method, `removeToolButton` must be executed within the same context as when `addToolButton` was executed.

If no document was open in Acrobat when the button was added, then there must be no document open in Acrobat when the button is removed. See Example 2 below.

Similarly, if a certain document was the active document when a tool button was added, then that same document must be active for the button to be removed using `removeToolButton`.

In the case of a document that is active when the tool button is added, the button is automatically removed when this document is closed. See also the notes following the description of [addToolButton](#).

**Parameters**


---

<b>cName</b>	The language independent identifier provided when <a href="#">addToolButton</a> was called.
--------------	---

---

**Returns**

Nothing

**Exceptions**

None

**Example 1**See the example following [addToolButton](#).**Example 2**

This example illustrates the removal of a toolbutton with the same context as **addToolButton**. Initially, there is no document open in the Acrobat. Execute the following code from the console

```
app.addToolButton({cName: "button1", cExec:"app.alert('pressed');",
  cTooltext:"Button1"});
```

Now open a PDF document in Acrobat, and execute the next line from the console

```
app.removeToolButton({cName:"button1"});
```

An exception is thrown, the removal of the button fails. Now close the PDF document and executed the **removeToolButton** script again, the button is removed.

**response**

3.01			
------	--	--	--

Displays a dialog box containing a question and an entry field for the user to reply to the question.

**Parameters**

<b>cQuestion</b>	The question to be posed to the user.
<b>cTitle</b>	(optional) The title to appear in the dialog's window title.
<b>cDefault</b>	(optional) A default value for the answer to the question. If not specified, no default value is presented.
<b>bPassword</b>	(optional) If <b>true</b> , indicates that the user's response should show as asterisks (*) or bullets (•) to mask the response, which might be sensitive information. The default is <b>false</b> .
<b>cLabel</b>	(optional, version 6.0) A short string to appear in front of and on the same line as the edit text field.

**Returns**

A string containing the user's response. If the user presses the **cancel** button on the dialog, the response is the **null** object.

**Example**

This example asks for a response from the user, and reports back the response.

```
var cResponse = app.response({
    cQuestion: "How are you today?",
    cTitle: "Your Health Status",
    cDefault: "Fine",
    cLabel: "Response:"
});
if (cResponse == null)
    app.alert("Thanks for trying anyway.");
else
    app.alert("You responded, \""+cResponse+"\", to the health "
        + "question.",3);
```

**setInterval**

5.0			
-----	--	--	--

Registers a JavaScript expression to be evaluated, and executes the expression each time a specified period elapses. Pass the returned **interval** object to [clearInterval](#) to terminate the periodic evaluation. The return value must be held in a JavaScript variable, otherwise the **interval** object will be garbage collected and the clock will stop.

See also [clearInterval](#), [setTimeout](#) and [clearTimeout](#).

**NOTE:** Opening and closing the document JavaScripts dialog causes the JavaScript interpreter to re-read the document JavaScripts, and consequently, to re-initialize any document level variables. Resetting document level variables in this way after JavaScript expressions have been registered to be evaluated by [setInterval](#) or [setTimeout](#) may cause JavaScript errors if those scripts use document level variables.

**Parameters**

<b>cExpr</b>	The JavaScript expression to evaluate.
<b>nMilliseconds</b>	The evaluation time period in milliseconds.

**Returns**

An **interval** object

**Example**

For example, to create a simple color animation on a field called "Color" that changes every second:

```
function DoIt() {
    var f = this.getField("Color");
    var nColor = (timeout.count++ % 10 / 10);
```



```

    // Various shades of red.
    var aColor = new Array("RGB", nColor, 0, 0);
    f.fillColor = aColor;
  }
  // save return value as a variable
  timeout = app.setInterval("DoIt()", 1000);
  // Add a property to our timeout object so that DoIt() can keep
  // a count going.
  timeout.count = 0;

```

See [setTimeout](#) for an additional example.

## setTimeout

5.0			
-----	--	--	--

Registers a JavaScript expression to be evaluated, and executes the expression after a specified period elapses. The expression is executed only once. Pass the returned **timeout** object to [clearTimeout](#) to cancel the timeout event. The return value must be held in a JavaScript variable, otherwise the **timeout** object will be garbage collected and the clock will stop.

See also [clearTimeout](#), [setInterval](#) and [clearInterval](#).

**NOTE:** Opening and closing the document JavaScripts dialog causes the JavaScript interpreter to re-read the document JavaScripts, and consequently, to re-initialize any document level variables. Resetting document level variables in this way after JavaScript expressions have been registered to be evaluated by [setInterval](#) or [setTimeout](#) may cause JavaScript errors if those scripts use document level variables.

### Parameters

<b>cExpr</b>	The JavaScript expression to evaluate.
<b>nMilliseconds</b>	The evaluation time period in milliseconds.

### Returns

A **timeout** object

### Example

This example creates a simple running marquee. Assume there is a text field named "marquee". The default value of this field is "Adobe Acrobat version 7.0 will soon be here!".

```

// Document level JavaScript function
function runMarquee() {
  var f = this.getField("marquee");
  var cStr = f.value;
  // get field value

```

```

        var aStr = cStr.split("");           // convert to an array
        aStr.push(aStr.shift());           // move first char to last
        cStr = aStr.join("");             // back to string again
        f.value = cStr;                   // put new value in field
    }

    // Insert a mouse up action into a "Go" button
    run = app.setInterval("runMarquee()", 100);
    // stop after a minute
    stoprun=app.setTimeout("app.clearInterval(run)", 6000);

    // Insert a mouse up action into a "Stop" button
    try {
        app.clearInterval(run);
        app.clearTimeout(stoprun);
    } catch (e){}

```

Here, we protect the "Stop" button code with a **try/catch**. If the user presses the "Stop" button without having first pressed the "Go", **run** and **stoprun** will be undefined, and the "Stop" code will throw an exception. When the exception is thrown, the **catch** code is executed. In the above example, code does nothing if the user presses "Stop" first.

## trustedFunction

7.0		Ⓢ		
-----	--	---	--	--

This method marks a function as "trusted". Trusted functions are functions that are capable of explicitly increasing the current privilege level for *their* stack frame. Typically, the stack frame (the body of the function) contains security restricted methods that require a privileged context in which to run. By increasing the privilege level, these restricted methods can then be executed in non-privileged contexts.

For background information, read the paragraphs [Privileged versus Non-privileged Context](#) and [JavaScript Execution through the Menu](#) on page 680.

Within the body of the function definition, an **app.beginPriv/app.endPriv** pair needs to enclose any code that normally executes in a privilege context, as the examples below illustrate.

Related methods are [app.trustPropagatorFunction](#), [app.beginPriv](#), and [app.endPriv](#).

(Security Ⓢ): This method is available only during batch, console, and application initialization.

## Parameters

---

<b>oFunc</b>	A function object that specifies the function to mark as trusted.
--------------	---

---

## Returns

On success, returns the same function object that was passed in. After successful execution, the function object will be trusted. On error, throws **NotAllowedError**.

## Method Syntax

This method can be called in two ways.

```
myTrustedFunction = app.trustedFunction(
    function()
    {
        <function body>
    }
);
```

or

```
function myOtherTrustedFunction()
{
    <function body>
};
app.trustedFunction(myOtherTrustedFunction);
```

In addition to the examples that follow, be sure to review the examples following the **app.trustPropagatorFunction()** method. Each of the examples need to be read carefully, they contain many comments that clarify the notion of trusted function and highlight some of the nuances of the topic.

## Example 1

The **app.newDoc()** is a typical example of a security restricted method that needs a privileged context in which to run. Place the following script in a `.js` in the User (or App) JavaScript folder.

```
trustedNewDoc = app.trustedFunction( function (nWidth, nHeight)
{
    // additional code may appear above
    app.beginPriv();           // explicitly raise privilege
    app.newDoc( nWidth, nHeight );
    app.endPriv();
    // additional code may appear below.
})
```

Now, after restarting Acrobat, execute the function **trustedNewDoc()** from anywhere, for example, a mouse up action from a push button. For example, the following is a script for a mouse up action of a button,

```
trustedNewDoc( 200, 200 );
```

Clicking on the button will create a new document 200 points by 200 points. Security restrictions do not allow the execution of `app.newDoc (200, 200)` from a mouse up event, but through this trusted function, the creation of a new document is permitted.

This is not a very sophisticated example, this function requires two positive integers as arguments, the `app.newDoc ()` method has two optional arguments. The above example can be modified so that it too has two optional arguments.

The `trustedNewDoc ()` function can also be executed as a menu item.

```
app.addItem( {
  cName: "myTrustedNewDoc",
  cUser: "New Doc", cParent: "Tools",
  cExec: "trustedNewDoc(200,200)", nPos: 0
} );
```

Again, the `trustedNewDoc ()` can be enhanced by having the user input the desired dimensions for the new page, either through a series of `app.response ()` dialogs, or a full dialog, created by `app.execDialog ()`.

**NOTE:** If `app.newDoc ()` is *not* enclosed in the `app.beginPriv/app.endPriv` pair, executing `trustedNewDoc ()` from a non-privileged context will fail, an exception will be thrown. You need to *explicitly* raise the privilege level in the way illustrated.

## Example 2

The `app.trustedFunction ()` method can be used to execute any code that needs a privileged context in which to execute. Consider, `activeDocs`, a property (not a method) of the `App Object`. This property behaves differently in a variety of settings: if executed from a non-privileged context, it returns an array of active documents that have their `disclosed` property set to `true`; if executed during a console or batch event, it returns an array of *all* active documents. To overcome this limitation, we can define a trusted version of `activeDocs` in a .js file in the User (or App) JavaScript folder:

```
trustedActiveDocs = app.trustedFunction (
  function()
  {
    app.beginPriv(); // explicitly raise privilege
    var d = app.activeDocs;
    app.endPriv();
    return d;
  }
)
```

Now, from a mouse up action of a form button, for example, execute the following code:

```
var d = trustedActiveDocs();
console.println("There are d = " + d.length
  + " files open in the viewer.")
for ( var i=0; i< d.length; i++)
  console.println((i+1) + ". " + d[i].documentFileName )
```

The console will report back the number and filename of *all* documents—disclosed or not—open in the viewer.

**Example 3**

According to the description of a trusted function, a trusted function is one that is capable of explicitly increasing the current privilege level for *its* stack frame.” This example illustrates what is meant by “*its* stack frame”.

In an attempt to make a trusted function more modular, the following code is used:

```
function mySaveAs(doc, path)
{
    doc.saveAs(doc, path);
}
myFunc = app.trustedFunction( function (doc, path)
{
    // privileged and/or non-privileged code here
    app.beginPriv();
    mySaveAs(doc, path);
    app.endPriv();
    // privileged and/or non-privileged code here
}
```

When **myFunc ()** is executed in a non-privileged context, it will throw an exception. This is because when the privileged code, **doc.saveAs (doc, path)** is executed it is not within the stack frame (the function body) of the calling trusted function, it is within the stack frame of **mySaveAs ()**, not that of **myFunc ()**.

You can make **mySaveAs ()** into a trusted function, in this case, **myFunc ()** will succeed, but in the process, you’ve exposed the privileged **doc.saveAs ()** function to non-privileged execution by anyone that knows this function are on your system.

You cannot simply enclose **doc.saveAs (doc, path)** in a **beginPriv/endPriv** pair, for when **myFunc ()** is run from a non-privileged context, an exception will be thrown by the **app.beginPriv ()** within the body of the **mySaveAs ()** function. This is because **mySaveAs ()** is not trusted, and therefore is not authorized to request an increased privilege level. Recall that a *trusted* function “is capable of explicitly increasing the current privilege level”.

To summarize the observations above, there is a need, for a kind of function that, (1) can be called by a trusted function, but, (2) is itself *not trusted*, and so cannot be directly called from a non-privileged context. It is the *trust propagator function* that satisfies these criteria, see **app.trustPropagatorFunction**.

**trustPropagatorFunction**

7.0		Ⓢ		
-----	--	---	--	--


This method marks a function as a “trust propagator”; such a function can inherit trust if called from a trusted function, but is not trusted.

A trust propagator function propagates *trust*, not *privilege*, so, as it is with the method **app.trustedFunction**, an **app.beginPriv/app.endPriv** pair needs to enclose any code, within the function body, that normally executes in a privilege context.

Functions defined in App JavaScript folder .JS files are implicitly **trustPropagator** functions; however, functions defined in the User JavaScript folder .JS files are not.

Trust propagator functions can play the role of utility functions; they can be called by a trusted function and by another trust propagator function, but they cannot successfully be called by a function that is not trusted in a non-privileged context.

See also **app.beginPriv**, **app.endPriv** and **app.trustedFunction**.

**NOTE:** (Security 

### Method Syntax

This method can be called in two ways.

```
myPropagatorFunction = app.trustPropagatorFunction (
    function ()
    {
        <function body>
    }
);
```

or

```
function myOtherPropagatorFunction ()
{
    <function body>
};
app.trustPropagatorFunction (myOtherPropagatorFunction);
```

### Parameters

<b>oFunc</b>	A function object that specifies the function to mark as a trust propagator.
--------------	--

### Returns

On success, returns the same function object that was passed in. After successful execution, the function object will be a trust propagator. On error, throws **NotAllowedError**.

### Example 1

As a preparation for this example, review [Example 3](#) on page 133, following the **app.trustedFunction()** method.

Define a trust propagator function, **mySaveAs**, to save a file to a folder, and define a trusted function, **myTrustedSpecialTaskFunc**, to perform various tasks involving privileged and non-privileged code. The function **mySaveAs()** cannot be called directly in a non-privileged context.

```
mySaveAs = app.trustPropagatorFunction (function (doc, path)
{
    app.beginPriv ();
    doc.saveAs (path);
});
```

```

    app.endPriv();
  })
  myTrustedSpecialTaskFunc = app.trustedFunction(function(doc,path)
  {
    // privileged and/or non-privileged code above
    app.beginPriv();
    mySaveAs(doc,path);
    app.endPriv();
    // privileged and/or non-privileged code below
  });

```

Now, executing the code

```
myTrustedSpecialTaskFunc(this, "/c/temp/mySavedDoc.pdf");
```

from a mouse-up button, for example, saves the current document to the specified path.

## Example 2

In this example, we develop a simple dialog, using the `app.execDialog()` method, and execute privileged code.

The dialog asks for your name, and asks you to browse for a document from your local hard drive (or a network drive). When "OK" button is clicked, the selected file will be loaded into the viewer, and the name entered into the dialog will be placed in the author field of the document properties. (The insertion of the name only occurs if the author field is empty.) The dialog also displays the value of `identity.email`, which is privileged information.

Any code that is privileged is enclosed by a `beginPriv/endPriv` pair.

Note the use of the function `ANTrustPropagateAll()`, defined in the `Annots.js` file. This function takes a single object as its argument, it turns every function in the object into a trust propagator function, then returns that object. It is useful for creating dialogs that use privileged code.

```

myDialog = app.trustedFunction(function()
{
  app.beginPriv();
  var dialog = ANTrustPropagateAll({
    initialize:function(dialog) {
      this.data = {}; // an object to hold dialog data
      app.beginPriv();
      dialog.load({ "emai": "Email: " + identity.email });
      app.endPriv();
    },
    commit:function (dialog) { // called when OK pressed
      var results = dialog.store();
      console.println("Your name is " + results["name"] );
      this.data.name = results["name"];
    },
    brws: function (dialog) {
      app.beginPriv();
      var oRetn = app.browseForDoc();
      if ( typeof oRetn != "undefined")

```

```
        this.data.oRetn = oRetn;
    app.endPriv();
},
doDialog:function() {
    app.beginPriv();
    var retn = app.execDialog(this);
    app.endPriv();
    return retn;
},
description: {
    name: "Open File & Populate Info Field",
    align_children: "align_left",
    elements:
    [
        {
            type: "view",
            align_children: "align_left",
            elements:
            [
                {
                    type: "view",
                    align_children: "align_row",
                    elements:
                    [
                        {
                            type: "static_text",
                            name: "Name: "
                        },
                        {
                            item_id: "name",
                            type: "edit_text",
                            alignment: "align_fill",
                            width: 300,
                            height: 20
                        }
                    ],
                }
            ],
        },
        {
            type: "static_text",
            item_id: "emai",
            name: "Email: ",
            char_width: 25
        },
        {
            type: "gap",
            height: 10
        },
        {
            type: "view",
            align_children: "align_row",
            elements:
```



```

        [
            {
                type: "button",
                name: "Browse",
                item_id: "brws"
            },
            {
                type: "ok_cancel",
                ok_name: "Ok",
                cancel_name: "Cancel"
            }
        ]
    }
]
}
});
app.endPriv();
try { // protect against user pressing the "Esc" key
    // After everything is set up, run the dialog using the doDialog
    // function, defined in the object dialog.
    var retn = dialog.doDialog();
    app.beginPriv();
    // if use clicked the ok button and there is oRetn data we load
    // the requested file using app.openDoc(), a restricted method.
    if ( (retn == "ok") && dialog.data.oRetn ) {
        var oDoc = app.openDoc({
            cPath: dialog.data.oRetn.cPath,
            cFS: dialog.data.oRetn.cFS
        });
        if ( !oDoc.info.Author )
            oDoc.info.Author = dialog.data.name;
    }
    app.endPriv();
} catch(e) {}
})

```

This dialog can be activated from button, or, more appropriately, from a menu item or a toolbar button. For example, place the following code in a User JavaScript file. This will add an menu item to the **Tools** menu.

```

app.addItem( { cName: "myDialog", cUser: "My Cool Dialog",
    cParent: "Tools", cExec: "myDialog()", nPos: 0 } );

```

## App.media Object

The global **app.media** object defines properties and functions useful in multimedia JavaScript code.

Several of the properties of **app.media** are enumeration objects that list the values allowed in various properties. Note that future versions of Acrobat may add more such values, and JavaScript code should be prepared to encounter values other than the ones listed here. Similarly, JavaScript code may be run on an older version of Acrobat than it was designed for, in which case it will have to fall back to using the values available in that version.

### App.media Object Properties

#### align

6.0				
-----	--	--	--	--

The **app.media.align** property enumerates the values that may be found in the **MediaSettings.floating.align** property. The alignment is positioned relative to the window specified by the **MediaSettings.floating.over** property, see the values for **app.media.over**.

These values are

Value	Description
<code>app.media.align.topLeft</code>	position floating window at the top left corner
<code>app.media.align.topCenter</code>	position floating window at the top center
<code>app.media.align.topRight</code>	position floating window at the top right corner
<code>app.media.align.centerLeft</code>	position floating window at the center left
<code>app.media.align.center</code>	position floating window at the center
<code>app.media.align.centerRight</code>	position floating window at the center right
<code>app.media.align.bottomLeft</code>	position floating window at the bottom left corner
<code>app.media.align.bottomCenter</code>	position floating window at the bottom center
<code>app.media.align.bottomRight</code>	position floating window at the bottom right corner

Type: Object (enumeration)

Access: R.

## canResize

6.0				
-----	--	--	--	--

**app.media.canResize** property enumerates the values that may be found in the **MediaSettings.floating.canResize** property, which specifies whether a floating window may be resized by the user.

These values are

Value	Description
app.media.canResize.no	may not be resized
app.media.canResize.keepRatio	may be resized only if aspect ration is preserved
app.media.canResize.yes	may be resized without preserving aspect ratio

Type: Object (enumeration)

Access: R.

## closeReason

6.0				
-----	--	--	--	--

**app.media.closeReason** enumerates the values that may be found in the event.reason property for a Close event. These values are:

```

app.media.closeReason.general
app.media.closeReason.error
app.media.closeReason.done
app.media.closeReason.stop
app.media.closeReason.play
app.media.closeReason.uiGeneral
app.media.closeReason.uiScreen
app.media.closeReason.uiEdit
app.media.closeReason.docClose
app.media.closeReason.docSave
app.media.closeReason.docChange
    
```

See the [afterClose](#) and [onClose](#) events.

Type: Object (enumeration)

Access: R.

## defaultVisible

6.0				
-----	--	--	--	--

The `app.media.defaultVisible` property is defined as `true`, which is the default value for `MediaSettings.visible`.

Type: Boolean

Access: R.

## ifOffScreen

6.0				
-----	--	--	--	--

The `app.media.ifOffScreen` property enumerates the values allowed in a `MediaSettings.floating.ifOffScreen` property, which specifies what action should be taken if the floating window is positioned totally or partially offscreen.

These values and their descriptions are given in the table below:

Value	Description
<code>app.media.ifOffScreen.allow</code>	take no action
<code>app.media.ifOffScreen.forceOnScreen</code>	move and/or resize the window so that it is on-screen
<code>app.media.ifOffScreen.cancel</code>	cancel playing the media clip

Type: Object (enumeration)

Access: R.

## layout

6.0				
-----	--	--	--	--

The `app.media.layout` property enumerates the values allowed in a `MediaSettings.layout` property.

The table below contains the values and their descriptions:

Value	Description
<code>app.media.layout.meet</code>	scale to fit all content, preserve aspect, no clipping, background fill
<code>app.media.layout.slice</code>	scale to fill window, preserve aspect, clip X or Y as needed
<code>app.media.layout.fill</code>	scale X and Y separately to fill window

Value	Description
<code>app.media.layout.scroll</code>	natural size with scrolling
<code>app.media.layout.hidden</code>	natural size with clipping
<code>app.media.layout.standard</code>	use player's default settings

Type: Object (enumeration)

Access: R.

## monitorType

6.0				
-----	--	--	--	--

The **app.media.monitorType** property enumerates the values allowed in a **MediaSettings.monitorType** property.

The table below contains the values and their descriptions:

Value	Description
<code>app.media.monitorType.document</code>	The monitor containing the largest section of the document window
<code>app.media.monitorType.nonDocument</code>	The monitor containing the smallest section of the document window
<code>app.media.monitorType.primary</code>	Primary monitor
<code>app.media.monitorType.bestColor</code>	Monitor with the greatest color depth
<code>app.media.monitorType.largest</code>	Monitor with the greatest area (in pixels squared)
<code>app.media.monitorType.tallest</code>	Monitor with the greatest height (in pixels)
<code>app.media.monitorType.widest</code>	Monitor with the greatest width (in pixels)

Type: Object (enumeration)

Access: R.

## openCode

6.0				
-----	--	--	--	--

The **app.media.openCode** enumerates the values that may be found in the code property of the return value from **MediaPlayer.open()**. The values are:

```
app.media.openCode.success
app.media.openCode.failGeneral
app.media.openCode.failSecurityWindow
```

```
app.media.openCode.failPlayerMixed
app.media.openCode.failPlayerSecurityPrompt
app.media.openCode.failPlayerNotFound
app.media.openCode.failPlayerMimeType
app.media.openCode.failPlayerSecurity
app.media.openCode.failPlayerData
```

Type: Object (enumeration)

Access: R.

## over

6.0				
-----	--	--	--	--

The `app.media.over` property enumerates the values allowed in a `MediaSettings.floating.over` property, the value of which is used to align a floating window. See `app.media.align`.

Value	Description
<code>app.media.over.pageWindow</code>	align floating window relative to the document (page) window
<code>app.media.over.appWindow</code>	align floating window relative to the application window
<code>app.media.over.desktop</code>	align floating window relative to the full virtual desktop
<code>app.media.over.monitor</code>	align floating window relative to the (selected) monitor display screen

Type: Object (enumeration)

Access: R.

## pageEventNames

6.0				
-----	--	--	--	--

The `app.media.pageEventNames` property enumerates the values that may be found in the `event.name` property for a page-level action. Event names that represent direct user actions are not included here. This enumeration is used to distinguish page-level actions from user actions. The values are:

```
app.media.pageEventNames.Open
app.media.pageEventNames.Close
app.media.pageEventNames.InView
app.media.pageEventNames.OutView
```

Type: Object (enumeration)

Access: R.

## Example

The `app.media.pageEventNames` can be used to distinguish between a page-level action and a direct user action. The script below is folder-level or document level JavaScript that can be called from anywhere in a document.

```
function myMMfunction () {
    if ( app.media.pageEventNames[event.name] ) {
        console.println("Page Event: " + event.name);
        ...
    } else {
        console.println("User Generated Event: " + event.name);
        ...
    }
}
```

## raiseCode

6.0				
-----	--	--	--	--

The `app.media.raiseCode` property enumerates values that may be found in the `error.raiseCode` property when an exception is thrown. This property exists only when `error.name` is "RaiseError". Other values may be encountered in addition to these.

```
app.media.raiseCode.fileNotFound
app.media.raiseCode.fileOpenFailed
```

*Type: Object (enumeration)*

*Access: R.*

## Example

See the definition of `app.media.getRenditionSettings()` in the `media.js` file for examples of usage.

## raiseSystem

6.0				
-----	--	--	--	--

The `app.media.raiseSystem` property enumerates values that may be found in the `error.raiseSystem` property when an exception is thrown. This property exists only when `error.name` is "RaiseError".

```
app.media.raiseSystem.fileError
```

Other values may be added to the above property.

*Type: Object (enumeration)*

*Access: R.*

**Example**

See the definition of `app.media.getRenditionSettings()` in the `media.js` file for examples of usage.

**renditionType**

6.0				
-----	--	--	--	--

The `app.media.renditionType` property enumerates the values that may be found in `Rendition.type`. The values and their descriptions are given below.

Value	Description
<code>app.media.renditionType.unknown</code>	a type not known by this version of Acrobat
<code>app.media.renditionType.media</code>	a media rendition
<code>app.media.renditionType.selector</code>	a rendition selector

Type: Object (enumeration)

Access: R.

**status**

6.0				
-----	--	--	--	--

The `app.media.status` property enumerates the values that may be found in the `event.media.code` property for a Status event (see `onStatus/afterStatus`). Most of these values have additional information that is found in the `event.text` property. The values are:

Value	Description
<code>app.media.status.clear</code>	empty string - this status event, clears any message
<code>app.media.status.message</code>	general message
<code>app.media.status.contacting</code>	hostname being contacted
<code>app.media.status.buffering</code>	progress message or nothing
<code>app.media.status.init</code>	name of the engine being initialize
<code>app.media.status.seeking</code>	empty string

Along with the `event.media.status` code, there is also the `event.media.text`, a string that reflects the current status, as described above.



Type: Object (enumeration)

Access: R.

See [afterStatus](#) and [onStatus](#).

## trace

6.0				
-----	--	--	--	--

Set **app.media.trace** to **true** to print trace messages to the JavaScript console during player creation and event dispatching.

**NOTE:** **app.media.trace** is for test purposes only. Do not use this property in a PDF file that you publish. It will change in future versions of Acrobat.

Type: Boolean

Access: R/W.

## version

6.0				
-----	--	--	--	--

**app.media.version** is the version number of the multimedia API defined in `media.js`, currently 6.0.

Type: Number

Access: R.

## windowType

6.0				
-----	--	--	--	--

The **app.media.windowType** property enumerates the values allowed in a `MediaSettings.windowType` property. These values are given in the table below.

Value	Description
<code>app.media.windowType.docked</code>	docked to PDF page
<code>app.media.windowType.floating</code>	floating (popup) window
<code>app.media.windowType.fullScreen</code>	full screen mode

Type: Object (enumeration)

Access: R.

## App.media Object Methods

### addStockEvents

6.0				
-----	--	--	--	--

The `app.media.addStockEvents()` method adds stock event listeners to a `MediaPlayer` (see [MediaPlayer Object](#)) and sets `player.stockEvents` as a reference to these listeners for later removal.

If the optional `annot` is provided, then a reference to the `annot` is saved in `MediaPlayer.annot`. Later, when the player is opened with `MediaPlayer.open()`, stock event listeners will also be added to this `annot`, and `annot.player` will be set as a reference to the player.

#### Parameters

<code>player</code>	A required <a href="#">MediaPlayer Object</a>
<code>annot</code>	(optional) A <a href="#">ScreenAnnot Object</a>

#### Returns

Nothing

The stock event listeners provide standard Acrobat behavior such as focus handling.

If `app.media.trace` is `true`, then debug trace listeners are also included with the stock event listeners.

Use the [removeStockEvents\(\)](#) method to remove event listeners that were added via `addStockEvents()`.

The `app.media.createPlayer()` and `app.media.openPlayer()` methods call `addStockEvents()` internally, so in most cases it is not necessary to call this method yourself.

### alertFileNotFound

6.0				
-----	--	--	--	--

The `app.media.alertFileNotFound()` method displays the standard file not found alert, with an optional don't show again checkbox.

#### Parameters

<code>oDoc</code>	<code>oDoc</code> is the document object the alert is associated with
<code>cFilename</code>	<code>cFilename</code> is the name of the missing file

---

<b>bCanSkipAlert</b>	(optional) If <b>bCanSkipAlert</b> is <b>true</b> and the user checks the checkbox, returns <b>true</b> , otherwise returns <b>false</b> . The default is <b>false</b> .
----------------------	--

---

**Returns**

If **bCanSkipAlert** is **true**, returns **true** if checkbox is checked, otherwise returns **false**.

**Example:**

```
if ( !doNotNotify )
{
    var bRetn = app.media.alertFileNotFound(this, cFileClip, true );
    var doNotNotify = bRetn;
}
```

**alertSelectFailed**

6.0				
-----	--	--	--	--

The **app.media.alertSelectFailed()** method displays the standard alert for a **rendition.select()** failure.

**Parameters**

---

<b>oDoc</b>	<b>oDoc</b> is the document object the alert is associated with
<b>oRejects</b>	(optional) If <b>oRejects</b> is provided, it should be an array of <b>MediaReject Objects</b> as returned by <b>PlayerInfoList.select()</b> .
<b>bCanSkipAlert</b>	(optional) If <b>bCanSkipAlert</b> is <b>true</b> and the user checks the checkbox, returns <b>true</b> , otherwise returns <b>false</b> . The default is <b>false</b> .
<b>bFromUser</b>	(optional) <b>bFromUser</b> affects the wording of the alert message. It should be <b>true</b> if a direct user action triggered this code, or <b>false</b> if some other action such as selecting a bookmark triggered it. The default is <b>false</b> .

---

**Returns**

If **bCanSkipAlert** is **true**, returns **true** if checkbox is checked, otherwise returns **false**.

**NOTE:** When **rendition.select()** fails to find a usable player, and the **select()** parameter **bWantRejects** is set to **true**, the returned **MediaSelection Object** will contain an array of **MediaReject Object**, which can be passed to this method as the **oRejects** parameter. The **alertSelectFailed()** method will, in turn, ask the user to go to the web to download an appropriate player.

**Example:**

Displays “Cannot play media clip”, with checkbox.

```
var bRetn = app.media.alertSelectFailed({
    oDoc: this,
    bCanSkipAlert: true
});
```

**argsDWIM**

6.0				
-----	--	--	--	--

The `app.media.argsDWIM` method is a “Do What I Mean” function that is used by `app.media.createPlayer()`, `app.media.openPlayer()`, and `app.media.startPlayer()`. It fills in default values for properties that are not provided in the [PlayerArgs Object](#), picking them out of the [Event Object](#), so that these functions may be used as rendition action event handlers with no arguments or in custom JavaScript with explicit arguments. See `app.media.createPlayer()` for details of the [PlayerArgs Object](#).

**Parameters**

<b>args</b>	The args is a <a href="#">PlayerArgs Object</a> . See <code>createPlayer()</code> for details of the <a href="#">PlayerArgs object</a> .
-------------	--

**Returns**

[PlayerArgs Object](#)

**Example**

See “[Example 1](#)” on page 152 for an example of usage.

**canPlayOrAlert**

6.0				
-----	--	--	--	--

The `app.media.canPlayOrAlert` method determines whether any media playback is allowed and returns `true` if it is. If playback is not allowed, it alerts the user and returns `false`.

**Parameters**

<b>args</b>	The args is a PlayerArgs object. See <a href="#">createPlayer ()</a> for details of the PlayerArgs object.
-------------	--

**Returns**

Returns **true** if media playback is allowed, otherwise, this method returns **false**.

**NOTE:** The [createPlayer \(\)](#) method calls this function before attempting to create a player. If you write your own code to substitute for [createPlayer \(\)](#), you can call [canPlayOrAlert \(\)](#) to alert the user in situations where playback is not allowed, such as in multimedia authoring mode.

The only property in the args object that is used is doc, so you can use:

```
// There is a doc object in myDoc
if( app.media.canPlayOrAlert({ doc: myDoc })
/* OK to create player here */ ;
```

The above code displays “Cannot play media while in authoring mode”, or other alerts, as appropriate.

**computeFloatWinRect**

6.0				
-----	--	--	--	--

The `app.media.computeFloatWinRect ()` method calculates and returns the rectangle in screen coordinates needed as specified by its parameters.

**Parameters**

<b>doc</b>	The document object for the document
<b>floating</b>	The floating parameters from the MediaSettings.floating object
<b>monitorType</b>	A number indicating which monitor to use. See the <a href="#">app.media.monitorType</a> property.
<b>uiSize</b>	(optional) The user interface size given as an array of four numbers [w,x,y,z] representing the size, as returned by <a href="#">MediaPlayer.uiSize</a> .

**Returns**

The rectangle in screen coordinates

**Example:**

```
var floating =
{
    over: app.media.over.monitor,
```

```

        align: app.media.align.center,
        canResize: app.media.canResize.no,
        hasClose: false,
        hasTitle: true,
        width: 400,
        height: 400
    }
    var rect = app.media.computeFloatWinRect
        (this, floating, app.media.monitorType.primary);

```

## constrainRectToScreen

6.0				
-----	--	--	--	--

The `app.media.constrainRectToScreen()` method returns a rectangle of screen coordinates, moved and resized if needed to place it entirely on some display monitor. If `anchorPt` is provided, and `rect` must be shrunk to fit, it shrinks proportionally toward `anchorPt` (which is an array of two numbers representing a point as `[x,y]`).

### Parameters

<code>rect</code>	An array of four number representing screen coordinates of the desired rectangle
<code>anchorPt</code>	(optional) An array of two points <code>[x,y]</code> that is to be an anchor point

### Returns

Returns a rectangle in screen coordinates.

## createPlayer

6.0				
-----	--	--	--	--

The `app.media.createPlayer()` creates a [MediaPlayer Object](#) without actually opening the player, using values provided in the `args` parameter. To open the player, call `MediaPlayer.open()`. You can combine these two steps into one by calling `app.media.openPlayer()` instead of `createPlayer()`.

If `createPlayer()` is called inside a rendition action (e.g. in custom JavaScript entered from the Actions tab in the Multimedia Properties panel), default values are taken from the action's [Event Object](#). The `args` parameter is not required in this case unless you want to override the rendition action's values. The `createPlayer()` calls `argsDWIM()` to process the [Event Object](#) and `args` (see [PlayerArgs Object](#)) parameter.

Unless `noStockEvents` of the [PlayerArgs Object](#) is set to `true`, the [MediaPlayer Object](#) is equipped with stock event listeners which provide the standard behavior required to

interact properly with Acrobat. Additional event listeners can be provided in the PlayerArgs object or may be added afterward with `MediaPlayer.events.add()`.

If `args.annot.player` is an open `MediaPlayer`, `createPlayer()` closes that player, which fires events.

## Parameters

<b>args</b>	(optional) The <b>args</b> parameter is a PlayerArgs object. The parameter <b>args</b> is optional if <code>createPlayer()</code> is executed within a Rendition action with an associated rendition; in this case, the properties of <b>args</b> are populated by the defaults and by options selected in the UI. Otherwise, an <b>args</b> parameter is required, see documentation of the PlayerArgs object below for required properties of the object.
-------------	---

## PlayerArgs Object

Property	Type	Description
<b>doc</b>	Object	The doc object of the document. Required if both <b>annot</b> and <b>rendition</b> are omitted, e.g. for URL playback.
<b>annot</b>	Object	A <a href="#">ScreenAnnot Object</a> . Required for docked playback unless it is found in the <a href="#">Event Object</a> or <code>MediaSettings.page</code> is provided. The new player is associated with the <b>annot</b> . If a player was already associated with the <b>annot</b> , it is stopped and closed.
<b>rendition</b>	Object	(optional) A <a href="#">Rendition Object</a> (either a <code>MediaRendition</code> or a <code>RenditionList</code> ). Required unless <b>rendition</b> found in the <a href="#">Event Object</a> , or <b>URL</b> is present.
<b>URL</b>	String	Either <b>URL</b> or <b>rendition</b> is required, with <b>URL</b> taking precedence.
<b>mimeType</b>	String	(optional) Ignored unless <b>URL</b> is present. If <b>URL</b> is present, either <b>mimeType</b> or <code>settings.players</code> , as returned by <code>app.media.getPlayers()</code> , is required
<b>settings</b>	Object	(optional) A <a href="#">MediaSettings Object</a> . Overrides the <b>rendition</b> settings.

Property	Type	Description
<b>events</b>	Object	(optional) An <a href="#">EventListener Object</a> . Optional if stock events are used, added after stock events.
<b>noStockEvents</b>	Boolean	(optional) If <b>true</b> , do not use stock events. The default is <b>false</b> .
<b>fromUser</b>	Boolean	(optional) It should be <b>true</b> if a direct user action will trigger this code, or <b>false</b> , otherwise. The default depends on <a href="#">Event Object</a> .
<b>showAltText</b>	Boolean	(optional) If <b>true</b> , show alternate text (see <a href="#">altText</a> ) if the media can't be played. The default is <b>true</b> .
<b>showEmptyAltText</b>	Boolean	(optional) If <b>true</b> and alternate text (see <a href="#">altText</a> ) is empty, show the alternate text as an empty box; if <b>false</b> , respond with an alert. The default value is <b>true</b> if <b>fromUser</b> is <b>false</b> , and <b>false</b> if <b>fromUser</b> is <b>true</b> .

## Returns

[MediaPlayer Object](#)

## Example 1

The following code is taken from `media.js`, it is the definition of [openPlayer\(\)](#), which uses [createPlayer\(\)](#) in its definition.

```
app.media.openPlayer = function( args )
{
    var player = null;
    try
    {
        args = app.media.argsDWIM( args );

        player = app.media.createPlayer( args );
        if( player )
        {
            var result = player.open();
            if( result.code != app.media.openCode.success )
            {
                player = null;
                app.media.alert
                    ( "Open", args, { code: result.code } );
            }
            else if( player.visible )
                player.setFocus(); // fires Focus event
        }
    }
}
```



```

catch( e )
{
    player = null;
    app.media.alert( 'Exception', args, { error: e } );
}

return player;
}

```

**Example 2**

See the examples at the end of the description of [openPlayer \(\)](#) for examples of [PlayerArgs](#) usage.

**getAltTextData**

6.0				
-----	--	--	--	--

The `app.media.getAltTextData ()` method returns a **MediaData** object (this is the same as the **MediaSettings.data** object) which represents alternate text data for the given text. This **MediaData** object can be used to create a player to display the alternate text.

**Parameters**

<b>cAltText</b>	A string that is to be used as alternate text data
-----------------	--

**Returns**

MediaData object

See **MediaSettings.data**.

**Example**

See the embedded example following [getAltTextSettings \(\)](#).

**getAltTextSettings**

6.0				
-----	--	--	--	--

The `app.media.getAltTextSettings ()` takes a **PlayerArgs** Object containing at least **settings**, **showAltText**, and **showEmptyAltText** properties, along with a selection object as returned by `rendition.select ()`, and finds the first available alternate text rendition if there is one. It then creates and returns a new **MediaSettings** Object suitable for playback of the alternate text. Otherwise it returns **null**.

## Parameters

<b>args</b>	A <b>PlayerArgs Object</b> , see <a href="#">PlayerArgs Object</a> for more information on this object.
<b>selection</b>	A <a href="#">MediaSelection Object</a>

## Returns

[MediaSettings Object](#) or **null**

## Example

This example plays back the alternate text of the rendition. The code plays back the alternate text in a screen annot, but can be modified for playback in a floating window.

```
var rendition = this.media.getRendition("myClip");
var settings = rendition.getPlaySettings();
var args = {
    settings: settings,
    showAltText: true,
    showEmptyAltText: true
};
var selection = rendition.select();
settings = app.media.getAltTextSettings( args, selection );

// You can also play custom alternate text by uncommenting the next line
// settings.data = app.media.getAltTextData("A. C. Robot");

// Uncomment the code below to obtain a floating window to playback
// the alternate text
/*
settings.windowType = app.media.windowType.floating
settings.floating = {
    canResize: app.media.canResize.keepRatio,
    hasClose: true,
    width: 400,
    height: 100
} */

// now define an args parameter for use with openPlayer, which will
// play the alternate text.
args = {
    rendition: rendition,
    annot: this.media.getAnnot({nPage: 0, cAnnotTitle:"myScreen"}),
    settings: settings
};
app.media.openPlayer( args );
```

## getAnnotStockEvents

6.0				
-----	--	--	--	--

The `app.media.getAnnotStockEvents ()` method returns an [Event Object](#) containing the stock event listeners required in a screen annot for normal playback in Acrobat. The stock event listeners provide standard Acrobat behavior such as focus handling.

If `app.media.trace` is `true`, then debug trace listeners are also included with the stock event listeners.

### Parameters

<code>settings</code>	A number corresponding to the <code>windowType</code> , see <code>app.media.windowType</code> .
-----------------------	---

### Returns

[Event Object](#)

## getAnnotTraceEvents

6.0				
-----	--	--	--	--

The `app.media.getAnnotTraceEvents ()` method returns an [Events Object](#) containing event listeners that provide a debugging trace as events are dispatched.

### Parameters

None

### Returns

[Events Object](#)

## getPlayers

6.0				
-----	--	--	--	--

The `app.media.getPlayers ()` method returns a [PlayerInfoList Object](#), which is an array of [PlayerInfo Objects](#) representing the available media players.

The `PlayerInfoList` may be filtered using its `select ()` method, and it may be used in the `settings.players` property when creating a media player with `createPlayer ()`.

See [PlayerInfoList Object](#) and [PlayerInfo Object](#) for more details.

**Parameters**


---

<b>cMimeType</b>	(optional) An optional MIME type such as "audio/wav". If <b>cMimeType</b> is omitted, the list includes all available players. If <b>cMimeType</b> is specified, the list includes only players that can handle that MIME type.
------------------	---

---

**Returns**

[PlayerInfoList Object](#)

**Example 1**

List MP3 players to the debug console.

```
var mp = app.media.getPlayers("audio/mp3")
for ( var i = 0; i < mp.length; i++) {
  console.println("\nmp[" + i + "] Properties");
  for ( var p in mp[i] ) console.println(p + ": " + mp[i][p]);
}
```

**Example 2**

Choose any player that can play Flash media by matching the MIME type. The code assumes the code below is executed as a Rendition action with associated rendition (so no arguments for **createPlayer ()** are required).

```
var player = app.media.createPlayer();
player.settings.players
  = app.media.getPlayers( "application/x-shockwave-flash" );
player.open();
```

**getPlayerStockEvents**

6.0				
-----	--	--	--	--

The **app.media.getPlayerStockEvents ()** returns a [Events Object](#) containing the stock event listeners required in a media player for normal playback in Acrobat. The stock event listeners provide standard Acrobat behavior such as focus handling.

Use **MediaPlayer.events.add ()** to add these stock events to a media player.

The **app.media.createPlayer ()** and **app.media.openPlayer ()** methods automatically call **getPlayerStockEvents ()** internally, so it is not necessary to call this method yourself unless you're writing code that sets up all event listeners explicitly.

If **app.media.trace** is **true**, then debug trace listeners are also included with the stock event listeners.

**Parameters**

<b>settings</b>	A <a href="#">MediaSettings Object</a>
-----------------	--

**Returns**

[Events Object](#)

**getPlayerTraceEvents**

6.0				
-----	--	--	--	--

The `app.media.getPlayerTraceEvents()` method returns an [Events Object](#) containing event listeners that provide a debugging trace as events are dispatched.

**Parameters**

None

**Returns**

[Events Object](#)

**getRenditionSettings**

6.0				
-----	--	--	--	--

The `app.media.getRenditionSettings()` method calls `Rendition.select()` to get a [MediaSelection Object](#), then `MediaSelection.rendition.getPlaySettings()` to get a [MediaSettings Object](#) for playback. If either of these fails, it calls the `getAltTextSettings()` method to get a [MediaSettings Object](#) for alternate text playback. Finally, it returns the resulting [MediaSettings Object](#), or `null` if `getAltTextSettings()` returned `null` (i.e. alt text was not specified or not allowed).

**Parameters**

<b>args</b>	A <a href="#">PlayerArgs Object</a> , see <a href="#">PlayerArgs Object</a> for more information on this object.
-------------	--

**Returns**

[MediaSettings Object](#) or `null`

**Example**

See [Example 3](#) following the `openPlayer()` method.

## getURLData

6.0				
-----	--	--	--	--

The **app.media.getURLData()** returns a **MediaData object** which represents data to be retrieved for a URL and optional MIME type. This **MediaData object** can be used to create a player which will access data from that URL. See **MediaSettings.data** for more information on the **MediaData object**.

### Parameters

<b>cURL</b>	The URL form which media data is to be retrieved.
<b>cMimeType</b>	(optional) The MIME type of the data.

### Returns

MediaData object

### Example

The following example retrieves a media clip from the Internet and plays it in a floating window.

```
var myURLClip = "http://www.mywebsite.com/myClip.mpg";
var args = {
  URL: myURLClip,
  mimeType: "video/x-mpg",
  doc: this,
  settings: {
    players: app.media.getPlayers("video/x-mpg"),
    windowType: app.media.windowType.floating,
    data: app.media.getURLData(myURLClip, "video/x-mpg"),
    floating: { height: 400, width: 600 }
  }
}
app.media.openPlayer(args);
```

## getURLSettings

6.0				
-----	--	--	--	--

The **app.media.getURLSettings()** method takes a **PlayerArgs Object** which contains a **settings** property and returns a **MediaSettings Object** suitable for playback of a URL. The settings property must contain a **URL** property and may contain a **mimeType** property. It may also contain additional settings which are copied into the resulting settings object.

**Parameters**


---

<b>args</b>	A PlayerArgs Object, see <a href="#">PlayerArgs Object</a> for more information on this object.
-------------	---

---

**Returns**[MediaSettings Object](#)**Example 1**

Same example as above. Basically, **getURLSettings ()** calls **getURLData ()** and inserts the return MediaData object into the **data** property into the **setting**, which it then returns.

```
var myURLClip = "http://www.mywebsite.com/myClip.mpg";
args = {
  URL: myURLClip,
  mimeType: "video/x-mpg",
  doc: this,
  settings:
  {
    players: app.media.getPlayers("video/x-mpg"),
    windowType: app.media.windowType.floating,
    floating: { height: 400, width: 600 }
  }
};
settings = app.media.getURLSettings(args)
args.settings = settings;
app.media.openPlayer(args);
```

**Example 2**

The example below is a custom Keystroke action of a combo box. The combobox is a simple playlist of streamed audio/video web sites. The export value of each element in the list has the form "URL,mimeType", for example

```
http://www.mySite.com/streaming/radio.asx,video/x-ms-asx
```

The script below first splits the export value into an array of length 2, the first element is the URL, the second is the mimeType. Any video will be shown in the screen annot "myScreen"; otherwise, only audio is heard.

```
if (!event.willCommit)
{
  var aURLMime = event.changeEx.split(",");
  console.println("aURLMime [0] = " + aURLMime [0]);
  console.println("aURLMime [1] = " + aURLMime [1]);
  var args = {
    annot: this.media.getAnnot ({ nPage: 0, cAnnotTitle: "myScreen" }),
    URL: aURLMime [0],
    mimeType: aURLMime [1],
    doc: this,
```

```

        settings: {
            players: app.media.getPlayers(aURLMime[1]),
            windowType: app.media.windowType.docked
        }
    };
    settings = app.media.getURLSettings(args);
    args.settings = settings;
    var player = app.media.openPlayer(args);
}

```

## getWindowBorderSize

6.0				
-----	--	--	--	--

The `app.media.getWindowBorderSize()` method returns an array of four numbers representing the size in pixels of the left, top, right, and bottom borders that would be used for a floating window with the properties specified in the parameters.

The `hasTitle` and `hasClose` parameters are booleans, and `canResize` may be any of the values in `app.media.canResize`.

These parameters have the same names as properties of a `MediaSettings.floating` object, so you can simply pass in a floating object as a single parameter:

```
var size = doc.media.getWindowBorderSize( settings.floating );
```

### Parameters

<code>hasTitle</code>	(optional) The default is <code>true</code>
<code>hasClose</code>	(optional) The default is <code>true</code>
<code>canResize</code>	(optional) The default is <code>app.media.canResize.no</code>

### Returns

An array of numbers of length 4

## openPlayer

6.0				
-----	--	--	--	--

The `app.media.openPlayer()` calls `app.media.createPlayer()` to create a [MediaPlayer Object](#), and then calls `MediaPlayer.open()` to open the actual player.

This function fires several events which may include **Ready** (see [onReady](#) and [afterReady](#)), **Play** (see [onPlay](#) and [afterPlay](#)) and **Focus** (see [onFocus](#) and [afterFocus](#)). See also the [EventListener Object](#) object for a general description of these events.

The method alerts the user and returns `null` on failure. Does not throw exceptions.



## Parameters

---

<b>args</b>	(optional) The args is a PlayerArgs object. See <a href="#">PlayerArgs Object</a> .
-------------	---

---

## Returns

A [MediaPlayer Object](#) or **null** on failure

## Example 1

The following is a minimal example. This is a Custom JavaScript from the Actions tab in the Multimedia Properties panel of a Screen Annot. To override the parameters specified by the UI of the Screen Annot, the **args** parameter is passed.

```
app.media.openPlayer();
```

Override **settings.repeat**: if repeat is set to 1, change it to 2; if not 1, set to 1.

```
var nRepeat =
    ( event.action.rendition.getPlaySettings().repeat == 1 ) ? 2 : 1;
var args = { settings: { repeat: nRepeat } };
app.media.openPlayer(args);
```

See the [Event Object](#) for an explanation of **event.action.rendition**. The above example also uses **Rendition.getPlaySettings()** to access the settings associated with the rendition to be played (the one associated with the Screen Annot).

## Example 2

The following script is executed from a mouse up action of a form button. It plays a docked media clip in a ScreenAnnot.

```
app.media.openPlayer({
    rendition: this.media.getRendition( "myClip" ),
    annot: this.media.getAnnot( {nPage:0,cAnnotTitle:"myScreen"} ),
    settings: { windowType: app.media.windowType.docked }
});
```

## Example 3

This is a Custom JavaScript from the Actions tab in the Multimedia Properties of a Screen Annot. The user click on the annot, and a randomly chosen movie clip is played.

```
// these are placed at the top level of the document JavaScripts
var myRenditions = new Array();
myRenditions[0] = "myClip1";
myRenditions[1] = "myClip2";
myRenditions[2] = "myClip3";

// this code is a Custom JavaScript of a ScreenAnnot. All renditions
// are docked and are played in the ScreenAnnot.
var l = myRenditions.length;
randomIndex = Math.floor( Math.random() * l ) % l;
```

```

var rendition = this.media.getRendition(myRenditions[randomIndex]);
var settings = app.media.getRenditionSettings({ rendition: rendition });

var args = { rendition: rendition, settings: settings }
app.media.openPlayer(args);

```

## removeStockEvents

6.0				
-----	--	--	--	--

The **app.media.removeStockEvents()** method removes any stock event listeners from a [MediaPlayer Object](#) and from any associated [ScreenAnnot Object](#), and deletes the **player.stockEvents**, **player.annot**, **annot.stockEvents**, and **annot.player** properties. This undoes the effect of a previous [addStockEvents\(\)](#) call.

### Parameters

<b>player</b>	A <a href="#">MediaPlayer Object</a>
---------------	--------------------------------------

### Returns

Nothing

## startPlayer

6.0				
-----	--	--	--	--

The **app.media.startPlayer()** method checks whether an **annot** is provided in the [PlayerArgs Object](#) and the **annot** already has a player open. If so, it calls **player.play()** on that player to start or resume playback. If not, it calls **app.media.openPlayer()** to create and open a new [MediaPlayer Object](#). See [openPlayer](#) for more details.

**NOTE:** **app.media.startPlayer()** is the default Mouse Up action when you use the Acrobat user interface to create a multimedia **annot** and **rendition** and don't specify any custom JavaScript.

### Parameters

<b>args</b>	(optional) The <b>args</b> is a <a href="#">PlayerArgs Object</a> . See <a href="#">PlayerArgs Object</a> .
-------------	---

### Returns

A [MediaPlayer Object](#) or **null** on failure

**Example**

Start a screen annot from a form button.

```
var args = {
    rendition: this.media.getRendition( "myClip" ),
    annot: this.media.getAnnot({ nPage: 0, cAnnotTitle: "myScreen" }),
};
app.media.startPlayer(args);
```

---

**Bookmark Object**

A [Bookmark Object](#) represents a node in the bookmark tree that appears in the bookmarks navigational panel. Bookmarks are typically used as a “table of contents” allowing the user to navigate quickly to topics of interest.

---

**Bookmark Properties****children**

5.0				
-----	--	--	--	--

Returns an array of [Bookmark Objects](#) that are the children of this bookmark in the bookmark tree. If there are no children of this bookmark, this property has a value of **null**.

See also [parent](#) and [bookmarkRoot](#).

*Type:* Array | null

*Access:* R.

**Example**

Dump all bookmarks in the document.

```
function DumpBookmark(bkm, nLevel)
{
    var s = "";
    for (var i = 0; i < nLevel; i++) s += " ";
    console.println(s + "+-" + bkm.name);
    if (bkm.children != null)
        for (var i = 0; i < bkm.children.length; i++)
            DumpBookmark(bkm.children[i], nLevel + 1);
}
console.clear(); console.show();
console.println("Dumping all bookmarks in the document.");
DumpBookmark(this.bookmarkRoot, 0);
```

## color

5.0	Ⓧ			
-----	---	--	--	--

Specifies the color for a bookmark. Values are defined by using gray, RGB or CMYK color. See [Color Arrays](#) for information on defining color arrays and how values are used with this property. See also [style](#).

**NOTE:** This property is read-only in Adobe Reader.

*Type: Array*

*Access: R/W.*

### Example

The following fun script will color the top level bookmark red, green and blue.

```
var bkm = this.bookmarkRoot.children[0];
bkm.color = color.black;
var C = new Array(1, 0, 0);
var run = app.setInterval(
    'bkm.color = ["RGB",C[0],C[1],C[2]]; C.push(C.shift());', 1000);
var stoprun=app.setTimeout(
    "app.clearInterval(run); bkm.color=color.black",12000);
```

## doc

5.0				
-----	--	--	--	--

The [Doc Object](#) that the bookmark resides in.

*Type: object*

*Access: R.*

## name

5.0	Ⓧ			
-----	---	--	--	--

The text string for the bookmark that the user sees in the navigational panel.

**NOTE:** This property is read-only in Adobe Reader.

*Type: String*

*Access: R/W.*

### Example

The following code puts the top level bookmark in bold.

```
var bkm = this.bookmarkRoot.children[0];
console.println( "Top level bookmark name: " + bkm.name );
```

The example that follows [bookmark.children](#) also uses the [name](#) **property**.

## open

5.0	Ⓓ			
-----	---	--	--	--

Determines whether the bookmark shows its children in the navigation panel (open) or whether the children sub-tree is collapsed (closed).

**NOTE:** This property is read-only in Adobe Reader.

Type: *Boolean*

Access: *R/W*.

## parent

5.0				
-----	--	--	--	--

Returns the parent bookmark of the bookmark or **null** if the bookmark is the root bookmark. See also [children](#) and [bookmarkRoot](#).

Type: *object* | **null**

Access: *R*.

## style

5.0	Ⓓ			
-----	---	--	--	--

Specifies the style for the bookmark's font: 0 indicates normal, 1 is italic, 2 is bold, and 3 is bold-italic. See also [color](#).

**NOTE:** This property is read-only in Adobe Reader.

Type: *Integer*

Access: *R/W*.

### Example

The following code puts the top level bookmark in bold.

```
var bkm = this.bookmarkRoot.children[0];
bkm.style = 2;
```

---

## Bookmark Methods

### createChild

5.0	Ⓓ		ⓧ	
-----	---	--	---	--

Creates a new child bookmark at the specified location.

See also [children](#), [insertChild](#) and [remove](#).

**Parameters**

<b>cName</b>	The name of the bookmark that the user will see in the navigation panel.
<b>cExpr</b>	(optional) An expression to be evaluated whenever the user clicks on the bookmark. Default is no expression. This is equivalent to creating a bookmark with a JavaScript action; see the <a href="#">PDF Reference</a> , "JavaScript Action" for details.
<b>nIndex</b>	(optional) The 0-based index into the children array of the bookmark at which to create the new child. Default is 0.

**Returns**

Nothing

**Example**

Create a bookmark at the top of the bookmark panel that takes you to the next page in the document.

```
this.bookmarkRoot.createChild("Next Page", "this.pageNum++");
```

**execute**

5.0				
-----	--	--	--	--

Executes the action associated with this bookmark. This can have a variety of behaviors. See the [PDF Reference](#), Section 7.5.3, "Actions Types" for a list of common action types. See also [createChild](#).

**Parameters**

None

**Returns**

Nothing

**Example**

This example implements a simple search of the bookmarks, on success, the action associated with the bookmark is executed.

```
// Document level or folder level JavaScript.
function searchBookmarks(bkm, nLevel, bkmName)
{
    if ( bkm.name == bkmName ) return bkm;
    if (bkm.children != null) {
        for (var i = 0; i < bkm.children.length; i++)
```

```

        {
            var bkMark = searchBookmarks(
                bkm.children[i], nLevel + 1, bkmName);
            if ( bkMark != null ) break;
        }
        return bkMark;
    }
    return null;
}
// Redefine this function for a more sophisticated compare.
function bmkCompare( name1, name2 )
{
    return ( name1 == name2 );
}

```

The following code initiates the search. This code could be executed as field level JavaScript, or be executed as a Menu action.

```

var bkmName = app.response({
    cQuestion: "Enter the name of the bookmark to find",
    cTitle: "Bookmark Search and Execute"
});
if ( bkmName != null ) {
    var bkm = searchBookmarks(this.bookmarkRoot, 0, bkmName );
    if ( bkm != null ) bkm.execute();
    else app.alert("Bookmark not found");
}

```

## insertChild

5.0	Ⓧ		Ⓧ	
-----	---	--	---	--

Inserts the specified bookmark as a child of this bookmark. If the bookmark already exists in the bookmark tree it is unlinked before inserting it back into the tree. In addition, the insertion is checked for circularities and disallowed if one exists. This prevents users from inserting a bookmark as a child or grandchild of itself. See also [children](#), [createChild](#), and [remove](#).

### Parameters

<b>oBookmark</b>	A bookmark object to add as the child of this bookmark.
<b>nIndex</b>	(optional) The 0-based index into the children array of the bookmark at which to insert the new child. The default is 0.

### Returns

Nothing

**Example**

Take the first child bookmark and move it to the end of the bookmarks.

```
var bm = bookmarkRoot.children[0];
bookmarkRoot.insertChild(bm, bookmarkRoot.children.length);
```

**remove**

5.0	Ⓓ		✕	
-----	---	--	---	--

Removes the bookmark and all its children from the bookmark tree. See also [children](#), [createChild](#), and [insertChild](#).

**Parameters**

None

**Returns**

Nothing

**Example**

Remove all bookmarks from the document.

```
bookmarkRoot.remove();
```

**setAction**

6.0				
-----	--	--	--	--

Sets a JavaScript action for a bookmark.

See also `doc.addScript`, `doc.setPageAction`, and `field.setAction`.

**NOTE:** This method will overwrite any action already defined for this bookmark.

**Parameters**


---

<b>cScript</b>	Defines the JavaScript expression that is to be executed whenever the user clicks on the bookmark.
----------------	--

---

**Returns**

Nothing

**Example**

Attach an action to the topmost bookmark.

```
var bm = bookmarkRoot.children[0]
bm.setAction("app.beep(0);");
```



## Catalog Object

A static object that accesses the functionality provided by the Acrobat Catalog plug-in. This plug-in must be installed in order to interface with the **catalog** object.

**NOTE:** Catalog plug-in (and the **catalog** object) is available only in the Acrobat Professional.

See also the [Index Object](#) , used to invoke various indexing operations provided by Catalog plug-in, and the [CatalogJob Generic Object](#).

## Catalog Properties

### isIdle

6.0			X		P
-----	--	--	---	--	---

Returns **true** when Catalog is idle and not busy with an indexing job.

Type: Boolean

Access: R.

### jobs

6.0			X		P
-----	--	--	---	--	---

Gets information about the Catalog jobs. Catalog maintains a list of its pending, in progress and completed jobs for each Acrobat session. Returns an array of [CatalogJob Generic Objects](#).

Type: Array

Access: R.

## Catalog Methods

### getIndex

6.0			X		P
-----	--	--	---	--	---

Uses a specified path of a Catalog index to get an **index** object. The returned **index** object can be used to perform various indexing operations such as building or deleting an index.

**Parameters**


---

<b>cDIPath</b>	The device-independent path of a Catalog index.
----------------	---

---

**Returns**

The [Index Object](#).

**remove**

6.0			X		P
-----	--	--	---	--	---

Removes the specified **CatalogJob** object from Catalog's job list. Catalog maintains a list of pending, in progress and completed jobs for each Acrobat session.

**Parameters**


---

<b>oJob</b>	The <a href="#">CatalogJob Generic Object</a> to remove, as returned by the <a href="#">jobs</a> property and various methods of the <a href="#">Index Object</a> .
-------------	---

---

**Returns**

Nothing

**Example**

Delete all jobs that are pending and need complete rebuild.

```

if (typeof catalog != undefined) {
    for (var i=0; i<catalog.jobs.length; i++){
        var job = catalog.jobs[i];
        console.println("Index: ", job.path);

        if (job.status == "Pending" && job.type == "Rebuild")
            catalog.remove(job);
    }
}

```

**CatalogJob Generic Object**

This generic JS object provides information about a job submitted to Catalog. It is returned by [index.build](#), and the [catalog.jobs](#) property, and passed to [catalog.remove](#).

It has the following properties:

Property	Type	Access	Description
<b>path</b>	String	R	Device independent path of the index associated with the job
<b>type</b>	String	R	Type of indexing operation associated with the job. Possible values are: Build Rebuild Delete
<b>status</b>	String	R	The status of the indexing operation. Possible values are: Pending Processing Completed CompletedWithErrors

## Certificate Object

The Certificate Object provides read-only access to the properties of an X.509 public key certificate.

Related objects and methods are:

Security Object: [importFromFile](#) and [getSecurityPolicies](#)

DirConnection Object: [search](#)

Field Object: [signatureInfo](#)

FDF Object: [signatureValidate](#)

RDN Generic Object

Usage Generic Object

**NOTE:** There are no security restrictions on this object.

## Certificate Properties

### binary

5.0			
-----	--	--	--

The raw bytes of the certificate, as a hex encoded string.

Type: String

Access: R.

### issuerDN

5.0			
-----	--	--	--

The distinguished name of the issuer of the certificate, returned as an [RDN Generic Object](#).

Type: RDN object

Access: R.

### keyUsage

6.0			
-----	--	--	--

An array of strings indicating the value of the certificate key usage extension. Possible values are

kDigitalSignature	kDataEncipherment	kCRLSign
kNonRepudiation	kKeyAgreement	kEncipherOnly
kKeyEncipherment	kKeyCertSign	kDecipherOnly

Type: Array of Strings

Access: R.

### MD5Hash

5.0			
-----	--	--	--

The MD5 digest of the certificate, represented as a hex-encoded string. This provides a unique fingerprint for this certificate.

Type: String

Access: R.

### SHA1Hash

5.0			
-----	--	--	--

The SHA1 digest of the certificate, represented as a hex -encoded string. This provides a unique fingerprint for this certificate.

*Type: String**Access: R.***serialNumber**

5.0			
-----	--	--	--

A unique identifier for this certificate, used in conjunction with [issuerDN](#).

*Type: String**Access: R.***subjectCN**

5.0			
-----	--	--	--

The common name of the signer.

*Type: String**Access: R.***subjectDN**

5.0			
-----	--	--	--

The distinguished name of the signer, returned as an [RDN Generic Object](#).

*Type: RDN object**Access: R.***ubRights**

7.0			
-----	--	--	--

The application rights that can be enabled by this certificate, returned as a generic [Rights Object](#).

*Type: Rights Object**Access: R.***Rights Object**

A **Rights** object has the following properties:

Property	Type	Access	Description
<b>mode</b>	String	R	Possible values are listed in the <a href="#">UbiquityMode</a> table below. Currently, this value is not used by Adobe's PDF viewer.

Property	Type	Access	Description
<b>rights</b>	Array of Strings	R	Array of strings indicating the application rights that can be enabled by this certificate. Possible values of the string elements of this array are listed in the <a href="#">UbiquityRights</a> table below.

### UbiquityMode

These are values of the **mode** property of the [Rights Object](#).

String	Description
<b>Evaluation</b>	Rights enabled by this certificate for this document are valid as long as this certificate is valid.
<b>Production</b>	Rights enabled by this certificate for this document are valid for eternity.

### UbiquityRights

These are values of the **rights** property of the [Rights Object](#).

String	Description
<b>FormFillInAndSave</b>	The right to fill in forms, excluding signature fields, and to save the modified file.
<b>FormImportExport</b>	The right to import and export form data.
<b>FormAddDelete</b>	The right to add or delete a form field.
<b>SubmitStandalone</b>	The right to submit a document outside a browser.
<b>SpawnTemplate</b>	The right to spawn page templates.
<b>Signing</b>	The right to sign existing form fields in a document.
<b>AnnotModify</b>	The right to create, delete and modify comments.
<b>AnnotImportExport</b>	The right to import and export annotations.
<b>BarcodePlaintext</b>	The right to encode the appearance of a form field as a plain text barcode.
<b>AnnotOnline</b>	Allow online commenting. Enables upload of any annotations in the document to a server. Enables download of annotations from a server. Does not enable the addition of these annotations into the document.
<b>FormOnline</b>	Enable forms-specific online mechanisms (e.g. SOAP or Active Data Object).

String	Description
<b>EModify</b>	The right to create, delete, modify and import a named embedded files. Does not apply to file attachment annotations..

## usage

6.0			
-----	--	--	--

The purposes for which this certificate may be used within the Acrobat environment returned as a [Usage Generic Object](#).

Type: *Usage Object*      Access: *R*.

### Usage Generic Object

This generic JS object represents a certificate usage value in the `certificate.usage` property. It has the following properties.

Property	Type	Access	Description
<b>endUserSigning</b>	Boolean	R	<b>true</b> if the certificate is useable for end-user signing.
<b>endUserEncryption</b>	Boolean	R	<b>true</b> if the certificate is useable for end-user encryption.

### Example

The following example shows how the `usage` property can be used. The result of this script execution will be that the currently open document is encrypted for everyone in the addressbook. Addressbook entries that contain sign-only certificates, CA certificates, no certificates at all, or are otherwise unsuitable for encryption, will not be included in the final recipient list.

```
var eng = security.getHandler( "Adobe.AAB" );
var dc = eng.directories[0].connect();
var recipients = dc.search();

var filteredRecipients = new Array();
for( i = 0; i < recipients.length; ++i ) {
    if( recipients[i].defaultEncryptCert &&
        recipients[i].defaultEncryptCert.usage.endUserEncryption ) {
        filteredRecipients[filteredRecipients.length] = recipients[i];
        continue;
    }
    if(recipients[i].certificates) {
        for( j = 0; j < recipients[i].certificates.length; ++j )
            if( recipients[i].certificates[j].usage.endUserEncryption ) {
```

```

        filteredRecipients[filteredRecipients.length]
        = recipients[i];
        continue;
    }
}
this.encryptForRecipients({ [userEntities: filteredRecipients] });

```

---

## Collab Object

This object represents the Collaboration functionality.

---

## Collab Methods

### addStateModel

6.0				
-----	--	--	--	--

Adds a new state model to Acrobat. A state model describes the valid states that an **annot** using the model can have (see the [Annot Object](#) for details about getting and setting the state of an **annot**). State models can be used to describe the workflow that a document review goes through and can be used for review management.

See also [removeStateModel](#), [getStateInModel](#) and [transitionToState](#).

#### Parameters

<b>cName</b>	A unique, language-independent identifier for the State Model.
<b>cUIName</b>	The display name of the state model used in the User Interface and should be localized.
<b>oStates</b>	The states in the state model, described by a <a href="#">States Object Literal</a> .
<b>cDefault</b>	(optional) One of the states in the model to be used as a default state if no other state is set. The default is for there to be no default state.
<b>bHidden</b>	(optional) Whether the state model should be hidden in the state model user interface. The default is <b>false</b> (the State Model is shown).
<b>bHistory</b>	(optional) Whether an audit history is maintained for the state model. Keeping an audit history requires more space in the file. The default is <b>true</b> .



**Returns**

Nothing

**States Object Literal**

This object literal represents a set of states in a state model, and is passed as the **oStates** parameter. The elements in the object literal are the unique state identifiers and the values are objects having the following properties:

<b>cUIName</b>	The UI (display name) for the state.
<b>oIcon</b>	(optional) An <a href="#">Icon Stream Generic Object</a> that will be displayed in the UI for the state.

**Example**

Add a new state model with a unique name of "ReviewStates":

```
Collab.addStateModel({
  cName: "ReviewStates",
  cUIName: "My Review",
  oStates:
  {
    "initial": {cUIName: "Haven't reviewed it"},
    "approved": {cUIName: "I approve"},
    "rejected": {cUIName: "Forget it"},
    "resubmit": {cUIName: "Make some changes"}
  },
  cDefault: "initial"
});
```

A state model can be removed with [Collab.removeStateModel](#).

**removeStateModel**

6.0				
-----	--	--	--	--

Removes a state model that was previously added by calling [addStateModel](#). Removing a state model does not remove the state information associated with individual **annots**—if the model is removed and added again, all of the state information for the **annots** will still be available.

See also [addStateModel](#), [getStateInModel](#) and [transitionToState](#).

**Parameters**


---

<b>cName</b>	A unique, language-independent identifier for the State Model that was used in <a href="#">addStateModel</a> .
--------------	--

---

**Returns**

Nothing

**Example**

Continuing the example in [addStateModel](#), we remove the state model "ReviewStates":

```
// Remove the state model
Collab.removeStateModel("ReviewStates");
```

**Color Object**

4.0			
-----	--	--	--

The **color** object is a convenience static object that defines the basic colors. These colors are accessed in JavaScripts via the **color** object. Use this object whenever you want to set a property or call a method that require a color array. The color object is defined in `AForm.js`.

**Color Arrays**

A color is represented in JavaScript as an array containing 1, 2, 4, or 5 elements corresponding to a Transparent, Gray, RGB, or CMYK color space, respectively. The first element in the array is a string denoting the color space type. The subsequent elements are numbers that range between zero and one inclusive. For example, the color red can be represented as `["RGB", 1, 0, 0]`.

Invalid strings or insufficient elements in a color array cause the color to be interpreted as the color black.

---

<b>Color Space</b>	<b>String</b>	<b>Number of Additional Elements</b>	<b>Description</b>
Transparent	"T"	0	A <i>transparent</i> color space indicates a complete absence of color and will allow those portions of the document underlying the current field to show through.

---

Color Space	String	Number of Additional Elements	Description
Gray	"G"	1	Colors in the <i>gray</i> color space are represented by a single value—the intensity of achromatic light. In this color space, 0 is black, 1 is white, and intermediate values represent shades of gray. For example, .5 represents medium gray.
RGB	"RGB"	3	Colors in the <i>RGB</i> color space are represented by three values: the intensity of the <i>red</i> , <i>green</i> , and <i>blue</i> components in the output. RGB is commonly used for video displays because they are generally based on red, green, and blue phosphors.
CMYK	"CMYK"	4	Colors in the <i>CMYK</i> color space are represented by four values, the amounts of the <i>cyan</i> , <i>magenta</i> , <i>yellow</i> , and <i>black</i> components in the output. This color space is commonly used for color printers, where they are the colors of the inks used in four-color printing. Only cyan, magenta, and yellow are necessary, but black is generally used in printing because black ink produces a better black than a mixture of cyan, magenta, and yellow inks, and because black ink is less expensive than the other inks.

## Color Properties

The color object defines the following colors:

Color Object	Keyword	Equivalent JS	Version
Transparent	<code>color.transparent</code>	[ "T" ]	
Black	<code>color.black</code>	[ "G", 0 ]	
White	<code>color.white</code>	[ "G", 1 ]	
Red	<code>color.red</code>	[ "RGB", 1, 0, 0 ]	
Green	<code>color.green</code>	[ "RGB", 0, 1, 0 ]	
Blue	<code>color.blue</code>	[ "RGB", 0, 0, 1 ]	
Cyan	<code>color.cyan</code>	[ "CMYK", 1, 0, 0, 0 ]	

Color Object	Keyword	Equivalent JS	Version
Magenta	<code>color.magenta</code>	[ "CMYK", 0, 1, 0, 0 ]	
Yellow	<code>color.yellow</code>	[ "CMYK", 0, 0, 1, 0 ]	
Dark Gray	<code>color.dkGray</code>	[ "G", 0.25 ]	4.0
Gray	<code>color.gray</code>	[ "G", 0.5 ]	4.0
Light Gray	<code>color.ltGray</code>	[ "G", 0.75 ]	4.0

**Example**

This example sets the text color of the field to red if the value of the field is negative, or to black if the field value is nonnegative.

```
var f = event.target; /* field that the event occurs at */
f.target.textColor = event.value < 0 ? color.red : color.black;
```

**Color Methods****convert**

5.0			
-----	--	--	--

Converts the colorspace and color values specified by the `color` object to the specified colorspace. Note that conversion to the gray colorspace is lossy in the same fashion that displaying a color TV signal on a black and white TV is lossy. The conversion of RGB to CMYK does not take into account any black generation or under color removal parameters.

**Parameters**

<b>colorArray</b>	Array of color values. See <a href="#">Color Arrays</a> .
<b>cColorspace</b>	The colorspace to which to convert.

**Returns**

A color array.

**Example**

The return value of the code line below is the array [ "CMYK", 0, 1, 1, 0 ].

```
color.convert ( ["RGB", 1, 0, 0], "CMYK" );
```

## equal

5.0			
-----	--	--	--

Compares two [Color Arrays](#) to see if they are the same. The routine performs conversions, if necessary, to determine if the two colors are indeed equal (for example, ["**RGB**", 1, 1, 0] is equal to ["**CMYK**", 0, 0, 1, 0]).

### Parameters

<b>colorArray1</b>	The first color array for comparison.
<b>colorArray2</b>	The second color array for comparison.

### Returns

**true** if the arrays represent the same color, **false** otherwise.

### Example

```
var f = this.getField("foo");
if (color.equal(f.textColor, f.fillColor))
    app.alert("Foreground and background color are the same!");
```

## Column Generic Object

This generic JS object contains the data from every row in a column. A column object is returned by `statement.getColumn` and `statement.getColumnArray`. See also the [ColumnInfo Generic Object](#).

It has the following properties.

Property	Type	Access	Description
<b>columnNum</b>	number	R	The number identifying the column.
<b>name</b>	string	R	The name of the column.
<b>type</b>	number	R	One of the <a href="#">SQL Types</a> for the data in the column.
<b>typeName</b>	string	R	The name of the type of data the column contains.
<b>value</b>	various	R/W	The value of the data in the column, in the format in which the data was originally retrieved.

## ColumnInfo Generic Object

This generic JS object contains basic information about a column of data, and is returned by `connection.getColumnList`. See also [Column Generic Object](#).

It has the following properties.

Property	Type	Access	Description
<code>name</code>	string	R	A string that represents the identifying name of a column. This string could be used in a <code>statement.getColumn</code> call to identify the associated column.
<code>description</code>	string	R	A string that contains database-dependent information about the column.
<code>type</code>	number	R	A numeric value identifying one of the <a href="#">ADBC SQL Types</a> that applies to the data contained in the column associated with the <code>ColumnInfo</code> object.
<code>typeName</code>	string	R	A string identifying the type of the data contained in the associated column. This is not the <a href="#">SQL Types</a> (see <code>type</code> above), but a database-dependent string representing the data type. This property may give useful information about user-defined data types.

## Connection Object

5.0				
-----	--	--	---	--

The `Connection` object encapsulates a session with a database. `Connection` objects are returned by `ADBC.newConnection`. See also the [ADBC Object](#), [Statement Object](#), [Column Generic Object](#), [ColumnInfo Generic Object](#), [Row Generic Object](#), and [TableInfo Generic Object](#).

## Connection Methods

### close

6.0			X	
-----	--	--	---	--

Closes an active connection and invalidates all the objects created from the connection.

#### Parameters

None

#### Returns

Nothing

### newStatement

5.0			X	
-----	--	--	---	--

Creates a [Statement Object](#) through which database operations may be performed.

#### Parameters

None

#### Returns

A [Statement](#) object on success or `null` on failure.

#### Example

```
// get a connection object, see newConnection
var con = ADBC.newConnection("q32000data");
// now get a statement object
var statement = con.newStatement();
var msg = (statement == null) ?
    "Failed to obtain newStatement!" : "newStatement Object obtained!";
console.println(msg);
```

### getTableList

5.0			X	
-----	--	--	---	--

Gets information about the various tables in a database.

#### Parameters

None

**Returns**

It returns an array of [ColumnInfo Generic Objects](#). This method never fails but may return a zero-length array.

**Example**

Assuming we have a Connection object (**con**) already in hand (see [newStatement](#) and [newConnection](#)), get the list of tables

```
var tableInfo = con.getTableList();
console.println("A list of all tables in the database.");
for (var i = 0; i < tableInfo.length; i++) {
    console.println("Table name: "+ tableInfo[i].name);
    console.println("Description: "+ tableInfo[i].description);
}
```

**getColumnList**

5.0			X	
-----	--	--	---	--

Gets information about the various columns in the table

**Parameters**

---

<b>cName</b>	The name of the table to get column information about.
--------------	--

---

**Returns**

Returns an array of [ColumnInfo Generic Objects](#). This method never fails but may return a zero-length array.

**Example**

Assuming we have a Connection object (**con**) already in hand (see [newStatement](#) and [newConnection](#)), get list of all column names.

```
var con = ADBC.newConnection("q32000data");
var columnInfo = con.getColumnList("sales");
console.println("Column Information");
for (var i = 0; i < columnInfo.length; i++) {
    console.println(columnInfo[i].name);
    console.println("Description: "+ columnInfo[i].description);
}
```



## Console Object



The **Console** object is a static object to access the JavaScript console for displaying debug messages and executing JavaScript. It does not function in the Adobe Reader previous to Adobe Reader 7.0, or in Acrobat Approval.

**NOTE:** Beginning with version 7.0, the Adobe Reader now has a console window. There is a preference under **Edit > Preferences > General > JavaScript** to “Show console on errors and messages”. The primary function of the console window in Reader is to report errors and messages. Though the console is not interactive, the methods of the **console** object function as they do in Acrobat Standard and Professional.

The debugging capability of the JavaScript Debugging window can be made available for Adobe Reader on Windows and Macintosh platform. In order to debug within Adobe Reader, the JavaScript file debugger.js needs to be installed, and the windows registry needs to be edited appropriately. See the [Acrobat JavaScript Scripting Guide](#) for the technical details.

See also the [Dbg Object](#).

## Console Methods

### show

3.01			
------	--	--	--

Shows the console window.

#### Parameters

None

#### Returns

Nothing

#### Example

Clear and show the console window:

```
console.clear();
console.show();
```

## hide

4.0			
-----	--	--	--

Closes the console window.

**Parameters**

None

**Returns**

Nothing

## println

3.01			
------	--	--	--

Prints a string value to the console window with an accompanying carriage return.

**Parameters**


---

<b>cMessage</b>	A string message to print.
-----------------	----------------------------

---

**Returns**

Nothing

**Example 1**

This example prints the value of a field to the console window. The script could be executed during a mouse up event.

```
var f = this.getField("myText");
console.clear(); console.show();
console.println("Field value = " + f.value);
```

**Example 2**

The console can be used as a debugging tool; you can write values of variables to the console, for example. The script below is taken from the document level.

```
var debugIsOn = true;
function myFunction ( n, m )
{
    if (debugIsOn)
    {
        console.println("Entering function: myFunction");
        console.println("  Parameter 1: n = " + n);
        console.println("  Parameter 2: m = " + m);
    }
    ....
    ....
```

```

        if (debugIsOn) console.println(" Return value: rtn = " + rtn);
        return rtn;
    }

```

Beginning with Acrobat 6.0, debugging can also be accomplished with the JavaScript Debugger. See [Dbg Object](#).

## clear

3.01			
------	--	--	--

Clears the console windows buffer of any output.

### Parameters

None

### Returns

Nothing

---

## Data Object

5.0			
-----	--	--	--

The Data Object is the representation of an embedded file or data stream that is stored in the document. Data objects are stored in the name tree in the document. See the section on the Names Tree and Embedded File Streams in the [PDF Reference](#) for details.

Data objects can be inserted from the external file system, queried, and extracted. This is a good way to associate and embed source files, metadata, and other associated data with a document.

See the following [Doc Object](#) properties and methods:

[createDataObject](#), [dataObjects](#), [exportDataObject](#), [getDataObject](#), [importDataObject](#), [removeDataObject](#), [openDataObject](#), [getDataObjectContents](#), [setDataObjectContents](#)

**NOTE:** While the methods for data objects were implemented in Acrobat 5.0, the ability to use these in an Adobe Reader additional usage rights only became available in Adobe Reader 6.0.

---

## Data Properties

### creationDate

The creation date of the file that was embedded.

*Type: Date*                      *Access:R.*

## **modDate**

The modification date of the file that was embedded.

*Type: Date*                      *Access:R.*

## **MIMETYPE**

The MIME type associated with this data object.

*Type: String*                      *Access:R.*

## **name**

The name associated with this data object.

*Type: String*                      *Access:R.*

### **Example**

```
console.println("Dumping all data objects in the document.");  
var d = this.dataObjects;  
for (var i = 0; i < d.length; i++)  
    console.println("DataObject[" + i + "]=" + d[i].name);
```

## **path**

The device-independent path to the file that was embedded.

*Type: String*                      *Access:R.*

## **size**

The size, in bytes, of the uncompressed data object.

*Type: Number*                      *Access:R.*

## DataSourceInfo Generic Object

This generic JS object contains basic information about a particular database. The **ADBC.getDataSourceList** method returns an array of these objects. The object has the following properties.

Property	Type	Access	Description
<b>name</b>	String	R	A string that represents the identifying name of a database. This string could be passed to <b>newConnection</b> to establish a connection to the database that the DataSourceInfo object is associated with.
<b>description</b>	String	R	A string that contains database dependent information about the database.

## Dbg Object

The Dbg Object is used to optionally control the JavaScript Debugger from a command-line console standpoint. The same functionality provided by the buttons in the JavaScript Debugger dialog toolbar available from the **dbg** methods. In addition, breakpoints can be created, deleted and inspected using the **dbg** object.

The **dbg** object and the JavaScript Debugger are only available in Acrobat Professional.

**NOTES:** Should the viewer lock up during a debugging session, pressing the Esc-key may resolve the problem.

Debugging is not possible with a model dialog open, this occurs, for example, when debugging a batch sequence.

Debugging script with an running event initiated by either **app.setInterval** or **app.setTimeout** may cause a recurring alert boxes to appear. Use the Esc-key after the model dialog is dismissed to resolve the problem.

(Version 7.0) While the Debugger is open, and a debugging session is under way, the Acrobat application will be unavailable.

## Dbg Properties

### bps

6.0						<b>P</b>
-----	--	--	--	--	--	----------

Returns an array of [Breakpoint Generic Objects](#), each element corresponding to a breakpoint set in the debugger.

Type: Array

Access:R.

#### Breakpoint Generic Object

This generic JS object contains basic information about a breakpoint, and is returned by the `Dbg.bps` property. It contains the following properties and methods:

Property	Type	Access	Description
<code>fileName</code>	string	R	A string that identifies the script in the debugger.
<code>condition</code>	string	R	A JavaScript expression evaluated whenever the debugger has to decide to stop or not at a breakpoint. Used to create conditional breakpoints. The default value for this property is the string "true".
<code>lineNum</code>	number	R	The line number in the script for which the breakpoint is set.
Method	Parameters	Returns	Description
<code>toString</code>	none	String	A string describing the breakpoint.

#### Example

List all currently active breakpoints.

```
var db = dbg.bps
for ( var i = 0; i < db.length; i++ )
{
    for ( var o in db[i] ) console.println(o + ": " + db[i][o]);
    console.println("-----");
}
```

See [sb](#) for another example of usage.

## Dbg Methods

### c

6.0						<b>P</b>
-----	--	--	--	--	--	----------

The **c** (continue) method resumes execution of a program stopped in the debugger. The JavaScript program may either stop again, depending on where the breakpoints are set, or reach execution end.

#### Parameters

None

#### Returns

Nothing

### cb

6.0	<b>D</b>					<b>P</b>
-----	----------	--	--	--	--	----------

The **cb** (clear breakpoint) method clears a breakpoint in the debugger.

#### Parameters

<b>fileName</b>	The name of the script from where the breakpoint is going to be deleted.
<b>lineNum</b>	The line number for the breakpoint that is going to be cleared in the script.

#### Returns

Nothing

### q

6.0						<b>P</b>
-----	--	--	--	--	--	----------

The **q** (quit) method quits debugging and executing the current JavaScript. It additionally dismisses the debugger dialog.

#### Parameters

None

**Returns**

Nothing

**sb**

6.0	Ⓓ				Ⓟ
-----	---	--	--	--	---

The **sb** (set breakpoint) method sets a new breakpoint in the debugger.

**Parameters**

<b>fileName</b>	The name of the script where the breakpoint is to be set.
<b>lineNum</b>	The line number where the breakpoint is going to be created in the script
<b>condition</b>	(optional) a JavaScript expression evaluated every time the debugger reaches a breakpoint . The decision to stop or not at a breakpoint is based on the result of evaluating such expression. If the expression evaluates to <i>true</i> , the debugger will stop at the breakpoint. If the expression evaluates to <b>false</b> , the debugger continues executing the script and will not stop at the breakpoint. The default value for this parameter is the string "true".

**Returns**

Nothing

**Example 1**

Some script is run and an exception is thrown due to some error. A breakpoint is programmatically set using the information given in the error message.

```
SyntaxError: missing ; before statement 213:Document-Level: myDLJS
// now set a breakpoint using the console
dbg.sb({
  fileName: "Document-Level: myDLJS",
  lineNum: 213,
  condition: "true"
});
```

**Example 2**

This example simulates the functionality of the "Store breakpoints in PDF file" checkbox in the **Preferences > General > JavaScript** dialog.

```
// save breakpoints in PDF file
this.addScript("myBreakpoints", "var myBPS = " + dbg.bps.toSource());

// now reset the breakpoints
for ( var i = 0; i < myBPS.length; i++ ) dbg.sb( myBPS[i] );
```



**Example 3**

Set a conditional break. Consider the following code, which is a mouse up action.

```
for (var i=0; i<100; i++)
    myFunction(i);           // defined at document level

// In the console, set a conditional break. Here, we break when the
// index of the loop is greater than 30.
dbg.sb({
    fileName:"AcroForm:Button1:Annot1:MouseUp:Action1",
    lineNum:2,
    condition:"i > 30"
})
```

**si**

6.0					<b>P</b>
-----	--	--	--	--	----------

The **si** (step in) method advances the program pointer to the next instruction in the JavaScript program, entering each function call that is encountered, and for which there is a script defined. Native JavaScript calls cannot be stepped into.

**Parameters**

None

**Returns**

Nothing

**sn**

6.0					<b>P</b>
-----	--	--	--	--	----------

The **sn** (step instruction) method advances the program pointer to the next byte-code in the JavaScript program. Each JavaScript instruction is made up of several byte-codes as defined by the JavaScript interpreter.

**Parameters**

None

**Returns**

Nothing

**so**

6.0					<b>P</b>
-----	--	--	--	--	----------

The **so** (step out) method executes the program until it comes out of the current function. It stops executing in the instruction immediately following the call to the function. If the scope currently under debug is the top level scope, the program may continue executing until it ends, or stop again when it reaches a breakpoint.

**Parameters**

None

**Returns**

Nothing

**sv**

6.0					<b>P</b>
-----	--	--	--	--	----------

The **sv** (step over) method advances the program pointer to the next instruction in the JavaScript program. If a function call is encountered, the debugger will not step into the instructions defined inside that function.

**Parameters**

None

**Returns**

Nothing

---

## Dialog Object

An instance of this object is passed as a parameter to the **initialize**, **validate**, **commit**, **destroy** and **ItemID** methods of the Dialog Descriptor object literal that is passed to **app.execDialog()**, see [Dialog Handlers](#). The Dialog object allows the current state of the Dialog to be queried and set.

## Dialog Methods

### enable

7.0						
-----	--	--	--	--	--	--

This method enables/disables various dialog elements using the object literal passed in. For each dialog item to modify, there should be an entry in the object literal with the Dialog *ItemID* as the label and a boolean as the value indicating if it is enabled or not.

Typically, `enable()` is called in the `initialize()` method (see see [Dialog Handlers](#)) of the object literal passed to `app.execDialog()` to preset whether various dialog elements are enabled or not.

#### Parameters

---

object literal

---

#### Returns

Nothing

#### Example

See the examples following `app.execDialog()`.

### end

7.0						
-----	--	--	--	--	--	--

This method terminates a currently executing dialog (as if the cancel button had been pressed). This method takes an optional parameter of the *ItemID*, a string, of the dialog element that will be reported as dismissing the dialog. This *ItemID* will be the return value of the `app.execDialog()` call which created the dialog.

#### Parameters

---

String	(optional) The <i>ItemID</i> of the dialog element that will be reported as dismissing the dialog.
--------	--

---

#### Returns

Nothing

#### Example

See the examples following `app.execDialog()`.

## load

7.0					
-----	--	--	--	--	--

This method *sets* the values of dialog elements using the object literal passed in. Dialog items are identified by an *ItemID* which is a unique 4 character string. For each dialog item to be modified, there should be an entry in the object literal with the *ItemID* as the label and the dialog element setting as the contents. If the dialog element takes multiple values (for example, a **list\_box** or a **popup**) then the value should be an object literal consisting of the displayed entry as the label and a numeric value as the contents. Similarly, if the dialog element is hierarchical in nature (for example, a **hier\_list\_box**) then the value should be a set of nested object literals. If the numeric value is greater than 0, then the item is selected, otherwise it is not selected.

Typically, `load()` is called in the `initialize()` method (see see [Dialog Handlers](#)) of the object literal passed to `app.execDialog()` to preset the value of various dialog elements.

### Parameters

---

object literal
----------------

---

### Returns

Nothing

### Example

See the examples following `app.execDialog()`.

## store

7.0					
-----	--	--	--	--	--

This method *gets* the values of dialog elements as an object literal returned. Dialog items are identified by an *ItemID* which is a unique 4 character string. For each dialog element in the dialog there will be an entry in the object literal with the *ItemID* as the label and the dialog element setting as the contents. If the dialog element takes multiple values (for example, a **list\_box** or a **popup**) then the value should be an object literal consisting of the displayed entry as the label and a numeric value as the contents. If the numeric value is greater than 0, then the item was selected, otherwise it was not selected.

Typically, `store()` is called in the `commit()` method (see see [Dialog Handlers](#)) of the object literal passed to `app.execDialog()` to extract the value of various dialog elements.

### Parameters

None

**Returns**

object literal

**Directory Object**

6.0		Ⓢ			
-----	--	---	--	--	--

Directories are a repository of user information, including public-key certificates. Directory Objects provide directory access and are obtained using the [directories](#) property or the [newDirectory](#) method of the [SecurityHandler Object](#).

Acrobat 6.0 provides several directories. The *Adobe.AAB* Security Handler has a single directory named *Adobe.AAB.AAB*. This directory provides access to the local Acrobat Address Book, also called the *Trusted Identity Store*. On Windows, the *Adobe.PPKMS* Security Handler provides access, via *Microsoft Active Directory Script Interface* (ADSI) to as many directories as have been created by the user. These directories are created sequentially with names *Adobe.PPKMS.ADSI.dir0*, *Adobe.PPKMS.ADSI.dir1*, and so on.

**NOTE:** (Security Ⓢ) This object can only be obtained from a [SecurityHandler Object](#) and is thus governed by the security restrictions of the [SecurityHandler Object](#). The [Directory Object](#) is therefore available only for batch, console, application initialization and menu execution, including in Acrobat Reader. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Directory Properties**

**info**

6.0		Ⓢ		
-----	--	---	--	--

The value of this property is a [DirectoryInformation Generic Object](#), a generic object used to set and get the properties for this [Directory Object](#).

Type: *Object*

Access: *R/W*.

**Example**

```
// Create and activate a new directory
var oDirInfo = { dirStdEntryID: "dir0",
  dirStdEntryName: "Employee LDAP Directory",
  dirStdEntryPrefDirHandlerID: "Adobe.PPKMS.ADSI",
  dirStdEntryDirType: "LDAP",
```

```

server: "ldap0.acme.com",
port: 389 };
var sh = security.getHandler( "Adobe.PPKMS" );
var newDir = sh.newDirectory();
newDir.info = oDirInfo;

```

### DirectoryInformation Generic Object

A directory information object is a generic object representing the properties for a directory and has the following standard properties:

---

#### Standard Directory Information Object properties

Property	Type	Access	Required	Description
<b>dirStdEntryID</b>	String	R/W	Yes	A unique, language independent name for the directory. Must be alphanumeric and can include underscores, periods and hyphens. For new directory objects it is suggested that the ID not be provided, in which case a new unique name will be automatically generated.
<b>dirStdEntryName</b>	String	R/W	Yes	A user friendly name for the directory.
<b>dirStdEntryPrefDirHandlerID</b>	String	R/W	No	The name of the directory handler that is to be used by this directory. Security handlers can support multiple directory handlers for multiple directory types (eg. local directories, LDAP directories).
<b>dirStdEntryDirType</b>	String	R/W	No	The type of directory. An example of this would be LDAP, ADSI, WINNT.

---

**Standard Directory Information Object properties**

Property	Type	Access	Required	Description
<code>dirStdEntryVersion</code>	String	R	No	The version of the data. The default value is 0 if this is not set by the directory. The value for Acrobat 6.0 directories for the <i>Adobe.AAB</i> and <i>Adobe.PPKMS.ADSI</i> directory handlers is <code>0x00010000</code> .

Directory information objects can include additional properties that are specific to a particular directory handler. The *Adobe.PPKMS.ADSI* directory handler includes the following additional properties:

**Adobe.PPKMS.ADSI additional directory information object properties**

Property	Type	Access	Description
<code>server</code>	String	R/W	The server that hosts the data. For example, <code>addresses.employees.xyz.com</code> .
<code>port</code>	Number	R/W	The port number for the server. The standard LDAP port number is 389.
<code>searchBase</code>	String	R/W	Narrows down the search to a particular section of the directory. An example of this would be <code>o=XYZ Systems,c=US</code> .
<code>maxNumEntries</code>	Number	R/W	The maximum number of entries that would be retrieved in a single search.
<code>timeout</code>	Number	R/W	The maximum time allowed for a search.

**Example 1**

Create and activate a new directory.

```
var oDirInfo = { dirStdEntryID: "dir0",
  dirStdEntryName: "Employee LDAP Directory",
  dirStdEntryPrefDirHandlerID: "Adobe.PPKMS.ADSI",
  dirStdEntryDirType: "LDAP",
  server: "ldap0.acme.com",
  port: 389
};
var sh = security.getHandler( "Adobe.PPKMS" );
var newDir = sh.newDirectory();
newDir.info = oDirInfo;
```

**Example 2**

Get information for existing directory.

```
var sh = security.getHandler("Adobe.PPKMS");
var dir0 = sh.directories[0];
// Get directory info object just once for efficiency
var dir0Info = dir0.info;
console.println( "Directory " + dir0Info.dirStdEntryName );
console.println( "address " + dir0Info.server + ":" + dir0Info.port );
```

## Directory Methods

### connect

6.0		Ⓢ		
-----	--	---	--	--

Returns a [DirConnection Object](#) that is a connection to the directory with the specified name. There can be more than one active connection for a directory.

See also [DirConnection Object](#) and the SecurityHandler Object's [directories](#) property.

#### Parameters

<b>oParams</b>	(optional) A generic object that can contain parameters that are necessary in order to create the connection. Properties of this object are dependent on the particular directory handler and can include <b>userid</b> and <b>password</b> .
<b>bUI</b>	(optional) A boolean value that defaults to <b>false</b> . It conveys to the directory handler if it could bring its UI in case that is required for establishing the connection.

#### Returns

A [DirConnection Object](#), or **null**, if there is no directory with the specified name.

#### Example:

Enumerate available directories and connect.

```
var sh = security.getHandler( "Adobe.PPKMS" );
var dirList = sh.directories;
for ( var i=0; i< dirList.length; i++)
    for ( var o in dirList[i].info )
        console.println( o + " = " + dirList[i].info[o] );
var dirConnection = dirList[0].connect();
```



## DirConnection Object

6.0		Ⓢ			
-----	--	---	--	--	--

The **DirConnection** object represents an open connection to a directory: a repository of user information, including public-key certificates. Directory connections are opened using the **Directory Object**'s **connect** method. A directory with a particular name can have more than one connection open at a time. All **DirConnection** objects must support all properties and methods listed here, unless otherwise specified.

**NOTES:** (Security Ⓢ) : This object can only be obtained from a **Directory Object** and is thus governed by the security restrictions of the **Directory Object**. The **DirConnection Object** is therefore available only for batch, console, application init and menu exec, including in Acrobat Reader. See also [Privileged versus Non-privileged Context](#).

## DirConnection Properties

### canList

6.0		Ⓢ		
-----	--	---	--	--

Indicates whether the directory connection is capable of listing all of its entries. Some directories may contain too many entries for this operation to be practical.

*Type: Boolean*

*Access: R.*

#### Example

The AAB directory allows listing of the local trusted identity list

```
var sh = security.getHandler( "Adobe.AAB" );
var dc = sh.directories[0].connect();
console.println( "CanList = " + dc.canList );
```

### canDoCustomSearch

6.0		Ⓢ		
-----	--	---	--	--

Whether the directory connection supports search using directory-specific search parameter attributes. As an example, directory-specific attributes for an LDAP directory include: o (organization), c (country), cn (common name), givenname, sn (surname), uid, st, postalcode, mail, and telephonenumber.

*Type: Boolean*

*Access: R.*

**canDoCustomUISearch**

6.0		Ⓢ		
-----	--	---	--	--

Whether the directory connection supports search using its own custom user interface to collect the search parameters.

*Type: Boolean*

*Access: R.*

**canDoStandardSearch**

6.0		Ⓢ		
-----	--	---	--	--

Whether the directory connection supports search using standard search parameter attributes. The standard attributes are

firstName  
 lastName  
 fullName  
 email  
 certificates

Some directory database implementations may not support these attributes, but directory handlers are free to translate these attributes to names understood by the directory.

*Type: Boolean*

*Access: R.*

**groups**

6.0		Ⓢ		
-----	--	---	--	--

Returns an array of language dependent names for groups that are available through this connection.

*Type: Array*

*Access: R.*

**name**

6.0		Ⓢ		
-----	--	---	--	--

Returns the language independent name of the directory that this object is connected to. An example of this would be `Adobe.PPKMS.ADSI.dir0`. All DirConnection objects must support this property.

*Type: String*

*Access: R.*

## uiName

6.0		Ⓢ		
-----	--	---	--	--

Returns the language dependent string of the directory this object is connected to. This string is suitable for user interfaces. An example of this would be XYZ's Employees. All DirConnection objects must support this property.

Type: String

Access: R.

## DirConnection Methods

### search

6.0		Ⓢ		
-----	--	---	--	--

Searches the directory and returns an array of [UserEntity Generic Objects](#) that match the search parameters. A [UserEntity Generic Object](#) is a generic object that contains properties for all attributes that were requested via the [setOutputFields](#) method. If the [setOutputFields](#) method is not called prior to a search it would return a [UserEntity Generic Object](#) containing no entries.

#### Parameters

<b>oParams</b>	(optional) A generic object containing an array of key-value pairs consisting of search attribute names and their corresponding strings. If <b>oParams</b> is not provided and <a href="#">canList</a> is <b>true</b> for this directory then all entries in the directory will be returned. If <b>oParams</b> is not provided and <a href="#">canList</a> is <b>false</b> , an exception occurs.
<b>cGroupName</b>	(optional) The name of a group (not to be confused with <a href="#">Group Objects</a> ). If specified then the search will be restricted to this group.
<b>bCustom</b>	(optional) If <b>false</b> (the default), <b>oParams</b> contains standard search attributes. The <a href="#">canDoStandardSearch</a> property must be <b>true</b> , or an exception occurs. If <b>true</b> , then <b>oParams</b> contains directory-specific search parameters. The <a href="#">canDoCustomSearch</a> property must be <b>true</b> , or an exception occurs.
<b>bUI</b>	(optional) If <b>true</b> , the handler shows user interface to allow collection of search parameters. The results of the search are returned by this method. <a href="#">canDoCustomUISearch</a> must also be <b>true</b> if <b>bUI</b> is <b>true</b> , or an exception will occur. If <b>bUI</b> is specified then <b>bCustom</b> must also be specified, though its value is ignored.

**Returns**

An array of [UserEntity Generic Objects](#).

**Example 1**

Directory search

```
var sh = security.getHandler( "Adobe.PPKMS" );
var dc= sh.directories[0].connect();
dc.setOutputFields( {oFields:["certificates","email"]} )
var retVal = dc.search({oParams:{lastName:"Smith"}});
if( retVal.length )
console.println( retVal[0].email );
```

**Example 2**

List all entries in local Acrobat Address Book. The script searches the directory and returns an array of users, along with their certificate information.

```
var sh = security.getHandler( "Adobe.AAB" );
var dc = sh.directories[0].connect();
if( dc.canList ) {
    var x = dc.search();
    for( j=0; j<x.length; ++j ) {
        console.println("Entry[" + j + "] = " + x[j].fullName + ":");
        for(i in x[j]) console.println("  " + i + " = " + x[j][i]);
    }
}
```

**UserEntity Generic Object**



A generic JS object that describes a user in a directory and the user's associated certificates. It contains standard properties that have a specific meaning for all directory handlers. Directory handlers translate these entries to the ones that are specific to them when required. An array of these objects is returned by [dirConnection.search](#).

It has the following properties.

Property	Type	Access	Description
<b>firstName</b>	String	R/W	The first name for the user.
<b>lastName</b>	String	R/W	The last name of the user.
<b>fullName</b>	String	R/W	The full name of the user.
<b>certificates</b>	Array of <a href="#">Certificate Objects</a>	R/W	An array of certificates that belong to this user. To find a certificate that is to be used for a particular use, the caller should inspect the certificate's <a href="#">keyUsage</a> property.

Property	Type	Access	Description
<b>defaultEncryptCert</b>	Array of <a href="#">Certificate Objects</a>	R/W	The preferred certificate to use when encrypting documents for this user entity. Routines that process <b>User Entity Objects</b> will look first to this property when choosing an encryption certificate: if this property is not set then the first valid match in the certificates property will be used.

## setOutputFields

6.0				
-----	--	---	---	--

Defines the list of attributes that should be returned when executing the [search](#) method.

**NOTE:** This method is not supported by the *Adobe.AAB* directory handler. Custom options are not supported by the *Adobe.PPKMS.ADSI* directory handler.

### Parameters

<b>oFields</b>	An array of strings containing the names of attributes that should be returned from the directory when calling the search method. The names in this array must either be names of standard attributes that can be used for all directory handlers, or custom attributes that are defined for a particular directory. The standard attributes are the property names defined for the <a href="#">UserEntity Generic Object</a> . Directory handlers can, when desired, translate standard attribute names to names that it understands.
<b>bCustom</b>	(optional) A boolean indicating that the names in <b>oFields</b> are standard output attribute names. If <b>true</b> then the names represent directory-specific attributes that are defined for a particular directory handler. The default is <b>false</b> .

### Returns

An array of strings, containing the names of attributes from **oFields** that are not supported by this directory. An empty array is returned if the **oFields** array is empty.

### Example

In this example, **dc.setOutputFields()** returns the array of strings **["x", "y"]**.

```
var sh = security.getHandler("Adobe.PPKMS");
var dc = sh.directories[0].connect();
var w = dc.setOutputFields( [ "certificates", "email", "x", "y" ] );
```

```
console.println( w );
```

See also the examples that follow the `DirConnection.search` method

---

## Doc Object

The JavaScript `doc` object provides the interface between a PDF document open in the viewer and the JavaScript interpreter. It provides methods and properties of the PDF document.

### Doc Access from JavaScript

You can access the `doc` object from JavaScript in a variety of ways.

- The most common way is through the `this Object`, which usually points to the `doc` object of the underlying document.
- Some properties and methods return `doc` objects, `activeDocs`, `openDoc`, or `extractPages` all return `doc` objects.
- JavaScript is executed as a result of some event. For each event, an `Event Object` is created. A `doc` object can often be accessed through `event.target`:
  - For `mouse`, `focus`, `blur`, `calculate`, `validate`, and `format` events, `event.target` returns the `Field Object` that initiated the event. You can then access the `doc` object through `field.doc`.
  - For all other events, `event.target` points to the `doc` object.

#### Example 1: Access through *this* object

Use `this` to get the number of pages in this document:

```
var nPages = this.numPages;  
// get the crop box for "this" document:  
var aCrop = this.getPageBox();
```

#### Example 2: Access through return values

Return values from one document to open, modify, save and close another.

```
// path relative to "this" doc:  
var myDoc = app.openDoc("myNovel.pdf", this);  
myDoc.info.Title = "My Great Novel";  
myDoc.saveAs(myDoc.path);  
myDoc.closeDoc(true);
```

#### Example 3: Access through the event object.

For mouse, calculate, validate, format, focus, and blur events:

```
var myDoc = event.target.doc;
```

For all other events (for example, batch or console events):

```
var myDoc = event.target;
```

## Doc Properties

### alternatePresentations

6.0				
-----	--	--	--	--

References the document's [AlternatePresentation Object](#). If the functionality needed to display alternate presentations is not available, this property is **undefined**.

The **alternatePresentation** object provides access to the document's alternate presentations. The PDF language extension specifies that each document can potentially have many named alternate presentations. Each alternate presentation with a known **type** will have a corresponding **doc.alternatePresentations** property in the document. This property should have the same name as its alternate presentation and should reference its alternate presentation's [AlternatePresentation Object](#). If there are no recognized alternate presentations in the document, this object is empty (does not have any properties).

Section 9.4, titled "Alternate Presentations", of the [PDF Reference](#) provide details on this topic.

**NOTE:** For compatibility with current implementation alternate presentation name must be an ASCII string. The only alternate presentation type currently implemented is "SlideShow".

See the [AlternatePresentation Object](#) for properties and methods that can be used to control an alternate presentation.

*Type: Object | undefined    Access: R.*

#### Example 1

Test whether the **alternatePresentations** object is present:

```
if( typeof this.alternatePresentations != "undefined" )
{
    // assume AlternatePresentations are present
    // list the names of all alternate presentations in the doc
    for ( var ap in this.alternatePresentations ) console.println(ap);
}
```

#### Example 2

Assume there is a named presentation "MySlideShow" within the document.

```
// oMySlideShow is an AlternatePresentation object
oMySlideShow = this.alternatePresentations["MySlideShow"];
oMySlideShow.start();
```

## author

ⓧ	Ⓓ		ⓧ	
---	---	--	---	--

The author of the document. See [info](#), which supersedes this property in later versions.

**NOTE:** This property is read-only in Adobe Reader.

*Type: String*

*Access: R/W.*

## baseURL

5.0	Ⓓ		
-----	---	--	--

The base URL for the document is used to resolve relative web links within the document. See also [URL](#).

*Type: String*

*Access: R/W.*

### Example

This example sets the base URL, creates a link to go to a page relative to the base URL.

```
console.println("Base URL was " + this.baseURL);
this.baseURL = "http://www.adobe.com/products/";
console.println("Base URL is " + this.baseURL);
// add a link to the first page
var link = this.addLink(0, [200,200, 400, 300])
// set action that goes to the Acrobat page on the Adobe web site.
link.setAction("this.getURL('acrobat', false)")
```

## bookmarkRoot

5.0			
-----	--	--	--

The root bookmark for the bookmark tree. This bookmark is not displayed to the user; it is a programmatic construct used to access the tree and the child bookmarks.

*Type: object*

*Access: R.*

### Example

See the [Bookmark Object](#) for an example of usage.



## calculate

4.0			
-----	--	--	--

If **true**, allows calculations to be performed for this document. If **false**, prevents all calculations from happening for this document. Its default value is **true**. This property supersedes the **app.calculate**, whose use is now discouraged.

Type: *Boolean*

Access: *R/W*.

## creationDate

ⓧ			
---	--	--	--

The document's creation date. See [info](#), which supersedes this property in later versions.

Type: *Date*

Access: *R*.

## creator

ⓧ			
---	--	--	--

The creator of the document (for example, "Adobe FrameMaker", "Adobe PageMaker", and so on). See [info](#), which supersedes this property in later versions.

Type: *String*

Access: *R*.

## dataObjects

5.0			
-----	--	--	--

An array containing all the named data objects in the document.

Related objects, properties and methods are the [Data Object](#), [doc.openDataObject](#), [doc.getDataObject](#), [doc.createDataObject](#), [doc.importDataObject](#), [doc.removeDataObject](#), [doc.getDataObjectContents](#) and [doc.setDataObjectContents](#).

Type: *Array*

Access: *R*.

### Example

List all embedded files in the document.

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

## delay

4.0			
-----	--	--	--

This boolean property can delay the redrawing of any appearance changes to every field in the document. It is generally used to buffer a series of changes to fields before requesting that the fields regenerate their appearance. When **true**, forces all changes to be queued until **delay** is reset to **false**. Once set to **false**, all the fields on the page are redrawn.

See also the **field.delay** property.

Type: Boolean

Access: R/W.

## dirty

3.01	Ⓣ		ⓧ
------	---	--	---

This boolean property can be used to determine whether the document has been dirtied as the result of a changes to the document, and therefore needs to be saved. It is useful to reset the **dirty** flag in a document when performing changes that do not warrant saving, for example, updating a status field in the document.

**NOTE:** If the document is temporary or newly created, setting **doc.dirty** to **false** has no effect, that is, the user is still asked to save changes before closing the document. See **doc.requiresFullSave** in this regard.

Type: Boolean

Access: R/W.

### Example 1

This example resets the form and sets the **doc.dirty** to **false**. After the reset, the user can close the document without having to dismiss a save dialog.

```
var f = this.getField("MsgField");
f.value = "You have made too many mistakes, I'm resetting the form. "
        + "Start over, this time follow the directions!";
this.resetForm();
this.dirty = false;
```

### Example 2

In this example, a text field is filled that informs the user to complete the form. The script is constructed so that the populating the field does not change the save state of the document.

```
var f = this.getField("MsgField");
var b = this.dirty;
f.value = "Please fill in the fields below.";
this.dirty = b;
```

## disclosed

5.05		Ⓢ	
------	--	---	--

A boolean property that determines whether the document should be accessible to JavaScripts in other documents.

The two methods `app.openDoc` and `app.activeDocs` check the `disclosed` property of the document before returning its [Doc Object](#).

**NOTE:** (Security Ⓢ): The `disclosed` property can only be set during batch, console, Page/Open and Doc/Open events. See the [Event Object](#) for a discussion of Acrobat JavaScript events. See also [Privileged versus Non-privileged Context](#).

Type: *Boolean*

Access: *R/W*.

### Example 1

A document can be disclosed to others by placing the code at the document level (or as a page open action) at the top level:

```
this.disclosed = true;
```

### Example 2

The following code can be used in a Execute JavaScript Batch Sequence to disclose all selected documents.

```
this.addScript("Disclosed", "this.disclosed = true;");
```

## docID

6.0				
-----	--	--	--	--

The value of this property is an array of two strings. The format of each string is hex encoded binary. The first string is a permanent identifier based on the contents of the file at the time it was originally created, and does not change when the file is incrementally updated. The second string is a changing identifier based on the file's contents at the time it was last updated. These identifiers are defined by the optional **ID** entry in a PDF file's trailer dictionary. See Section 10.3 of [PDF Reference](#) for more details.

Type: *Array*

Access: *R*.

See ["Example 6 \(Version 7.0\)" on page 315](#) for an example of usage.

## documentFileName

6.0				
-----	--	--	--	--

The base filename with extension of the document referenced by the `doc` object. The device-independent path is not returned. See also `doc.path` and `doc.URL`. The file size of the document can be obtained from `doc.filesize`.

Type: *String*

Access: *R*.

### Example

Executing the script

```
console.println("The filename of this document is '
+ this.documentFileName +'.");
```

on this document, the *Acrobat JavaScript Scripting Reference*, yields

```
"The filename of this document is AcroJS.pdf."
```

## dynamicXFAForm

7.0				
-----	--	--	--	--

Returns **true** if the document is XFA, and it is dynamic . Returns **false** otherwise.

A dynamic XFA form is one in which some of the fields can grow or shrink in size to accomodate the values they contain.

Type: *Boolean*

Access: *R*.

### Example

See the [XFAObject Object](#) for an example of usage.

## external

4.0			
-----	--	--	--

Whether the current document is being viewed in the Acrobat application or in an external window (such as a web browser).

Type: *Boolean*

Access: *R*.

### Example

```
if ( this.external )
{
    // viewing from a browser
}
else
```

```
{
    // viewing in the Acrobat application.
}
```

## filesize

3.01			
------	--	--	--

The file size of the document in bytes.

*Type: Integer*

*Access: R.*

### Example (Version 5.0)

Get a readout of difference in file sizes before and after saving a document.

```
// add the following code to the "Document Will Save" section
var filesizeBeforeSave = this.filesize
console.println("File size before saving is " + filesizeBeforeSave);

// add the following code to the "Document Did Save" section
var filesizeAfterSave = this.filesize
console.println("File size after saving is " + filesizeAfterSave);
var difference = filesizeAfterSave - filesizeBeforeSave;
console.println("The difference is " + difference );
if ( difference < 0 )
    console.println("Reduced filesize!");
else
    console.println("Increased filesize!");
```

## hidden

7.0				
-----	--	--	--	--

This property is **true** if the document's window is hidden. A document's window may be hidden by virtue of being operated on through batch, if it was explicitly opened hidden, or if there is no AVDoc associated with it. See [app.openDoc](#) and [doc.openDataObject](#) for methods that can be used to open a document with a hidden window.

*Type: Boolean*

*Access: R.*

### Example

Open a document and verify its hidden status.

```
oDoc = app.openDoc({
    cPath: "/C/myDocs/myHidden.pdf",
    bHidden: true
});
console.println("It is " + oDoc.hidden + " that this document hidden.");
oDoc.closeDoc();
```

## icons

5.0			
-----	--	--	--

The value of `doc.icons` is the array of named [Icon Generic Objects](#) that are present in the document level named icons tree. If there are no named icons in the document, the property has a value of `null`.

See also [addIcon](#), [getIcon](#), [importIcon](#), [removeIcon](#), the [Field Object](#) properties [buttonGetIcon](#), [buttonImportIcon](#), [buttonSetIcon](#), and the [Icon Generic Object](#).

Type: `Array | null`      Access: *R*.

### Example 1

```
if (this.icons == null)
    console.println("No named icons in this doc");
else
    console.println("There are " + this.icons.length
        + " named icons in this doc");
```

### Example 2

```
// list all named icons
for (var i = 0; i < this.icons.length; i++) {
    console.println("icon[" + i + "]=" + this.icons[i].name);
}
```

## info

### In Adobe Reader

5.0			
-----	--	--	--

For the Adobe Reader, `doc.info` returns an object with properties from the document information dictionary in the PDF file. Standard entries are:

- Title
- Author
- Subject
- Keywords
- Creator
- Producer
- CreationDate
- ModDate
- Trapped

See Table 10.2, "Entries in a document information dictionary," in the [PDF Reference](#), for more details.

Writing to any property in this object in the Adobe Reader throws an exception.

Type: object

Access: R.

**Example**

```
// get title of document
var docTitle = this.info.Title;
```

**In Acrobat**

5.0	Ⓓ		ⓧ	
-----	---	--	---	--

For Acrobat, properties of the **info** object are writeable, and setting a property in this object will dirty the document. Additional document information fields can be added by setting non-standard properties.

**NOTE:** Standard entries are case insensitive, that is, **doc.info.Keywords** is the same as **doc.info.keywords**.

Type: object

Access: R/W.

**Example**

The following script

```
this.info.Title = "JavaScript, The Definitive Guide";
this.info.ISBN = "1-56592-234-4";
this.info.PublishDate = new Date();
for (var i in this.info)
    console.println(i + ": " + this.info[i]);
```

could produce the following output:

```
CreationDate: Mon Jun 12 14:54:09 GMT-0500 (Central Daylight Time) 2000
Producer: Acrobat Distiller 4.05 for Windows
Title: JavaScript, The Definitive Guide
Creator: FrameMaker 5.5.6p145
ModDate: Wed Jun 21 17:07:22 GMT-0500 (Central Daylight Time) 2000
SavedBy: Adobe Acrobat 4.0 Jun 19 2000
PublishDate: Tue Aug 8 10:49:44 GMT-0500 (Central Daylight Time) 2000
ISBN: 1-56592-234-4
```

**innerAppWindowRect**

6.0				
-----	--	--	--	--

This property returns the rectangle, an array of screen coordinates, for the Acrobat inner application window. The application window is available as an outer rectangle as well. The outer rectangle includes any title bar, resizing border, or the like, and the inner rectangle does not include these items.

Type: Array of Numbers

Access: R.

**Example:**

```
var coords = this.innerAppWindowRect;
console.println(coords.toSource())
// possible output: [115, 154, 1307, 990]
```

See also [innerDocWindowRect](#), [outerAppWindowRect](#) and [outerDocWindowRect](#).

## innerDocWindowRect

6.0				
-----	--	--	--	--

This property returns the rectangle, an array of screen coordinates, for the Acrobat inner document window. The document window is also available as an outer rectangle as well. The outer rectangle includes any title bar, resizing border, or the like, and the inner rectangle does not include these items.

These rectangles may differ quite a bit on different platforms. For example, on Windows, the doc window is always inside the app window, while on the Macintosh they are the same.

Type: Array of Numbers

Access: R.

See also [innerAppWindowRect](#), [outerAppWindowRect](#), [outerDocWindowRect](#) and [pageWindowRect](#)

## keywords

ⓧ	ⓓ		ⓧ	
---	---	--	---	--

The keywords that describe the document (for example, "forms", "taxes", "government"). See [info](#), which supersedes this property in later versions.

**NOTE:** This property is read-only in the Adobe Reader.

Type: object

Access: R/W.

## layout

5.0			
-----	--	--	--

Changes the page layout of the current document. Valid values are:

- SinglePage
- OneColumn
- TwoColumnLeft
- TwoColumnRight

In Acrobat 6.0, there are two additional properties:



```
TwoPageLeft
TwoPageRight
```

*Type: String*                      *Access: R/W.*

### Example

Put the document into a continuous facing layout, the first page of the document appears in the left column.

```
this.layout = "TwoColumnLeft";
```

## media

6.0				
-----	--	--	--	--

Each document has its own **doc.media** object, which contains properties that are specific to a particular document. **doc.media** also contains methods that apply to a document. The section on the [Doc.media Object](#) contains the documentation of the properties and methods of this object.

*Type: DocMedia Object*    *Access: R/W.*

## metadata

6.0			X	
-----	--	--	---	--

Allows you to access the XMP metadata embedded in a PDF document. Returns a string containing the XML text stored as metadata in a particular PDF document. For information on embedded XMP metadata, see section 9.6 of the [PDF Reference](#). This property throws a **RaiseError** if the user tries to set the property to a string that is not in the XMP metadata format.

*Type: String*                      *Access: R/W.*

### Exceptions

**RaiseError** is thrown if setting metadata to a string not in XMP format.

### Example 1

Try to create metadata not in XMP format.

```
this.metadata = "this is my metadata";
RaiseError: The given metadata was not in the XMP format
Global.metadata:1:Console undefined:Exec
===> The given metadata was not in the XMP format
```

### Example 2

Create a PDF report file with metadata from a document.

```
var r = new Report();
r.writeText(this.metadata);
r.open("myMetadataReportFile");
```

## modDate

ⓧ			
---	--	--	--

The date the document was last modified. See [info](#), which supersedes this property in later versions.

*Type: Date*

*Access: R.*

## mouseX

7.0			
-----	--	--	--

Gets the x-coordinate of the mouse coordinates in default user space in relation to the current page.

*Type: Number*

*Access: R.*

### Example

Get the coordinates of the mouse as the user moves it around the viewer.

```
function getMouseCoor() {
    console.println( "("+this.mouseX+","+ this.mouseY+" )" );
}
var ckMouse = app.setInterval("getMouseCoor()", 100);
var timeout = app.setTimeout(
    "app.clearInterval(ckMouse); app.clearTimeOut(timeout)",2000);
```

## mouseY

7.0			
-----	--	--	--

Gets the y-coordinate of the mouse coordinates in default user space in relation to the current page.

*Type: Number*

*Access: R.*

## noautocomplete

7.0			
-----	--	--	--

This property is used to turn off the Auto-Complete feature of Acrobat Forms, *for this document only*.

If set to **true**, no suggestions are made as the user enters data into a field. If this property is set to **false**, auto-complete respects the user preference as set under **File > Preferences > General > Forms**.

Setting this property does not affect the Auto-Complete preferences under **File > Preferences > General > Forms**.

**NOTE:** Initially, this property has a value of **undefined**.

Type: Boolean

Access: R/W.

### Example

For this document, it is desired that auto-complete be turned off. The following script is executed from an open page action, or as a top-level document JavaScript.

```
this.noautocomplete = true;
```

## nocache

7.0			
-----	--	--	--

This property is used to turn off forms data caching *for this document only*.

Setting this property to **true** prevents Acrobat from retaining forms data in an Internet browser. If **nocache** is set to **false**, Acrobat respects the user preference as set under **File > Preferences > General > Forms**.

The value of the **nocache** property does not affect the checkbox item “Keep forms data temporarily available on disk” under **File > Preferences > General > Forms**.

**NOTE:** Before this property is set for the first time, it has a value of **undefined**.

Type: Boolean

Access: R/W.

### Example

For this document, it is desired to turn off caching of form data, so that sensitive data are not left on the local hard drive. The following script is executed from an open page action, or as a top-level document JavaScript.

```
this.nocache = true;
```

## numFields

4.0			
-----	--	--	--

The total number of fields in the document. See also [getNthFieldName](#).

*Type: Integer*

*Access: R.*

### Example 1

```
console.println("There are " + this.numFields + " in this document");
```

### Example 2

The `doc.numFields` property, along with `doc.getNthFieldName`, can be used to loop through all fields in the document. In the script below, we change all button fields so that they have a beveled appearance.

```
for ( var i = 0; i < this.numFields; i++) {
    var fname = this.getNthFieldName(i);
    if ( fname.type = "button" ) f.borderStyle = border.b;
}
```

Other modifications to the buttons of the document can also be made.

## numPages

3.01			
------	--	--	--

The number of pages in the document.

*Type: Integer*

*Access: R.*

### Example 1

```
console.println("There are " + this.numPages + " in this document");
```

### Example 2

Delete the last page from the document. The (0-based) page number of the last page in the document is `this.numPages - 1`.

```
this.deletePages({ nStart: this.numPages - 1 });
```

## numTemplates

ⓧ			
---	--	--	--

The number of templates in the document. See [templates](#), which supersedes this property in later versions.

*Type: Integer*

*Access: R.*

## path

3.01			
------	--	--	--

The device-independent path of the document, for example `/c/Program Files/Adobe/Acrobat 5.0/Help/AcroHelp.pdf`. See Section 3.10.1, “File Specification Strings”, in the [PDF Reference](#) for exact syntax of the path.

*Type:* String                      *Access:* R.

The file name of the document can be acquired by `doc.documentFileName`. See also `doc.URL`.

## outerAppWindowRect

6.0				
-----	--	--	--	--

This property returns the rectangle, an array of screen coordinates, for the Acrobat outer application window. The application window is available as an inner rectangle as well. The outer rectangle includes any title bar, resizing border, or the like, and the inner rectangle does not include these items.

*Type:* Array of Numbers    *Access:* R.

See also [innerAppWindowRect](#), [outerDocWindowRect](#), [outerDocWindowRect](#) and [pageWindowRect](#).

## outerDocWindowRect

6.0				
-----	--	--	--	--

This property returns the rectangle, an array of screen coordinates, for the Acrobat outer document window. The document window is available as an inner rectangle as well. The outer rectangle includes any title bar, resizing border, or the like, and the inner rectangle does not include these items.

These rectangles may differ quite a bit on different platforms. For example, on Windows, the doc window is always inside the app window, while on the Macintosh they are the same.

*Type:* Array of Numbers    *Access:* R.

See also [innerAppWindowRect](#), [outerDocWindowRect](#), [outerAppWindowRect](#) and [pageWindowRect](#).

## pageNum

3.01			
------	--	--	--

Gets or sets a page of the document. When setting the **pageNum** to a specific page, remember that the values are 0-based.

*Type: Integer*

*Access: R/W.*

### Example

This example goes to the first page of the document.

```
this.pageNum = 0;
```

This example advances the document to the next page.

```
this.pageNum++;
```

## pageWindowRect

6.0				
-----	--	--	--	--

This property returns the rectangle, an array of screen coordinates, for the Acrobat page view window. The page view window is the area inside the inner document window in which the actual PDF content is displayed.

*Type: Array of Numbers*

*Access: R.*

See also [innerAppWindowRect](#), [outerDocWindowRect](#), [outerAppWindowRect](#) and [outerDocWindowRect](#).

## permStatusReady

6.0			
-----	--	--	--

Indicates whether the permissions for this document have been resolved. This can return **false** if the document is not available, for example when downloading over a network connection, and permissions are determined based on a signature that covers the entire document. Such documents will be signed with an author signature.

*Type: Boolean*

*Access: R.*

## producer

⊗			
---	--	--	--

The producer of the document (for example, "Acrobat Distiller", "PDFWriter", and so on). See [info](#), which supersedes this property in later versions.

Type: *String*

Access: *R*.

## requiresFullSave

7.0				
-----	--	--	--	--

The property **requiresFullSave** is a boolean which is **true** if the document requires a full save because it is temporary or newly created; otherwise, **requiresFullSave** is false.

Type: *Boolean*

Access: *R*.

### Example

```
var oDoc = app.newDoc();
console.println("It is " + oDoc.requiresFullSave
  + " that this document requires a full save.");
```

## securityHandler

5.0			
-----	--	--	--

The name of the security handler used to encrypt the document. Returns **null** if there is no security handler (for instance, the document is not encrypted).

Type: *String | null*

Access: *R*.

### Example

```
console.println(this.securityHandler != null ?
  "This document is encrypted with " + this.securityHandler
  + " security." : "This document is unencrypted.");
```

This could print out the following if the document was encrypted with the standard security handler.

```
This document is encrypted with Standard security.
```

## selectedAnnots

5.0			<b>A</b>	<b>X</b>
-----	--	--	----------	----------

An array of [Annot Objects](#) corresponding to every markup annotation the user currently has selected.

See also [getAnnot](#) and [getAnnots](#).

Type: Array

Access: R.

### Example

Show all the comments of selected annots in console.

```
var aAnnots = this.selectedAnnots;
for (var i=0; i < aAnnots.length; i++)
    console.println(aAnnots[i].contents);
```

## sounds

5.0			
-----	--	--	--

An array containing all of the named [Sound Objects](#) in the document.

See also [getSound](#), [importSound](#), [deleteSound](#), and the [Sound Object](#).

Type: Array

Access: R.

### Example

```
var s = this.sounds;
for (i = 0; i < s.length; i++)
    console.println("Sound[" + i + "]=" + s[i].name);
```

## spellDictionaryOrder

5.0			
-----	--	--	--

Gets or sets the dictionary array search order for this document. For example, if a user is filling out a Medical Form the form designer may want to specify a Medical dictionary to be searched first before searching the user's preferred order.

The Spelling plug-in searches for words first in this array, and then searches the dictionaries the user has selected on the Spelling Preference panel. The user's preferred order is



available from `spell.dictionaryOrder`. An array of the currently installed dictionaries can be obtained using `spell.dictionaryNames`.

**NOTE:** When setting this property, an exception is thrown if any of the elements in the array is not a valid dictionary name.

Type: Array

Access: R/W.

## spellLanguageOrder

6.0			X	
-----	--	--	---	--

This property can be used to access or specify the language array search order for this document. The Spelling plug-in will search for words first in this array and then in will search the languages the user has selected on the Spelling Preferences panel. The user's preferred order is available from the `spell.languageOrder`. An array of currently installed languages can be obtained using the `spell.languages` property.

Type: Array

Access: R/W.

## subject

X	D		X	
---	---	--	---	--

The document's subject. See [info](#), which supersedes this property in later versions.

**NOTE:** This property is read-only in Adobe Reader.

Type: String

Access: R/W.

## templates

5.0			
-----	--	--	--

An array of all of the [Template Objects](#) in the document. See also [createTemplate](#), [getTemplate](#) and [removeTemplate](#).

Type: Array

Access: R.

### Example

List all templates in the document.

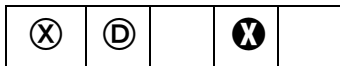
```
var t = this.templates
for ( var i=0; i < t.length; i++)
{
    var state = (t[i].hidden) ? "visible" : "hidden"
```

```

        console.println("Template: \" + t[i].name
            + "\", current state: \" + state);
    }

```

## title



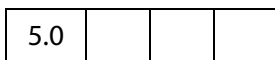
The title of the document. See [info](#), which supersedes this property in later versions.

**NOTE:** This property is read-only in Adobe Reader.

*Type: String*

*Access: R/W.*

## URL



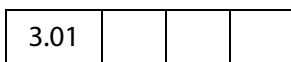
The document's URL. If the document is local, returns a URL with a `file:///` scheme, for a window and Unix OS, and `file://localhost/`, for a Mac OS. This may be different from the [baseUrl](#).

*Type: String*

*Access: R.*

See also `doc.path` and `doc.documentFileName`.

## zoom



Gets or sets the current page zoom level. Allowed values are between 8.33% and 6400%, specified as an percentage number, for example, a zoom value of 100 specifies 100%.

*Type: Number*

*Access: R/W.*

### Example

This example zooms in to twice the current zoom level.

```
this.zoom *= 2;
```

This sets the zoom to 200%.

```
this.zoom = 200;
```

## zoomType

3.01			
------	--	--	--

The current zoom type of the document. The table below lists the valid zoom types.

The convenience **zoomtype** object defines all the valid zoom types and is used to access all zoom types.

Zoom Type	Keyword	Version
NoVary	<code>zoomtype.none</code>	
FitPage	<code>zoomtype.fitP</code>	
FitWidth	<code>zoomtype.fitW</code>	
FitHeight	<code>zoomtype.fitH</code>	
FitVisibleWidth	<code>zoomtype.fitV</code>	
Preferred	<code>zoomtype.pref</code>	
ReflowWidth	<code>zoomtype.refW</code>	6.0

*Type: String*

*Access: R/W.*

### Example

This example sets the zoom type of the document to fit the width.

```
this.zoomType = zoomtype.fitW;
```

## Doc Methods

### addAnnot

5.0	ⓓ		✗	✗
-----	---	--	---	---

Creates an **annot** object having the specified properties. Properties not specified are given their default values for the specified **type** of annotation.

## Parameters

---

<b>objectLiteral</b>	A generic object which specifies the properties of the <b>annot</b> object, such as <b>type</b> , <b>rect</b> , and <b>page</b> , to be created.
----------------------	--

---

## Returns

The new [Annot Object](#).

### Example 1

This example creates a "Square" annotation.

```
var sqannot = this.addAnnot({type: "Square", page: 0});
```

This is a minimal example; **sqannot** will be created as annotation of type "Square" located on the first page (0-based page numbering).

### Example 2

```
var annot = this.addAnnot
({
  page: 0,
  type: "Text",
  author: "A. C. Robat",
  point: [300,400],
  strokeColor: color.yellow,
  contents: "Need a little help with this paragraph.",
  noteIcon: "Help"
});
```

### Example 3

```
var annot = this.addAnnot({
  page: 0,
  type: "Square",
  rect: [0, 0, 100, 100],
  name: "OnMarketShare",
  author: "A. C. Robat",
  contents: "This section needs revision."
});
```

### Example 4

Below is a fancy ink annotation in the shape of a three-leaf rose.

```
var inch = 72, x0 = 2*inch, y0 = 4*inch;
var scaledInch = .5*inch;
var nNodes = 60;
var theta = 2*Math.PI/nNodes;
var points = new Array();
for (var i = 0; i <= nNodes; i++) {
  Theta = i*theta;
  points[i] = [x0 + 2*Math.cos(3*Theta)*Math.cos(Theta)*scaledInch,
  y0 + 2*Math.cos(3*Theta)*Math.sin(Theta)*scaledInch];
}
```

```

}
var annot = this.addAnnot({
  type: "Ink",
  page: 0,
  name: "myRose",
  author: "A. C. Robot",
  contents: "Three leaf rose",
  gestures: [points],
  strokeColor: color.red,
  width: 1
});

```

## addField

5.0	Ⓓ		Ⓕ	
-----	---	--	---	--

Creates a new form field and returns it as a [Field Object](#).

**NOTE:** (Ⓕ, version 6.0): Beginning with version 6.0, `doc.addField` can now be used from within Adobe Reader for documents with “Advanced Form Features”.

### Parameters

<b>cName</b>	The name of the new field to create. This name can use the dot separator syntax to denote a hierarchy (for example, <code>name.last</code> creates a parent node, <code>name</code> , and a child node, <code>last</code> ).
<b>cFieldType</b>	The type of form field to create. Valid types are: text button combobox listbox checkbox radiobutton signature
<b>nPageNum</b>	The 0-based index of the page to which to add the field.
<b>oCoords</b>	An array of four numbers in <i>rotated user space</i> that specifies the size and placement of the form field. These four numbers are the coordinates of the bounding rectangle, in the following order: upper-left x, upper-left y, lower-right x and lower-right y. See also <a href="#">field.rect</a> .  <b>NOTE:</b> If you use the <b>Info</b> panel to obtain the coordinates of the bounding rectangle, you must transform them from <i>info space</i> to <i>rotated user space</i> . To do this, subtract the info space y-coordinate from the on-screen page height.

**Returns**

The newly created [Field Object](#).

**Example**

The following code might be used in a batch sequence to create a navigational icon on every page of a document, for each document in a selected set of documents.

```
var inch = 72;
for (var p = 0; p < this.numPages; p++) {
  // position rectangle (.5 inch, .5 inch)
  var aRect = this.getPageBox( {nPage: p} );
  aRect[0] += .5*inch;           // from upper left hand corner of page.
  aRect[2] = aRect[0]+.5*inch; // Make it .5 inch wide
  aRect[1] -= .5*inch;
  aRect[3] = aRect[1] - 24;     // and 24 points high

  // now construct button field with a right arrow from ZapfDingbats
  var f = this.addField("NextPage", "button", p, aRect );
  f.setAction("MouseUp", "this.pageNum++");
  f.delay = true;
  f.borderStyle = border.s;
  f.highlight = "push";
  f.textSize = 0;                // auto sized
  f.textColor = color.blue;
  f.fillColor = color.ltGray;
  f.textFont = font.ZapfD
  f.buttonSetCaption("\341")     // a right arrow
  f.delay = false;
}
```

See [field.setAction](#) for another example.

**addIcon**

5.0	Ⓓ		
-----	---	--	--

Adds a new named [Icon Generic Object](#) to the document level icon tree, storing it under the specified name.

See also [icons](#), [getIcon](#), [importIcon](#), [removeIcon](#), and the **field** methods [buttonGetIcon](#), [buttonImportIcon](#), and [buttonSetIcon](#).

**Parameters**

<b>cName</b>	The name of the new object
<b>icon</b>	The <a href="#">Icon Generic Object</a> to add.

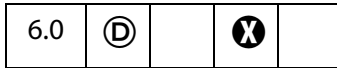
**Returns**

Nothing

**Example**

This example takes an icon already attached to a form button field in the document and assigns a name to it. This name can be used to retrieve the icon object with a [getIcon](#) for use in another button, for example.

```
var f = this.getField("myButton");
this.addIcon("myButtonIcon", f.buttonGetIcon());
```

**addLink**

Adds a new link to the specified page with the specified coordinates, if the user has permission to add links to the document. See also [getLinks](#), [removeLinks](#) and the [Link Object](#).

**Parameters**

<b>nPage</b>	The page on which to add the new link.
<b>oCoords</b>	An array of four numbers in <i>rotated user space</i> that specifies the size and placement of the link. These four numbers are the coordinates of the bounding rectangle, listed in the following order: upper-left x, upper-left y, lower-right x and lower-right y.

**Returns**The newly created [Link Object](#).**Example 1**

Create simple navigational links in the lower left and right corners of each page of the current document. The link in lower left corner goes to the previous page; the one in the lower right corner goes to the next page.

```
var linkWidth = 36, linkHeight = 18;
for ( var i=0; i < this.numPages; i++)
{
    var cropBox = this.getPageBox("Crop", i);
    var linkRect1 = [0,linkHeight,linkWidth,0];
```

```

var offsetLink = cropBox[2] - cropBox[0] - linkWidth;
var linkRect2 = [offsetLink,linkHeight,linkWidth + offsetLink,0]
var lhLink = this.addLink(i, linkRect1);
var rhLink = this.addLink(i, linkRect2);
var nextPage = (i + 1) % this.numPages;
var prevPage = (i - 1) % this.numPages;
var prevPage = (prevPage>=0) ? prevPage : -prevPage;
lhLink.setAction( "this.pageNum = " + prevPage);
lhLink.borderColor = color.red;
lhLink.borderWidth = 1;
rhLink.setAction( "this.pageNum = " + nextPage);
rhLink.borderColor = color.red;
rhLink.borderWidth = 1;
}

```

See the [Link Object](#) for setting the properties and for setting the action of a link.

## Example 2

Search through the document for the word “Acrobat” and create a link around that word.

```

for (var p = 0; p < this.numPages; p++)
{
    var numWords = this.getPageNumWords(p);
    for (var i=0; i<numWords; i++)
    {
        var ckWord = this.getPageNthWord(p, i, true);
        if ( ckWord == "Acrobat")
        {
            var q = this.getPageNthWordQuads(p, i);
            // convert quads in default user space to rotated
            // user space used by Links.
            m = (new Matrix2D).fromRotated(this,p);
            mInv = m.invert()
            r = mInv.transform(q)
            r=r.toString()
            r = r.split(",");
            l = addLink(p, [r[4], r[5], r[2], r[3]]);
            l.borderColor = color.red
            l.borderWidth = 1
            l.setAction("this.getURL('http://www.adobe.com/');");
        }
    }
}

```

The **Matrix2D** object and its methods are defined in the `Annots.js` file.



## addRecipientListCryptFilter

6.0	Ⓓ	Ⓔ	ⓧ	
-----	---	---	---	--

This method adds a crypt filter to this document. The crypt filter is used for encrypting [Data Objects](#).

See also the `cCryptFilter` parameter of the `doc.importDataObject`, `doc.createDataObject` and `doc.setDataObjectContents` methods.

**NOTES:** (Security Ⓔ): Can only be executed during batch, application initialization, menu or console events. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

Not available in the Adobe Reader.

### Parameters

<code>cCryptFilter</code>	The language independent name of the crypt filter. This same name should be used as the value of the <code>cCryptFilter</code> parameter of the <code>Doc Object</code> methods <a href="#">importDataObject</a> , <a href="#">createDataObject</a> and <a href="#">setDataObjectContents</a> .
<code>oGroup</code>	An array of <a href="#">Group Objects</a> that lists the recipients for whom the data is to be encrypted.

### Returns

Nothing

### Example

This script takes the current document open in the viewer, and encrypts and embeds the document into a "ePaper" envelope PDF document. This script was executed in the console, but is perhaps best executed a folder JavaScript as part of larger script for sending PDF docs in a secure way.

```
var Note = "Select the list of people that you want to send this"
    + " document to. Each person must have both an email address"
    + " and a certificate that you can use when creating the"
    + "envelope.";
var oOptions = { bAllowPermGroups: false, cNote: Note,
    bRequireEmail: true };
var oGroups = security.chooseRecipientsDialog( oOptions );
var env = app.openDoc( "/c/temp/ePaperMailEnvelope.pdf" );
env.addRecipientListCryptFilter( "MyFilter", oGroups );
env.importDataObject( "secureMail0", this.path, "MyFilter" );
var envPath = "/c/temp/outMail.pdf";
env.saveAs( envPath );
```

## addScript

6.0	Ⓓ		ⓧ	
-----	---	--	---	--

Sets a document level script for a document. See also [setAction](#), [setPageAction](#), [bookmark.setAction](#), and [field.setAction](#).

**NOTE:** This method will overwrite any script already defined for **cName**.

### Parameters

<b>cName</b>	The name of the script that will be added. If a script with this name already exists, the new script replaces the old one.
<b>cScript</b>	The JavaScript expression that is to be executed when the document is opened.

### Returns

Nothing

### Example

Create a beeping sound every time the document is opened.

```
this.addScript("My Code", "app.beep(0);");
```

See [Example 2](#) following the `doc.disclosed` property for another example.

## addThumbnails

5.0	Ⓓ		ⓧ	
-----	---	--	---	--

Creates thumbnails for the specified pages in the document. See also [removeThumbnails](#).

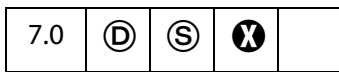
**Parameters**

<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages. If <b>nStart</b> and <b>nEnd</b> are not specified then the range of pages is for all pages in the document. If only <b>nStart</b> is specified then the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages. If <b>nStart</b> and <b>nEnd</b> are not specified then the range of pages is for all pages in the document. If only <b>nEnd</b> is specified then the range of a pages is 0 to <b>nEnd</b> .

**Returns**

Nothing

**addWatermarkFromFile**



Adds a page as a watermark to the specified pages in the document, and places the watermark in an Optional Content Group (OCG). Since converting a file into a PDF document could potentially invoke an external application, this method can only be executed during batch or console events.

See the [OCG Object](#).

**NOTE:** (Security ©): Can only be executed during batch or console events. See also [Privileged versus Non-privileged Context](#).

**Parameters**

<b>cDIPath</b>	The device-independent path of the source file to use for the watermark. If the file at this location is not a PDF file, then Acrobat will attempt to convert the file to a PDF file.
<b>nSourcePage</b>	(optional) The 0-based index of the page in the source file to be used as the watermark. Default is 0.
<b>nStart</b>	(optional) The 0-based index of the first page in the range of pages to which the watermark should be added. If <b>nStart</b> and <b>nEnd</b> are not specified then the range of pages is for all pages in the document. If only <b>nStart</b> is specified then the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) The last page in the range of pages to which the watermark should be added. If <b>nStart</b> and <b>nEnd</b> are not specified then the range of pages is for all pages in the document. If only <b>nEnd</b> is specified then the range of a pages is 0 to <b>nEnd</b> .

---

<b>bOnTop</b>	(optional) A boolean indicating the desired z-ordering of the watermark. A value of <b>true</b> will result in the watermark being added above all other page content. A value of <b>false</b> will result in the watermark being added below all other page content. This parameter is ignored if <b>bFixedPrint</b> is <b>true</b> . Default is <b>true</b> .
<b>bOnScreen</b>	(optional) A boolean to indicate whether or not the watermark should be displayed when viewing the document on screen. Default is <b>true</b> .
<b>bOnPrint</b>	(optional) A boolean to indicate whether or not the watermark should be displayed when printing the document. Default is <b>true</b> .
<b>nHorizAlign</b>	(optional) A number indicating how the watermark should be aligned horizontally. See <a href="#">app.constants.align</a> for possible values. The default is <a href="#">app.constants.align.center</a> .
<b>nVertAlign</b>	(optional) A number indicating how the watermark should be aligned horizontally. See <a href="#">app.constants.align</a> for possible values. The default is <a href="#">app.constants.align.center</a> .
<b>nHorizValue</b>	(optional) Number used to shift the horizontal position of the watermark on the page. If <b>bPercentage</b> is <b>true</b> , then this number represents a percentage of the horizontal page size. If <b>bPercentage</b> is <b>false</b> , then this number represents the number of points to be offset. Default is 0.
<b>nVertValue</b>	(optional) Number used to shift the vertical position of the watermark on the page. If <b>bPercentage</b> is <b>true</b> , then this number represents a percentage of the vertical page size. If <b>bPercentage</b> is <b>false</b> , then this number represents the number of points to be offset. Default is 0.
<b>bPercentage</b>	(optional) A boolean used to indicate whether <b>nHorizValue</b> and <b>nVertValue</b> represent a percentage of the page size or an explicit number of points. Default is <b>false</b> .
<b>nScale</b>	(optional) The scale to be used for the watermark, where 1.0 is 100%. A value of -1 specifies that the watermark should fit to the page while maintaining its proportions. Default is 1.0.
<b>bFixedPrint</b>	(optional) A boolean used to indicate that this watermark should be added as FixedPrint Watermark annotation. This allows watermarks to be printed at a fixed size/position regardless of the size of the page being printed to. If <b>true</b> , then <b>bOnTop</b> is ignored. Default is <b>false</b> .
<b>nRotation</b>	(optional) The number of degrees to rotate the watermark counterclockwise. Default is 0.

---




---

<b>nOpacity</b>	(optional) The opacity to be used for the watermark, where 0 is transparent, and 1.0 is opaque. Default is 1.0.
-----------------	---

---

**Returns**

Nothing

**Example 1**

Adds the first page of watermark.pdf as a watermark to the center all pages of the current document.

```
this.addWatermarkFromFile("/C/temp/watermark.pdf");
```

**Example 2**

Adds the second page of watermark.pdf as a watermark to the first 10 pages of the current document. The watermark will be rotated counter-clockwise 45 degrees, and positioned one inch down and two inches over from the upper-left corner of the page.

```
this.addWatermarkFromFile({
    cDIPath: "/C/temp/watermark.pdf",
    nSourcePage: 4, nEnd: 9,
    nHorizAlign: app.constants.align.left,
    nVertAlign: app.constants.align.top,
    nHorizValue: 144, nVertValue: -72,
    nRotation: 45
});
```

**addWatermarkFromText**



Adds the given text as a watermark to the specified pages in the document, and places the watermark in an Optional Content Group (OCG).

See the [OCG Object](#).

**Parameters**

---

<b>cText</b>	The text to use as the watermark. Multiline text is allowed. A newline can be specified with the characters "\r".
<b>nTextAlign</b>	(optional) The text alignment to use for <b>cText</b> within the watermark. See <a href="#">app.constants.align</a> for possible values. This parameter has no effect if <b>cText</b> is only one line.
<b>cFont</b>	(optional) The font to be used for this watermark. Valid fonts are defined as properties of the <a href="#">font Object</a> , as listed in field.textFont. An arbitrary font can be used by passing a string that represents the PostScript name of the font. Default is <b>font.Helv</b> .

---

---

<b>nFontSize</b>	(optional) The point size of the font to use for the watermark. Default is 24.
<b>aColor</b>	(optional) The color to use for the watermark. See <a href="#">Color Arrays</a> for information on defining color arrays. Default is <code>color.black</code> .
<b>nStart</b>	(optional) The 0-based index of the first page in the range of pages to which the watermark should be added. If <b>nStart</b> and <b>nEnd</b> are not specified then the range of pages is for all pages in the document. If only <b>nStart</b> is specified then the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) optional) The last page in the range of pages to which the watermark should be added. If <b>nStart</b> and <b>nEnd</b> are not specified then the range of pages is for all pages in the document. If only <b>nEnd</b> is specified then the range of a pages is 0 to <b>nEnd</b> .
<b>bOnTop</b>	(optional) A boolean indicating the desired z-ordering of the watermark. A value of <code>true</code> will result in the watermark being added above all other page content. A value of <code>false</code> will result in the watermark being added below all other page content. This parameter is ignored if <b>bFixedPrint</b> is <code>true</code> . Default is <code>true</code> .
<b>bOnScreen</b>	(optional) A boolean to indicate whether or not the watermark should be displayed when viewing the document on screen.
<b>bOnPrint</b>	(optional) A boolean to indicate whether or not the watermark should be displayed when printing the document.
<b>nHorizAlign</b>	(optional) A number indicating how the watermark should be aligned horizontally. See <a href="#">app.constants.align</a> for possible values. The default is <code>app.constants.align.center</code> .
<b>nVertAlign</b>	(optional) A number indicating how the watermark should be aligned vertically. See <a href="#">app.constants.align</a> for possible values. The default is <code>app.constants.align.center</code> .
<b>nHorizValue</b>	(optional) Number used to shift the horizontal position of the watermark on the page. If <b>bPercentage</b> is <code>true</code> , then this number represents a percentage of the horizontal page size. If <b>bPercentage</b> is <code>false</code> , then this number represents the number of points to be offset. Default is 0.
<b>nVertValue</b>	(optional) Number used to shift the vertical position of the watermark on the page. If <b>bPercentage</b> is <code>true</code> , then this number represents a percentage of the vertical page size. If <b>bPercentage</b> is <code>false</code> , then this number represents the number of points to be offset. Default is 0.

---

<b>bPercentage</b>	(optional) A boolean used to indicate whether <b>nHorizValue</b> and <b>nVertValue</b> represent a percentage of the page size or an explicit number of points. Default is <b>false</b> .
<b>nScale</b>	(optional) The scale to be used for the watermark, where 1.0 is 100%. A value of -1 specifies that the watermark should fit to the page while maintaining its proportions. Default is 1.0.
<b>bFixedPrint</b>	(optional) A boolean used to indicate that this watermark should be added as FixedPrint Watermark annotation. This allows watermarks to be printed at a fixed size/position regardless of the size of the page being printed to. If <b>true</b> , then <b>bOnTop</b> is ignored. Default is <b>false</b> .
<b>nRotation</b>	(optional) The number of degrees to rotate the watermark counterclockwise. Default is 0.
<b>nOpacity</b>	(optional) The opacity to be used for the watermark, where 0 is transparent, and 1.0 is opaque. Default is 1.0.

**Returns**

Nothing

**Example 1**

Adds "Confidential" as a watermark to the center of all pages of the current document.

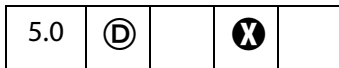
```
this.addWatermarkFromText("Confidential", 0, font.Helv, 24, color.red);
```

**Example 2**

Adds a multiline watermark to each page of the current document one inch down and one inch over from the upper-right corner.

```
this.addWatermarkFromText ({
  cText: "Confidential Document\rA. C. Robot",
  nTextAlign: app.constants.align.right,
  nHorizAlign: app.constants.align.right,
  nVertAlign: app.constants.align.top,
  nHorizValue: -72, nVertValue: -72
});
```

**addWeblinks**



Scans the specified pages looking for instances of text with a `http:` scheme, and converts them into links with URL actions.

See also [removeWeblinks](#).

**Parameters**

<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages. If <b>nStart</b> and <b>nEnd</b> are not specified then the range of pages is for all pages in the document. If only <b>nStart</b> is specified then the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages. If <b>nStart</b> and <b>nEnd</b> are not specified then the range of pages is for all pages in the document. If only <b>nEnd</b> is specified then the range of a page is 0 to <b>nEnd</b> .

**Returns**

The number of web links added to the document.

**Example**

Search the entire document and convert all content that appear to be a web address into a web link. Report back the number of links created.

```
var numWeblinks = this.addWeblinks();
console.println("There were " + numWeblinks +
    " instances of text that looked like a web address,"
    +" and converted as such.");
```

**bringToFront**

5.0			
-----	--	--	--

Brings the document open in the Viewer to the front, if it is not already there.

**Parameters**

None

**Returns**

Nothing

**Example**

This example searches among the documents open in the Viewer for the document with a title of "Annual Report" and brings it to the front.

```
var d = app.activeDocs; // lists only disclosed documents
for (var i = 0; i < d.length; i++)
    if (d[i].info.Title == "Annual Report") d[i].bringToFront();
```



## calculateNow

3.01			
------	--	--	--

Forces computation of all calculation fields in the current document.

### Parameters

None

### Returns

Nothing

### Example

When a form contains a lot of calculations, there can be a significant delay after the user inputs data into a field, even if that field is not a calculation field. One strategy is to turn off calculations at some point and turn them back on at a later point.

```
// turn off calculations
this.calculate = false;
.....
// turn on calculations
this.calculate = true;
// Unless the user committed data after this.calculate is set to true,
// automatic calculation does not occur. Calculation can be forced to
// occur by using...
this.calculateNow();
```

## closeDoc

5.0			Ⓢ	
-----	--	--	---	--

Closes the document.

**NOTE:** (Document Save Rights **Ⓢ**): For Adobe Reader 5.1 or later, the method is always allowed. However, if the document was changed and no *Document Save Rights* are available, the document is closed without any warnings and changes are lost. If *Document Save Rights* are available, the user gets the option of saving the changed file. It is important to use this method carefully as it is an abrupt change in the document state that can affect any JS executing after the close. Triggering this method from a Page event or Document event could cause the application to behave strangely.

**NOTE:** In versions of Acrobat prior to 7.0, a document that closes itself by executing **this.closeDoc** terminates any script that follows it. In Acrobat 7.0, the script is allowed to continue and to terminate naturally; however, if the **doc** object of the closed document is referenced, an exception will be thrown.

**Parameters**


---

<b>bNoSave</b>	(optional) Whether to close the document without saving. If <b>false</b> (the default), the user is prompted to save the document if it has been modified. If <b>true</b> , the document is closed without prompting the user and without saving, even if the document has been modified. Because this can cause data loss without user approval, use this feature judiciously.
----------------	---

---

**Returns**

Nothing

**Example 1**

From the console, close all open documents.

```
var d = app.activeDocs;
for( var i in d ) d[i].closeDoc();
```

The following code can be executed as a mouse up action from a document open in the viewer. It closes all *disclosed* documents that are open in the viewer. The code is designed to close the active document last so that the execution of the code will not be abruptly terminated.

```
var d = app.activeDocs;
for( var i in d )
    if( d[i] != this ) d[i].closeDoc();
if ( this.disclosed ) this.closeDoc();
```

**Example 2**

Create a series of three test files and save them to a directory. This code needs to be executed in the console, because [saveAs](#) has a security restriction.

```
var myDoc = app.newDoc();
for (var i=0; i < 3; i++) {
    myDoc.info.Title = "Test File " + i;
    myDoc.saveAs("/c/temp/test"+i+".pdf");
}
myDoc.closeDoc(true);
```

See [saveAs](#) for an another example of `closeDoc`.

**createDataObject**

5.0	Ⓓ		Ⓕ	
-----	---	--	---	--

Creates a [Data Object](#).

Data objects can be constructed *ad hoc*. This is useful if the data is being created in JavaScript from sources other than an external file (for example, ADBC database calls).

Related objects, properties and methods are the [Data Object](#), `doc.dataObjects`, `doc.getDataObject`, `doc.openDataObject`, `doc.importDataObject`, `doc.removeDataObject`, `doc.getDataObjectContents` and `doc.setDataObjectContents`.

### Parameters

<b>cName</b>	The name to associate with the data object.
<b>cValue</b>	A string containing the data to be embedded.
<b>cMIMETYPE</b>	(optional) The MIME type of the data. Default is "text/plain".
<b>cCryptFilter</b>	(optional, version 6.0) The language independent name of a crypt filter to use when encrypting this data object. This crypt filter must have previously been added to the document's list of crypt filters, using the <a href="#">Doc Object addRecipientListCryptFilter</a> method, otherwise an exception will be thrown. The predefined <b>"Identity"</b> crypt filter can be used if it is desired that this data object not be encrypted in a file that is otherwise encrypted by the <a href="#">Doc Object encryptForRecipients</a> method.

### Returns

Nothing

### Example

```
this.createDataObject("MyData.txt", "This is some data.");
```

See also the example that follows [addRecipientListCryptFilter](#).

## createTemplate

5.0	Ⓓ	Ⓔ		
-----	---	---	--	--

Creates a visible template from the specified page. See also `doc.templates`, the `doc.getTemplate`, `doc.removeTemplate` methods, and the [Template Object](#).

**NOTES:** (Security Ⓔ): This method can only be executed during batch, console, or menu events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

(Version 7.0) In Adobe Reader 5.1 and later, this method was allowed with Advanced Form Features rights (Ⓕ), beginning with this version of Adobe Reader, this method is not allowed and will throw a **NotAllowedError** exception.

**Parameters**

<b>cName</b>	The name to be associated with this page.
<b>nPage</b>	(optional) The 0-based index of the page to operate on. Default is 0, the first page in the document.

**Returns**

The newly created [Template Object](#).

**Example**

Convert all pages beginning with page 2 (base 0) to hidden templates. We have to be a little careful, as the templates are hidden, **this.numPages** is updated to reflect that change in the number of (visible) pages. Notice that in the loop below, only page 2 is made a template then hidden; the next page will become the new page 2.

```
numNewTemplates = this.numPages - 2;
for ( var i = 0; i < numNewTemplates; i++)
{
    var t = this.createTemplate({cName:"myTemplate"+i, nPage:2 });
    t.hidden = true;
}
```

**deletePages**

5.0	Ⓓ		Ⓕ	
-----	---	--	---	--

Deletes pages from the document. If neither page of the range is specified, the first page (page 0) is deleted. See also [insertPages](#), [extractPages](#) and [replacePages](#).

**NOTE:** You cannot delete all pages in a document: there must be at least one page remaining.

**NOTE:** (Ⓕ, version 6.0): Beginning with version 6.0, **doc.deletePages** can now delete *spawned* pages from within Adobe Reader for documents with “Advanced Form Features”.

**Parameters**

<b>nStart</b>	(optional) The 0-based index of the first page in the range of pages to be deleted. Default is 0, the first page in the document.
<b>nEnd</b>	(optional) The last page in the range of pages to be deleted. If <b>nEnd</b> is not specified then only the page specified by <b>nStart</b> is deleted.

**Returns**

Nothing

**Example**

Delete pages 1 through 3 (base 0), inclusive

```
this.deletePages({nStart: 1, nEnd: 3});
```

**deleteSound**

5.0	ⓓ		ⓧ	
-----	---	--	---	--

Deletes the [Sound Object](#) with the specified name from the document.

See also [sounds](#), [getSound](#), [importSound](#), and the [Sound Object](#).

**Parameters**


---

<b>cName</b>	The name of the sound object to delete.
--------------	---

---

**Returns**

Nothing

**Example**

```
this.deleteSound("Moo");
```

**embedDocAsDataObject**

7.0			Ⓢ	
-----	--	--	---	--

Embeds the specified document as a Data Object in the document.

**NOTE:** (Document Save Rights **Ⓢ**): For Acrobat 7.0 Reader and later, this method is commonly allowed, but document Save rights on the document to be embedded are required in case the document to be embedded has changed and this changed document is to be embedded.

**Parameters**


---

<b>cName</b>	The name to associate with the data object.
<b>oDoc</b>	The document to embed as a data object.

---

---

<b>cCryptFilter</b>	(optional) The language independent name of a crypt filter to use when encrypting this data object. This crypt filter must have previously been added to the document's list of crypt filters, using the <a href="#">Doc.addRecipientListCryptFilter</a> method, otherwise an exception will be thrown. The predefined "Identity" crypt filter can be used if it is desired that this data object not be encrypted in a file that is otherwise encrypted by the <a href="#">Doc.encryptForRecipients</a> method.
<b>bUI</b>	(optional) If <b>true</b> , an alert may be shown if <b>oDoc</b> requires saving and permissions do not allow it to be saved. Default value is <b>false</b> .

---

**Returns**




Nothing

**Example**


In this example an envelope file has been previously authored which includes a 'myFilter' crypt filter and the envelope file has been included in 'this' document.

```
var authorEmail = "johndoe@acme.com";
var envelopeDoc = this.openDataObject( "envelope" );
envelopeDoc.embedDocAsDataObject( "attachment", this, "myFilter" );
envelopeDoc.title.Author = authorEmail;
envelopeDoc.mailDoc( {
    cTo: "support@mycompany.com",
    cSubject: "Application from " + authorEmail
});
```

**encryptForRecipients**

6.0				
-----	---	---	---	--

Encrypts the document for the specified lists of recipients, using the public-key certificates of each recipient. Encryption does not take place until the document is saved. Recipients can be placed into groups, and each group can have its own unique permission settings. This method throws an exception if it is unsuccessful.

**NOTES:** (Security Privileged versus Non-privileged Context.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

It is also available in the Adobe Reader

See also the [security.chooseRecipientsDialog](#) method, the [Data Object](#) and the [createDataObject](#).

**Parameters**

<b>oGroups</b>	An array of generic <a href="#">Group Objects</a> that list the recipients for which the document is to be encrypted.
<b>bMetaData</b>	(optional) Whether document meta data should be encrypted. The default value is <b>true</b> . Setting this value to <b>false</b> will produce a document that can only be viewed in Acrobat 6.0 or later.
<b>bUI</b>	(optional) When <b>true</b> , the handler displays the user interface, in which the user can select the recipients for whom to encrypt the document. The default value is <b>false</b> .

**Returns**

**true**, if successful, otherwise an exception is thrown.

**Group Object**

A generic JS object that allows a set of permissions to be attached to a list of recipients for which a document or data is to be encrypted. This object is passed to `doc.encryptForRecipients`, and returned by `security.chooseRecipientsDialog`. It contains the following properties:

Property	Description
<b>permissions</b>	A <a href="#">Group Object</a> with the permissions for the group.
<b>userEntities</b>	An array of <a href="#">UserEntity Generic Objects</a> , the users to whom the permissions apply.

**Permissions Object**

A generic JS object that contains a set of permissions, used in a [Group Object](#). It contains the following properties. The default value for all properties is **false**.

Property	Type	Access	Description
<b>allowAll</b>	Boolean	R/W	Whether full, unrestricted access is permitted. If <b>true</b> , overrides all other properties.
<b>allowAccessibility</b>	Boolean	R/W	Whether content access for the visually impaired is permitted. When <b>true</b> , allows content to be extracted for use by applications that, for example, read text aloud.

Property	Type	Access	Description
<b>allowContentExtraction</b>	Boolean	R/W	Whether content copying and extraction is permitted.
<b>allowChanges</b>	String	R/W	What changes are allowed to be made to the document. Values are: none documentAssembly fillAndSign editNotesFillAndSign all
<b>allowPrinting</b>	String	R/W	What the allowed printing security level is for the document. Values are: none lowQuality highQuality

### Example

Encrypt all strings and streams in the document. This will produce a file that can be opened with Acrobat 5.0 and later:

```
var sh = security.getHandler( "Adobe.PPKMS" );
var dir = sh.directories[0];
var dc = dir.connect();

dc.setOutputFields({oFields:["certificates"]});
var importantUsers = dc.search({oParams:{lastName:"Smith"}});
var otherUsers = dc.search({oParams: {lastName:"jones" }});

this.encryptForRecipients({
  oGroups :
  [
    {oUserEntities:importantUsers,oPermissions:{allowAll:true }},
    {oUserEntities:
      otherUsers,
      oPermissions:{allowPrinting:"highQuality"}
    }
  ],
  bMetaData : true
});
```



## encryptUsingPolicy

7.0		Ⓢ	ⓧ	
-----	--	---	---	--

Encrypt the document using the specified policy object and handler. This method may require user interaction and may result in a new Security Policy being created.

**NOTES:** (Security Ⓢ) This method can be executed only during batch, console or application initialization events. See also [Privileged versus Non-privileged Context](#).  
Not available in Reader.

### Parameters

<b>oPolicy</b>	<p>The policy object to use when encrypting the document. This may be a <a href="#">SecurityPolicy Object</a> returned from <a href="#">chooseSecurityPolicy</a> or <a href="#">getSecurityPolicies</a>.</p> <p>This may also be a generic object with the <b>policyId</b> property defined. If a predefined policy id is passed, the associated policy will be retrieved and used. If the policy id passed is unknown, an error will be returned.</p> <p>There is a predefined policy id which has a special behavior. If <b>policyId</b> is set to "adobe_secure_for_recipients" then a new policy will be created by the Adobe Policy Server.</p> <p><b>NOTE:</b> If this special policy id is used and <b>oGroups</b> is <b>null</b>, an error will be returned.</p>
<b>oGroups</b>	<p>(optional) This is an array of <a href="#">Group Objects</a> that the handler should use when applying the policy handler when applying the policy. The exact behavior will depend on the policy used and the handler involved. The Group object may have embedded permission information. Whether or not that information is used, depends on the policy and associated security handler.</p> <p>Default value is <b>null</b>.</p>
<b>oHandler</b>	<p>(optional) The <a href="#">SecurityHandler Object</a> to be used for encryption. This will result in failure if this handler does not match the handler name specified in the <b>oPolicy</b> object. If not specified, the default object associated with this handler will be used.</p> <p>See the paragraph below, <a href="#">On the use of oHandler</a>.</p>
<b>bUI</b>	<p>(optional) If <b>true</b>, user interface may be displayed (e.g. for authentication). If <b>false</b>, user interface will not be displayed. If user interaction is required but not allowed, an error is returned.</p> <p>Default value is <b>false</b>.</p>

**Returns**

The value returned is a [SecurityPolicyResults Generic Object](#).

**On the use of oHandler**

If you are using the APS security handler, you could create a new SecurityHandler ahead of time, authenticate to a server not configured in Acrobat via the `login ()` call, and then pass that SecurityHandler in `oHandler`. This would allow you to use policies which are not defined on the server Acrobat is configured to use.

If you are using the PPKLite security handler, you could create a new SecurityHandler ahead of time, open a digital id file not configured in Acrobat via the `login ()` call, and then pass that SecurityHandler in `oHandler`. This would allow you to use certificates contained in the digital id file but not in Acrobat.

**SecurityPolicyResults Generic Object**

The SecurityPolicyResults object has the following properties:

Property	Type	Description
<b>errorCode</b>	Integer	This will contain the error code returned from the handler implementing the policy. There are three possible errors: 0 = Success. <b>errorText</b> is not defined, <b>unknownRecipients</b> may be defined, <b>policyApplied</b> is defined. 1 = Failure. <b>errorText</b> is defined. <b>unknownRecipients</b> may be defined. <b>policyApplied</b> is not defined. 2 = Abort, the user aborted the process. <b>errorText</b> is not defined. <b>unknownRecipients</b> is not defined. <b>policyApplied</b> is not defined.
<b>errorText</b>	String	If defined, this will contain the localized error description. See <b>errorCode</b> for when this is defined.
<b>policyApplied</b>	Object	If defined, this will contain the <a href="#">SecurityPolicy Object</a> actually applied. If the policy passed in was "adobe_secure_for_recipients", a new policy was created by the call and the corresponding policy object will be returned here. See <b>errorCode</b> for when this is defined.

Property	Type	Description
<b>unknownRecipients</b>	Recipients Object	If defined, this holds recipients passed in which could not be used when applying the policy. See <b>errorCode</b> for when this is defined.

**Example 1**

In this example a newly created document is encrypted using a chosen policy.

```
var doc = app.newDoc();
var policy = security.chooseSecurityPolicy();
var results = doc.encryptUsingPolicy( { oPolicyId: policy } );
console.println("The policy applied was: "
    + results.policyApplied.name);
```

**Example 2**

In this example a newly created document is encrypted using a template policy.

```
var doc = app.newDoc();
var groups = [ { userEntities: [{email:"jdoe@mycorp.com"},
    {email:"bsmith@mycorp.com"} ] }
];
var policy = { policyId: "adobe_secure_for_recipients" };
var results = doc.encryptUsingPolicy({
    oPolicy: policy,
    oGroups: groups,
    bUI: true
});
console.println("The policy applied was: "
    + results.policyApplied.name);
```

**exportAsText**

6.0		Ⓒ	Ⓕ	
-----	--	---	---	--


Exports form fields as a tab-delimited text file to a local hard disk. The text file that is created follows the conventions specified by Microsoft Excel. In particular, **exportAsText** correctly handles quotes and multiline text fields.

This method writes two lines to the text file, the first line a tab-delimited list of the names of the fields specified by **aFields**, the second line is a tab-delimited list of the values of the fields.

**NOTE:** (SecurityⒸ): If the **cPath** parameter is specified, this method can only be executed during batch, console or menu events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) includes a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Parameters**

<b>bNoPassword</b>	(optional) If <b>true</b> (the default), do not include text fields that have the "password" flag set in the exported text file.
<b>aFields</b>	(optional) The array of field names to submit or a string containing a single field name. <ul style="list-style-type: none"> <li>● If specified, only these fields are exported, except those excluded by <b>bNoPassword</b>.</li> <li>● If <b>aFields</b> is an empty array, no fields are exported.</li> <li>● If this parameter is omitted or is <b>null</b>, all fields in the form are exported, except those excluded by <b>bNoPassword</b>.</li> </ul>
<b>cPath</b>	(optional) A string specifying the device-independent pathname for the file. (See Section 3.10.1 of the <a href="#">PDF Reference</a> for a description of the device-independent pathname format.) The pathname may be relative to the location of the current document. If the parameter is omitted a dialog is shown to let the user select the file. <p><b>NOTE:</b> (Security Safe Path and have a <code>.txt</code> extension. This method will throw a <b>NotAllowedError</b> (see the <a href="#">Error Objects</a>) exception if these security conditions are not met, and the method will fail.</p>

**Returns**

Nothing



**Example**

To export all fields to a tab-delimited file, execute the following script in the console:


```
this.exportAsText ( );
```

To create a tab-delimited file with more than just one data line, see the [Example](#) on [page 264](#).

**exportAsFDF**

4.0				
-----	--	---	---	--

Exports form fields as a FDF file to the local hard drive.

**NOTES:** (Security Privileged versus Non-

[privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

## Parameters

<b>bAllFields</b>	(optional) If <b>true</b> , all fields are exported, including those that have no value. If <b>false</b> (the default), excludes those fields that currently have no value.
<b>bNoPassword</b>	(optional) If <b>true</b> (the default), do not include text fields that have the "password" flag set in the exported FDF.
<b>aFields</b>	(optional) The array of field names to submit or a string containing a single field name. <ul style="list-style-type: none"> <li>● If specified, only these fields are exported, except those excluded by <b>bNoPassword</b></li> <li>● If <b>aFields</b> is an empty array, no fields are exported. The FDF might still contain data, depending on the <b>bAnnotations</b> parameter.</li> <li>● If this parameter is omitted or is <b>null</b>, all fields in the form are exported, except those excluded by <b>bNoPassword</b></li> </ul> Specify non-terminal field names to export an entire subtree of fields; see the example below.
<b>bFlags</b>	(optional) If <b>true</b> , field flags are included in the exported FDF. The default is <b>false</b>
<b>cPath</b>	(optional) A string specifying the device-independent pathname for the file. (See Section 3.10.1 of the <a href="#">PDF Reference</a> for a description of the device-independent pathname format.) The pathname may be relative to the location of the current document. If the parameter is omitted a dialog is shown to let the user select the file. <p><b>NOTE:</b> (Security<sup>Ⓢ</sup>): The parameter <b>cPath</b> is required to have a <a href="#">Safe Path</a> and have a <code>.fdf</code> extension. This method will throw a <b>NotAllowedError</b> (see the <a href="#">Error Objects</a>) exception if these security conditions are not met, and the method will fail.</p>
<b>bAnnotations</b>	(optional, version 6.0) If <b>true</b> , annotations are included in the exported FDF. The default is <b>false</b>

## Returns

Nothing

**Example 1**

Export the entire form (including empty fields) with flags.

```
this.exportAsFDF(true, true, null, true);
```

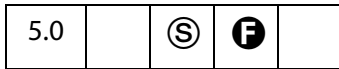
**Example 2**

Export the *name* subtree with no flags.

```
this.exportAsFDF(false, true, "name");
```

The example above illustrates a shortcut to exporting a whole subtree. Passing "name" as part of the **aFields** parameter, exports "**name.title**", "**name.first**", "**name.middle**" and "**name.last**", and so on.

**exportAsXFDF**



Exports form fields an XFDF file to the local hard drive.

XFDF is an XML representation of Acrobat form data. See the document entitled "Forms System Implementation Notes" for an overview of XFDF, available as [Adobe Web Documentation](#). The *XML Form Data Format Specification*, XFDF, can be found at <http://partners.adobe.com/asn/tech/pdf/xmlformspec.jsp>.

There is an import version of this same method, [importAnXFDF](#).

**NOTES:** (SecurityⒸ): If the **cPath** parameter is specified, then this method can only be executed during batch, console or menu events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Parameters**

<b>bAllFields</b>	(optional) If <b>true</b> , all fields are exported, including those that have no value. If <b>false</b> (the default), excludes those fields that currently have no value.
<b>bNoPassword</b>	(optional) If <b>true</b> (the default), do not include text fields that have the "password" flag set in the exported XFDF.

---

<b>aFields</b>	<p>(optional) The array of field names to submit or a string containing a single field name.</p> <ul style="list-style-type: none"> <li>● If specified, only these fields are exported, except those excluded by <b>bNoPassword</b></li> <li>● If <b>aFields</b> is an empty array, no fields are exported. The XPDF might still contain data, depending on the <b>bAnnotations</b> parameter.</li> <li>● If this parameter is omitted or is <b>null</b>, all fields in the form are exported, except those excluded by <b>bNoPassword</b></li> </ul> <p>Specify non-terminal field names to export an entire subtree of fields; see the example below.</p>
<b>cPath</b>	<p>(optional) A string specifying the device-independent pathname for the file. (See Section 3.10.1 of the <a href="#">PDF Reference</a> for a description of the device-independent pathname format.) The pathname may be relative to the location of the current document. If the parameter is omitted a dialog is shown to let the user select the file.</p> <p><b>NOTE:</b> (Security<sup>Ⓢ</sup>): The parameter <b>cPath</b> is required to have a <a href="#">Safe Path</a> and have a <code>.xpdf</code> extension. This method will throw a <b>NotAllowedError</b> (see the <a href="#">Error Objects</a>) exception if these security conditions are not met, and the method will fail.</p>
<b>bAnnotations</b>	<p>(optional, version 6.0) If <b>true</b>, annotations are included in the exported XPDF. The default is <b>false</b></p>

---

**Returns**

Nothing

**exportDataObject**

5.0		Ⓢ		
-----	--	---	--	--

This method extracts the specified data object to an external file.

Related objects, properties and methods are the [Data Object](#), `doc.dataObjects`, `doc.openDataObject`, `doc.createDataObject`, `doc.removeDataObject`, `doc.importDataObject`, `doc.getDataObjectContents`, and `doc.setDataObjectContents`.

**NOTES:** (Security<sup>Ⓢ</sup>): Beginning with Acrobat 6.0, if the parameter **cDIPath** is non-NULL a **NotAllowedError** (see the [Error Objects](#)) exception will be thrown and the method will fail.

If **cDIPath** is not passed to this method, a file selection dialog will open to allow the user to select a save path for the embedded data object.

**Parameters**

<b>cName</b>	The name of the data object to extract.
<b>cDIPath</b>	(optional) A device-independent path to which to extract the data object. This path may be absolute or relative to the current document. If not specified, the user is prompted to specify a save location. See “File Specification Strings” in the <a href="#">PDF Reference Manual</a> for the exact syntax of the path.  <b>NOTE:</b> (version 6.0) The use of this parameter is no longer supported and should not be used. See the security notes above.
<b>bAllowAuth</b>	(optional, version 6.0) If <b>true</b> , a dialog is used to obtain user authorization. Authorization may be required if the data object was encrypted using <a href="#">Doc.encryptForRecipients</a> . Authorization dialogs are allowed if <b>bAllowAuth</b> is <b>true</b> . The default value is <b>false</b> .
<b>nLaunch</b>	(optional, version 6.0) <b>nLaunch</b> controls whether the file is launched, or opened, after it is saved. Launching may involve opening an external application if the file is not a PDF file. The values of <b>nLaunch</b> are <ul style="list-style-type: none"> <li>● If the value is 0, the file will not be launched after it is saved.</li> <li>● If the value is 1, the file will be saved and then launched. Launching will prompt the user with a security alert warning if the file is not a PDF file. The user will be prompted for a save path.</li> <li>● If the value is 2, the file will be saved and then launched. Launching will prompt the user with a security alert warning if the file is not a PDF file. A temporary path is used, and the user will not be prompted for a save path. The temporary file that is created will be deleted by Acrobat upon application shutdown.</li> </ul> The default value is 0.

**Returns**

Nothing

**Example 1**

Prompt the user for a file and location to extract to.

```
this.exportDataObject("MyData");
```

**Example 2 (Version 6.0)**

Extract PDF document and launch it in the viewer.

```
this.exportDataObject({ cName: "MyPDF.pdf", nLaunch: 2 });
```



**Example 3**

When a file attachment is imported using `Doc.importDataObject`, the value of its `Data.name` is assigned via the parameter `cName` of that method; however, when a file is attached using the UI, the `name` of the file attachment is automatically assigned. The first attachment is assigned a `name` of "Untitled Object"; the second, a `name` of "Untitled Object 2"; the third, a `name` of "Untitled Object 3"; and so on.

To export a file attached through the UI, the `name` of the attachment needs to be found. For the code that follows, the last file attached by the UI, if any, is exported.

```
var d = this.dataObjects;
if ( d == null ) console.println("No file attachments");
else {
  for ( var i = d.length - 1; i>=0; i--)
    if ( d[i].name.indexOf("Untitled Object") != -1 ) break;
  if ( i != -1 ) this.exportDataObject(d[i].name);
  else console.println("No attachment was embedded by UI");
}
```

**exportXFADData**

6.0		Ⓒ	Ⓕ	
-----	--	---	---	--

Exports an XFA data file to the local hard drive.

Form Rights (Ⓕ): When exporting XFA data from the Adobe Reader, the document must have export form rights.

**NOTES:** (Security Ⓒ): If the `cPath` parameter is specified, then this method can only be executed during batch, console or menu events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.


**Parameters**


---

<b>cPath</b>	<p>(optional) A device-independent pathname for the file. The pathname may be relative to the document. See <i>File Specification Strings</i> in the <a href="#">PDF Reference Manual</a> for the exact syntax of the path. If this parameter is omitted, a dialog is shown to let the user select the file.</p> <p><b>NOTE:</b> (Security Ⓒ): The parameter <code>cPath</code> is required to have a <a href="#">Safe Path</a>. Additionally, the file name must have a <code>.xdp</code> extension, if <code>bXDP</code> is <code>true</code>, or a <code>.xml</code> extension, if <code>bXDP</code> is <code>false</code>. This method will throw a <code>NotAllowedError</code> (see the <a href="#">Error Objects</a>) exception if these security conditions are not met, and the method will fail.</p>
--------------	--

---

---

<b>bXDP</b>	(optional) If <b>true</b> (the default), the method exports in the XDP format. Otherwise, it exports in the plain XML data format.
<b>aPackets</b>	<p>(optional) An array of strings specifying which packets to include in the XDP export. This parameter is only applicable if <b>bXDP</b> is <b>true</b>.</p> <p>Possible strings are:</p> <ul style="list-style-type: none"> <li>template</li> <li>datasets</li> <li>stylesheet</li> <li>xfdf</li> <li>sourceSet</li> <li>pdf</li> <li>config</li> <li>*</li> </ul> <p><b>pdf</b> means that the PDF should be embedded. If <b>pdf</b> is not specified, only a link to the PDFs included in the XDP.</p> <p><b>xfdf</b> means include annotations in the XDP (since that packet uses XFDF format).</p> <p><b>*</b> means that all packets should be included in the XDP. The default for <b>pdf</b> is to include it as a <i>reference</i>. To embed the PDF file in the XDP, <i>explicitly</i> specify <b>pdf</b> as one of the packets.</p> <p><b>NOTE:</b> (Save rights required .</p>

---

**Returns**

Nothing

**Example**

In the first example, all packets are included; the PDF document is referenced, not embedded, however.

```
this.exportXFADData({
  cPath: "/c/temp/myData.xdp",
  bXDP: true,
  aPackets: ["*"]
})
```

In this example, all packets are included, with the PDF document embedded in the XDP.

```
this.exportXFADData({
  cPath: "/c/temp/myData.xdp",
  bXDP: true,
  aPackets: ["*", "pdf"]
})
```

## extractPages

5.0	ⓓ	Ⓢ	ⓧ	
-----	---	---	---	--

Creates a new document consisting of pages extracted from the current document. If a page range is not specified, the method extracts all pages in the document.

See also [deletePages](#), [insertPages](#), and [replacePages](#).

**NOTES:** (SecurityⓈ) If the **cPath** parameter is specified, then this method can only be executed during batch, console or menu events, or through an external call (for example, OLE). See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

### Parameters

<b>nStart</b>	(optional) A 0-based index that defines the start of the range of pages to extract from the source document. If only <b>nStart</b> is specified then the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of the range of pages to extract from the source document. If only <b>nEnd</b> is specified then the range of pages is 0 to <b>nEnd</b> .
<b>cPath</b>	(optional) The device-independent pathname to save the new document. See 3.10.1 of the <a href="#">PDF Reference</a> for a description of the device-independent path name format. The path name may be relative to the location of the current document.  <b>NOTE:</b> (SecurityⓈ): The parameter <b>cPath</b> is required to have a <a href="#">Safe Path</a> and have a .pdf extension. This method will throw a <b>NotAllowedError</b> (see the <a href="#">Error Objects</a> ) exception if these security conditions are not met, and the method will fail.

### Returns

If **cPath** is not specified, returns the [Doc Object](#) for the new document; otherwise, returns the **null** object.

### Example

The following batch sequence would take each of the selected files, extract each page and save the page to a folder with a unique name. This example may be used in the following setting. Client's one-page bills are produced by an application and placed in a single PDF file. It is desired to separate the pages for distribution and/or separate printing jobs.

```
/* Extract Pages to Folder */
// regular expression used to acquire the base name of file
```

```

var re = /\.pdf$/i;
// filename is the base name of the file Acrobat is working on
var filename = this.documentFileName.replace(re, "");
try {for (var i = 0; i < this.numPages; i++)
    this.extractPages({
        nStart: i,
        cPath: "/F/temp/"+filename+"_" + i + ".pdf"
    });
} catch (e) { console.println("Aborted: " + e) }

```

## flattenPages

5.0	Ⓓ		✕	
-----	---	--	---	--

Converts all annotations in the specified page range to page contents. If a page range is not specified, converts annotation for all the pages in the current document.

**NOTE:** Great care must be used when using this method. All annotations—including form fields, comments and links—on the specified range of pages are flattened; they may have appearances, but they will no longer be annotations.

### Parameters

<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages in the current document. If only <b>nStart</b> is specified, then the page range is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages in the current document.
<b>nNonPrint</b>	(optional, version 6.0) This parameter determines how to handle non printing annotations. Values are 0 (default): Non-printing annotations are flattened. 1: Non-printing annotations are left as is. 2: Non-printing annotations are removed from the document.

### Returns

Nothing

### Example

Flatten all pages in the document.

```
this.flattenPages();
```

## getAnnot

5.0			<b>A</b>	<b>X</b>
-----	--	--	----------	----------

Gets the name of an **annot** object contained on a specific document page.

### Parameters

<b>nPage</b>	The page that contains the desired <a href="#">Annot Object</a> .
<b>cName</b>	The name of the desired <a href="#">Annot Object</a> .

### Returns

The [Annot Object](#), or **null** if there is no such annotation.

### Example

```
var ann = this.getAnnot(0, "OnMarketShare");
if (ann == null)
    console.println("Not Found!");
else
    console.println("Found it! type: " + ann.type);
```

## getAnnot3D

7.0				
-----	--	--	--	--

Gets the [Annot3D Object](#) with the given name from the given page.

### Parameters

<b>nPage</b>	The 0-based page number that contains the desired <a href="#">Annot3D Object</a> .
<b>cName</b>	The name of the desired <a href="#">Annot3D Object</a> .

### Returns

The [Annot3D Object](#), or **undefined**, if there is no such annotation.

## getAnnots

5.0			<b>A</b>	<b>X</b>
-----	--	--	----------	----------

Gets an array of [Annot Objects](#) satisfying specified criteria. See also [getAnnot](#) and [syncAnnotScan](#).

**Parameters**

<b>nPage</b>	(optional) A 0-based page number. If specified, gets only annotations on the given page. If not specified, gets annotations that meet the search criteria from all pages.
<b>nSortBy</b>	(optional) A sort method applied to the array. Values are: <b>ANSB_None</b> : (default) Do not sort; equivalent to not specifying this parameter. <b>ANSB_Page</b> : Use the page number as the primary sort criteria. <b>ANSB_Author</b> : Use the author as the primary sort criteria. <b>ANSB_ModDate</b> : Use the modification date as the primary sort criteria. <b>ANSB_Type</b> : Use the annot type as the primary sort criteria.
<b>bReverse</b>	(optional) If <b>true</b> , causes the array to be reverse sorted with respect to <b>nSortBy</b> .
<b>nFilterBy</b>	(optional) Gets only annotations satisfying certain criteria. Values are: <b>ANFB_ShouldNone</b> : (default) Get all annotations. Equivalent of not specifying this parameter. <b>ANFB_ShouldPrint</b> : Only include annotations that can be printed. <b>ANFB_ShouldView</b> : Only include annotations that can be viewed. <b>ANFB_ShouldEdit</b> : Only include annotations that can be edited. <b>ANFB_ShouldAppearInPanel</b> : Only annotations that appear in the annotations pane. <b>ANFB_ShouldSummarize</b> : Only include annotations that can be included in a summarization <b>ANFB_ShouldExport</b> : Only include annotations that can be included in an export

**Returns**

An array of [Annot Objects](#), or **null**, if none are found.

**Example**

```

this.syncAnnotScan();
var annots = this.getAnnots({
    nPage:0,
    nSortBy: ANSB_Author,
    bReverse: true
});
console.show();
console.println("Number of Annots: " + annots.length);

```

```
var msg = "%s in a %s annot said: \"%s\"";
for (var i = 0; i < annots.length; i++)
    console.println(util.printf(msg, annots[i].author, annots[i].type,
        annots[i].contents));
```

## getAnnots3D

7.0				
-----	--	--	--	--

Returns an array of [Annot3D Objects](#) for the given page.

### Parameters

<b>nPage</b>	The 0-based page number that contains the desired <a href="#">Annot3D Object</a> .
--------------	--

### Returns

An array of [Annot3D Objects](#), or **undefined**, if none is found.

## getDataObject

5.0				
-----	--	--	--	--

Obtains a specific **data** object. See also [dataObjects](#), [createDataObject](#), [exportDataObject](#), [importDataObject](#), [removeDataObject](#).

### Parameters

<b>cName</b>	The name of the <b>data</b> object to obtain.
--------------	---

### Returns

The [Data Object](#) corresponding to the specified name.

### Example

```
var MyData = this.getDataObject("MyData");
console.show(); console.clear();
for (var i in MyData) console.println("MyData." + i + "=" + MyData[i]);
```

## getDataObjectContents

7.0				
-----	--	--	--	--

This method allows access to the contents of the file attachment associated with a [DataObject](#).

## Parameters

<b>cName</b>	The name associated with the <a href="#">Data Object</a> to get.
<b>bAllowAuth</b>	(optional) The default value is <b>false</b> . If <b>true</b> , a dialog is used to obtain user authorization. Authorization may be required if the data object was encrypted using <a href="#">Doc.encryptForRecipients</a> . Authorization dialogs are allowed if <b>bAllowAuth</b> is true.

## Returns

### [ReadStream Object](#)

Related objects, properties and methods are the [Data Object](#), [doc.dataObjects](#), [doc.getDataObject](#), [doc.openDataObject](#), [doc.createDataObject](#), [doc.importDataObject](#), [doc.setDataObjectContents](#) and [doc.removeDataObject](#).

## ReadStream Object

A ReadStream Object is an object literal that represents a stream of data. It contains a method to allow reading data the stream.

Method	Parameters	Returns	Description
read	nBytes	String	The read method takes the number of bytes to read and returns a hex encoded string with the data from the stream. The read method is a destructive operation on the stream and returns a zero length string to indicate end of stream.

## Example

This code is part of a circulating memo. A PDF file is circulated among members on an email list. Each recipient enters a budget figure, then forwards the document on to the next person on the list. Before the document is sent, the budget number is appended to an embedded tab-delimited document, `budget.xls`, an attachment to this document. The last recipient can open the attachment, `budget.xls`, in a spreadsheet application to view the various budget numbers.

```
// get the name and department of the current recipient
var firstName = this.getField("Name.First").value;
var lastName = this.getField("Name.Last").value;
var deptName = this.getField("Dept.Name").value;
// get the budget number
var deptBudget = this.getField("Dept.Budget").value;
if ( app.viewerVersion >= 7 ) {
    // get the file stream object of the embedded file
    var oFile = this.getDataObjectContents("budget.xls");
    // convert to a string
    var myBudget = util.stringFromStream(oFile, "utf-8");
    // append current data to the end, using tabs to separate info
```



```

var myBudget = myBudget + "\r\n" + firstName
    + "\t" + lastName + "\t" + deptName + "\t" + deptBudget;
// convert back to a file stream
var oFile = util.streamFromString(myBudget, "uft-8");
// now "overwrite" budget.xls
this.setDataObjectContents("budget.xls", oFile);
} else {
    app.alert("Acrobat 7.0 or later is required."
        + " Your budget data will not be included. "
        + "Will e-mail on to the next correspondent, sorry. "
        + "Send in your budget request using traditional methods.");
}

```

The rest of the code, not shown, is to save the document and sent to the next person on the mailing list.

This examples uses `doc.getDataObjectContents`, `util.stringFromStream`, `util.streamFromString` and `doc.setDataObjectContents`.

## getField

3.01			
------	--	--	--

Maps a [Field Object](#) in the PDF document to a JavaScript variable.

Beginning with Acrobat 6.0, this method can return the [Field Object](#) of an individual Widget. For more information, see [Field Access from JavaScript](#).

### Parameters

<b>cName</b>	The name of the field of interest.
--------------	------------------------------------

### Returns

A [Field Object](#) representing a form field in the PDF document.

### Example 1

Make a text field multiline and triple its height

```

var f = this.getField("myText");
var aRect = f.rect;           // get bounding rectangle
f.multiline = true;         // make it multiline
var height = aRect[1]-aRect[3]; // calculate height
aRect[3] -= 2* height;      // triple the height of the text field
f.rect = aRect;             // and make it so

```

### Example 2 (Version 6.0)

Attach a JavaScript action to an individual widget, in this case, a Radio Button.

```

var f = this.getField("myRadio.0");
f.setAction("MouseUp",

```

```
"app.alert('Thanks for selecting the first choice.');"");
```

### Example 3

The following code lists all properties of a field. This technique can be used to programmatically duplicate a field.

```
f = this.getField("myField");
for ( var i in f ) {
  try {
    if ( typeof f[i] != "function" ) // remove a field methods
      console.println( i + ":" + f[i] )
  } catch(e) {} // an exception occurs when we get a property that
                // does not apply to this field type.
}
```

## getIcon

5.0			
-----	--	--	--

Obtains a specific **icon** object. See also [icons](#), [addIcon](#), [importIcon](#), and [removeIcon](#), and **field** methods [buttonGetIcon](#), [buttonImportIcon](#), and [buttonSetIcon](#).

### Parameters

<b>cName</b>	The name of the <b>icon</b> object to obtain.
--------------	---

### Returns

An [Icon Generic Object](#) associated with the specified name in the document or **null** if no icon of that name exists.

### Example

The following is a custom keystroke script from a combobox. The face names of the items in the combobox are the names of some of the icons that populate the document. As the user chooses different items from the combobox, the corresponding icon appears as the button face of the field "myPictures".

```
if (!event.willCommit) {
  var b = this.getField("myPictures");
  var i = this.getIcon(event.change);
  b.buttonSetIcon(i);
}
```

See **field**.[buttonSetIcon](#) for a more elaborate variation on this example.

## getLegalWarnings

6.0	Ⓣ		ⓧ	
-----	---	--	---	--

This method returns the legal warnings for this document in the form of an object with entries for each warning that has been found in the document. Legal warnings can be embedded in a file at the time that a file is signed by an author signature. Legal warnings can be embedded using the **cLegalAttest** of the **field.signatureSign** method.

The process that analyses a file to determine this list of warnings not available in the Adobe Reader. The value of each entry is the number of occurrences of this warning in the document. Refer to [PDF Reference 1.5](#).

### Parameters

---

<b>bExecute</b>	If <b>true</b> , will cause the file to be examined and all detected warnings will be returned. If <b>false</b> , the default value, the warnings that have been embedded in the file will be returned.
-----------------	---

---

### Returns

A object containing property names and values of legal warnings

### Example

Process a document and get legal PDF warnings.

```
var w = this.getLegalWarnings( true );
console.println( "Actual Legal PDF Warnings:" );
for(i in w) console.println( i + " = " + w[i] );

var w1 = this.getLegalWarnings( false );
console.println( "Declared Legal PDF Warnings:" );
for(i in w1) console.println( i + " = " + w1[i] );

// For an author signature, note also if annotations are
// allowed by MDP settings

var f = this.getField( "AuthorSig" );
var s = f.signatureInfo();
if( s.mdp == "defaultAndComments" )
    console.println( "Annotations are allowed" );

// What does author have to say about all this?

console.println( "Legal PDF Attestation:" );
console.println( w1.Attestation );
```

## getLinks

6.0				
-----	--	--	---	--

Gets an array of **link** objects that are enclosed within the specified **oCoords** on a specified page, **nPage**. See also [addLink](#), [removeLinks](#) and the [Link Object](#),

### Parameters

<b>nPage</b>	The page that contains the desired <b>link</b> object. The first page is 0.
<b>oCoords</b>	An array of four numbers in <i>rotated user space</i> , the coordinates of a rectangle listed in the following order: upper-left x, upper-left y, lower-right x and lower-right y.

### Returns

An array of [Link Objects](#).

### Example

Count the number of links in a document and report to the console.

```
var numLinks=0;
for ( var p = 0; p < this.numPages; p++)
{
    var b = this.getPageBox("Crop", p);
    var l = this.getLinks(p, b);
    console.println("Number of Links on page " + p + " is " + l.length);
    numLinks += l.length;
}
console.println("Number of Links in Document is " + numLinks);
```

## getNthFieldName

4.0			
-----	--	--	--

Gets the *n*th field name in the document. See also [numFields](#).

### Parameters

<b>nIndex</b>	The field name to obtain.
---------------	---------------------------

### Returns

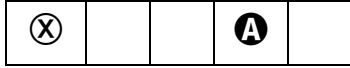
The name of the field in the document.

### Example

Enumerate through all of the fields in the document.

```
for (var i = 0; i < this.numFields; i++)
    console.println("Field[" + i + "] = " + this.getNthFieldName(i));
```

## getNthTemplate



Gets the name of the *n*th template within the document.

This method is superseded by the [templates](#) property, the [getTemplate](#) method, and the [Template Object](#) in later versions.

### Parameters

<b>nIndex</b>	The template to obtain.
---------------	-------------------------

### Returns

The name of the specified template.

## getOCGs



Gets an array of [OCG Objects](#) found on a specified page.

Related methods are `doc.getOCGOrder` and `doc.setOCGOrder`, and the [OCG Object](#).

### Parameters

<b>nPage</b>	(optional) The 0-based page number. If not specified, all the OCGs found in the document are returned.
--------------	--

### Returns

Returns an array of [OCG Objects](#) or `null` if no OCGs are present.

### Example

Turn on all the OCGs on the given document and page.

```
function TurnOnOCGsForPage(doc, nPage)
{
    var ocgArray = doc.getOCGs(nPage);
    for (var i=0; i < ocgArray.length; i++)
        ocgArray[i].state = true;
}
```

## getOCGOrder

7.0			
-----	--	--	--

Returns this document's OCGOrder array. This array represents how layers are displayed in the UI.

Related methods are `doc.getOCGs` and `doc.setOCGOrder`, and the [OCG Object](#).

### Parameters

None

### Returns

An array containing OCG objects, strings, and similar subarrays.

See [setOCGOrder](#) for a description of the order array.

## getPageBox

5.0			
-----	--	--	--

Gets a rectangle in *rotated user space* that encompasses the named box for the page. See also [setPageBoxes](#).

### Parameters

---

<b>cBox</b>	(optional) The type of box. Values are: Art Bleed BBox Crop ( <i>default</i> ) Trim  For definitions of these boxes see "Page Boundaries" in the <a href="#">PDF Reference</a> .
<b>nPage</b>	(optional) The 0-based index of the page. Default is 0, the first page in the document.

---

### Returns

A rectangle in *rotated user space* that encompasses the named box for the page.

### Example

Get the dimensions of the Media box.

```
var aRect = this.getPageBox("Media");
var width = aRect[2] - aRect[0];
var height = aRect[1] - aRect[3];
console.println("Page 1 has a width of " + width + " and a height of "
+ height);
```

## getPageLabel

5.0			
-----	--	--	--

Gets page label information for the specified page.

### Parameters

---

<b>nPage</b>	(optional) The 0-based index of the page. Default is 0, the first page in the document.
--------------	---

---

### Returns

Page label information for the specified page.

### Example

See [setPageLabels](#) for an example.

## getPageNthWord

5.0		Ⓢ	
-----	--	---	--

Gets the *n*th word on the page.

See also [getPageNumWords](#) and [selectPageNthWord](#).

**NOTE:** (Security Ⓢ): This method throws an exception if the document security is set to prevent content extraction.

### Parameters

---

<b>nPage</b>	(optional) The 0-based index of the page. Default is 0, the first page in the document.
<b>nWord</b>	(optional) The 0-based index of the word. Default is 0, the first word on the page.
<b>bStrip</b>	(optional) Whether punctuation and whitespace should be removed from the word before returning. Default is <b>true</b> .

---

### Returns

The *n*th word on the page.

### Example

See [Example 2](#) of [spell.checkWord](#) for an example.

## getPageNthWordQuads

5.0		Ⓢ	
-----	--	---	--

Gets the [quads](#) list for the *n*th word on the page. The [quads](#) can be used for constructing text markup annotations, [Underline](#), [StrikeOut](#), [Highlight](#) and [Squiggly](#). See also [getPageNthWord](#), [getPageNumWords](#), and [selectPageNthWord](#).

**NOTE:** (Security Ⓢ): This method throws an exception if the document security is set to prevent content extraction.

### Parameters

<b>nPage</b>	(optional) The 0-based index of the page. Default is 0, the first page in the document.
<b>nWord</b>	(optional) The 0-based index of the word. Default is 0, the first word on the page.

### Returns

The [quads](#) list for the *n*th word on the page.

### Example

The following example underlines the fifth word on the second page of a document.

```
var annot = this.addAnnot({
  page: 1,
  type: "Underline",
  quads: this.getPageNthWordQuads(1, 4),
  author: "A. C. Acrobat",
  contents: "Fifth word on second page"
});
```

See [spell.checkWord](#) for an additional example.

## getPageNumWords

5.0			
-----	--	--	--

Gets the number of words on the page.

See also [getPageNthWord](#), [getPageNthWordQuads](#), and [selectPageNthWord](#).

### Parameters

<b>nPage</b>	(optional) The 0-based index of the page. Default is 0, the first page in the document.
--------------	---



**Returns**

The number of words on the page.

**Example**

```
// count the number of words in a document
var cnt=0;
for (var p = 0; p < this.numPages; p++)
    cnt += getPageNumWords(p);
console.println("There are " + cnt + " words in this doc.");
```

See [Example 2](#) of `spell.checkWord` for an additional example.

**getPageRotation**

5.0			
-----	--	--	--

Gets the rotation of the specified page. See also [setPageRotations](#).

**Parameters**


---

**nPage** (optional) The 0-based index of the page. Default is 0, the first page in the document.

---

**Returns**

The rotation value of 0, 90, 180, or 270.

**getPageTransition**

5.0			
-----	--	--	--

Gets the transition of the specified page. See also [setPageTransitions](#).

**Parameters**


---

**nPage** (optional) The 0-based index of the page. Default is 0, the first page in the document.

---

**Returns**

An array of three values: [ **nDuration**, **cTransition**, **nTransDuration** ].

- **nDuration** is the maximum amount of time the page is displayed before the viewer automatically turns to the next page. A duration of -1 indicates that there is no automatic page turning.
- **cTransition** is the name of the transition to apply to the page. See the application property [transitions](#) for a list of valid transitions.

- **cTransDuration** is the duration (in seconds) of the transition effect.

## getPrintParams

6.0			
-----	--	--	--

Gets a **printParams** object that reflects the default print settings. See [print](#), which now takes the **printParams** object as its parameter.

### Parameters

None

### Returns

A [printParams Object](#).

### Example

Get the **printParams** object of the default printer.

```
var pp = this.getPrintParams();
pp.colorOverride = pp.colorOverrides.mono; // set some properties
this.print(pp); // print
```

## getSound

5.0			
-----	--	--	--

Gets the **sound** object corresponding to the specified name. See also [sounds](#), [importSound](#), [deleteSound](#), and the [Sound Object](#).

### Parameters

---

<b>cName</b>	The name of the object to obtain.
--------------	-----------------------------------

---

### Returns

The [Sound Object](#) corresponding to the specified name.

### Example

```
var s = this.getSound("Moo");
console.println("Playing the " + s.name + " sound.");
s.play();
```

## getTemplate

5.0			
-----	--	--	--

Gets the named template from the document. See also [templates](#), [createTemplate](#), [removeTemplate](#), and the [Template Object](#).

### Parameters

<b>cName</b>	The name of the template to retrieve.
--------------	---------------------------------------

### Returns

The [Template Object](#) or `null` if the named template does not exist in the document.

### Example

```
var t = this.getTemplate("myTemplate");
if ( t != null ) console.println( "myTemplate exists and is "
    + eval( '( t.hidden) ? "hidden" : "visible" ' ) + ".");
else console.println( "myTemplate is not present!");
```

## getURL

4.0	Ⓓ	Ⓒ	
-----	---	---	--

Gets the specified URL over the internet using a GET. If the current document is being viewed inside the browser, or Acrobat Web Capture is not available, the method uses the Weblink plug-in to retrieve the requested URL. If running inside Acrobat, the method gets the URL of the current document either from the [baseURL](#), from the URL of the first page (page 0) if the document was WebCaptured, or from the file system.

**NOTE:** This method roughly corresponds to the “open a web page” action.


A related method is [app.launchURL](#).

### Parameters

<b>cURL</b>	A fully qualified URL or a relative URL. There can be a query string at the end of the URL.
-------------	---

---

**bAppend** (optional) If **true** (the default), the resulting page or pages should be appended to the current document. This flag is considered to be **false** if the document is running inside the web browser, the Acrobat Web Capture plug-in is not available, or if the URL is of type "file:///".

**NOTE:** (Security Privileged versus Non-privileged Context.

**NOTE:** Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

---

**Returns**

Nothing

**Example**

```
this.getURL("http://www.adobe.com/", false);
```

**gotoNamedDest**

3.01			
------	--	--	--

Goes to a named destination within the PDF document. For details on named destinations and how to create them, see Section 8.2, *Document-Level Navigation*, of the [PDF Reference](#).

**Parameters**


---

<b>cName</b>	The name of the destination within a document.
--------------	--

---

**Returns**

Nothing

**Example**

The following example opens a document then goes to a named destination within that document.

```
// open new document
var myNovelDoc = app.openDoc("/c/fiction/myNovel.pdf");
// go to destination in this new doc
myNovelDoc.gotoNamedDest("chapter5");
// close old document
this.closeDoc();
```

## importAnFDF

4.0	Ⓓ		Ⓕ	
-----	---	--	---	--

Imports the specified FDF file. See also [importAnXFDF](#) and [importTextData](#).

### Parameters

---

<b>cPath</b>	(optional) The device-independent pathname to the FDF file. See Section 3.10.1 of the <a href="#">PDF Reference</a> for a description of the device-independent pathname format. It should look like the value of the <b>/F</b> key in an FDF exported with the <a href="#">submitForm</a> method or with the <b>Advanced &gt; Forms &gt; Export Form Data</b> menu item. The pathname may be relative to the location of the current document. If this parameter is omitted, a dialog is shown to let the user select the file.
--------------	--

---

### Returns

Nothing

### Example

The following code, which is an action of a Page Open event, checks whether a certain function, **ProcResponse**, is already defined, if not, it installs a document level JavaScript, which resides in an FDF file.

```
if(typeof ProcResponse == "undefined") this.importAnFDF("myDLJS.fdf");
```

Here, the pathname is a relative one. This technique may be useful for automatically installing document level JavaScripts for PDF files distilled from a PostScript file.

## importAnXFDF

5.0	Ⓓ		Ⓕ	
-----	---	--	---	--

Imports the specified XFDF file containing XML form data.

XFDF is an XML representation of Acrobat form data. See the document entitled "Forms System Implementation Notes" for an overview of XFDF, available as [Adobe Web Documentation](#). The *XML Form Data Format Specification*, XFDF, can be found at <http://partners.adobe.com/asn/tech/pdf/xmlformspec.jsp>.

See also [exportAsXFDF](#), [importAnFDF](#) and [importTextData](#).

**Parameters**


---




<b>cPath</b>	(optional) The device-independent pathname to the XPDF file. See Section 3.10.1 of the <a href="#">PDF Reference</a> for a description of the device-independent pathname format. The pathname may be relative to the location of the current document. If the parameter is omitted, a dialog is shown to let the user select the file.
--------------	---

---

**Returns**


Nothing

**importDataObject**

5.0				
-----	---	---	---	--

Imports an external file into the document and associates the specified name with the **data** object. Data objects can later be extracted or manipulated.

Related objects, properties and methods are the [Data Object](#), `doc.dataObjects`, `doc.getDataObject`, `doc.openDataObject`, `doc.createDataObject`, `doc.exportDataObject`, `doc.removeDataObject`, `doc.getDataObjectContents`, and `doc.setDataObjectContents`.

**NOTES:** (Security Privileged versus Non-privileged Context. The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

When a file attachment is imported using `Doc.importDataObject`, the value of its `Data.name` is assigned via the parameter **cName**; however, when a file is attached using the UI, the **name** of the file attachment is automatically assigned. The first attachment is assigned a **name** of "Untitled Object"; the second, a **name** of "Untitled Object 2"; the third, a **name** of "Untitled Object 3"; and so on.

**Parameters**


---

<b>cName</b>	The name to associate with the data object.
<b>cDIPath</b>	(optional) A device-independent path to a data file on the user's hard drive. This path may be absolute or relative to the current document. If not specified, the user is prompted to locate a data file. See <i>File Specification Strings</i> in the <a href="#">PDF Reference Manual</a> for the exact syntax of the path.

---

---

**cCryptFilter** (optional, version 6.0) The language independent name of a crypt filter to use when encrypting this data object. This crypt filter must have previously been added to the document's list of crypt filters, using the Document Object [addRecipientListCryptFilter](#) method, otherwise an exception will be thrown. The predefined **"Identity"** crypt filter can be used if it is desired that this data object not be encrypted in a file that is otherwise encrypted by the Document Object [encryptForRecipients](#) method.

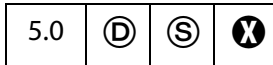
---

**Returns**

**true** on success. An exception is thrown on failure.

**Example**


```
function DumpDataObjectInfo(dataobj)
{
    for (var i in dataobj)
        console.println(dataobj.name + "[" + i + "]= " + dataobj[i]);
}
// Prompt the user for a data file to embed.
this.importDataObject("MyData");
DumpDataObjectInfo(this.getDataObject("MyData"));
// Embed Foo.xml (found in parent director for this doc).
this.importDataObject("MyData2", "../Foo.xml");
DumpDataObjectInfo(this.getDataObject("MyData2"));
```

**importIcon**

Imports an icon into the document and associates it with the specified name.

See also [icons](#), [addIcon](#), [getIcon](#), [removeIcon](#), [field](#) methods [buttonGetIcon](#), [buttonImportIcon](#), [buttonSetIcon](#), and the [Icon Generic Object](#).

Beginning with version 6.0, Acrobat will first attempt to open **cDIPath** as a PDF. On failure, Acrobat will try to convert **cDIPath** to PDF from one of the known graphics formats (BMP, GIF, JPEG, PCX, PNG, TIFF) and then import the converted file as a button icon.

**NOTES:** (Security ): If **cDIPath** is specified, this method can only be executed during batch, console or menu events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Parameters**


---

<b>cName</b>	The name to associate with the icon.
--------------	--------------------------------------

---

<b>cDIPath</b>	(optional) A device-independent path to a PDF file on the user's hard drive. This path may be absolute or relative to the current document. <b>cDIPath</b> may only be specified in a batch environment or from the console. See Section 3.10.1, "File Specification Strings" in the <a href="#">PDF Reference</a> for the exact syntax of the path. If not specified, the <b>nPage</b> parameter is ignored and the user is prompted to locate a PDF file and browse to a particular page.
<b>nPage</b>	(optional) The 0-based index of the page in the PDF file to import as an icon. Default is 0.

**Returns**

An integer code indicating whether it was successful or not:

- 0: No error
- 1: The user cancelled the dialog
- 1: The selected file could not be opened
- 2: The selected page was invalid

**Example**

This function is useful to populate a document with a series of named icons for later retrieval. For example, if a user of a document selects a particular state in a listbox, the author may want the picture of the state to appear next to the listbox. In prior versions of the application, this could be done using a number of fields that could be hidden and shown. This is difficult to author, however; instead, the appropriate script might be something like this:

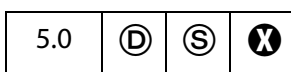
```
var f = this.getField("StateListBox");
var b = this.getField("StateButton");
b.buttonSetIcon(this.getIcon(f.value));
```

This uses a single field to perform the same effect.

A simple user interface can be constructed to add named icons to a document. Assume the existence of two fields: a field called **IconName** which will contain the icon name and a field called **IconAdd** which will add the icon to the document. The mouse up script for **IconAdd** would be:

```
var t = this.getField("IconName");
this.importIcon(t.value);
```

The same kind of script can be applied in a batch setting to populate a document with every selected icon file in a folder.

**importSound**

Imports a sound into the document and associates the specified name with the sound.



**NOTES:** (Security<sup>Ⓢ</sup>): If `cDIPath` is specified, this method can only be executed during batch, console, or menu events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

### Parameters

<b>cName</b>	The name to associate with the sound object.
<b>cDIPath</b>	(optional) A device-independent path to a sound file on the user's hard drive. This path may be absolute or relative to the current document. If not specified, the user is prompted to locate a sound file. See Section 3.10.1, "File Specification Strings", in the <a href="#">PDF Reference</a> for the exact syntax of the path.

### Returns

Nothing

### Example

```
this.importSound("Moo");
this.getSound("Moo").play();
this.importSound("Moof", "./moof.wav");
this.getSound("Moof").play();
```

See also [sounds](#), [getSound](#), [deleteSound](#), and the [Sound Object](#).

## importTextData

5.0	Ⓓ		Ⓕ	
-----	---	--	---	--

Imports a row of data from a text file. Each row must be *tab delimited*. The entries in the first row of the text file are the column names of the tab delimited data. These names are also field names for text fields present in the PDF file. The data row numbers are 0-based; that is, the first row of data is row zero (this does not include the column name row). When a row of data is imported, each column datum becomes the field value of the field that corresponds to the column to which the data belongs.

See also the export version of this method, [exportAsText](#).

**Parameters**

<b>cPath</b>	(optional) A relative device-independent path to the text file. If not specified, the user is prompted to locate the text data file.
<b>nRow</b>	(optional) The 0-based index of the row of the data to import, not counting the header row. If not specified, the user is prompted to select the row to import.

**Returns**

Integer, a return code. The return codes are given below.

Return Code	Description	Return Code	Description
-3	Warning: Missing Data	1	Error: Cannot Open File
-2	Warning: User Cancelled Row Select	2	Error: Cannot Load Data
-1	Warning: User Cancelled File Select	3	Error: Invalid Row
0	No Error		

**Example 1**

Suppose there are text fields named "First", "Middle" and "Last", and there is also a data file, the first row of which consists of the three strings, **First**, **Middle** and **Last**, separated by tabs. Suppose there are four additional rows of name data, again separated by tabs.

```

First           Middle           Last
A.              C.              Robot
T.              A.              Croba
A.              T.              Acrob
B.              A.              Tacro
// Import the first row of data from "myData.txt".
this.importTextData("/c/data/myData.txt", 0)

```

**Example (continued)**

The following code is a mouse up action for a button. Clicking on the button cycles through the text file and populates the three fields "First", "Middle" and "Last" with the name data.

```

if (typeof cnt == "undefined") cnt = 0;
this.importTextData("/c/data/textdata.txt", cnt++ % 4)

```

The same functionality can be obtained using the [ADBC Object](#) and associated properties and methods. The data file can be a spreadsheet or a database.

## importXFAData

6.0	Ⓓ	Ⓔ	Ⓕ	
-----	---	---	---	--

Imports the specified XFA file. See also [importAnXFDF](#) and [importTextData](#).

**NOTES:** (Security Ⓔ): This method is only allowed in batch, console, and menu events. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

### Parameters

---

<b>cPath</b>	(optional) The device-independent pathname to the XFA file. The pathname may be relative to the location of the current document. If this parameter is omitted a dialog is shown to let the user select the file.
--------------	---

---

### Returns

Nothing

## insertPages

5.0	Ⓓ	Ⓔ	ⓧ	
-----	---	---	---	--

Inserts pages from the source document into the current document. If a page range is not specified, the method gets all pages in the source document.

See also [deletePages](#) and [replacePages](#).

**NOTES:** (Security Ⓔ) This method can only be executed during batch, console, or menu events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Parameters**

<b>nPage</b>	(optional) The 0-based index of the page after which to insert the source document pages. Use -1 to insert pages before the first page of the document.
<b>cPath</b>	The device-independent pathname to the PDF file that will provide the inserted pages. See Section 3.10.1 of the <a href="#">PDF Reference</a> for a description of the device-independent pathname format. The pathname may be relative to the location of the current document.
<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages in the source document to insert. If only <b>nStart</b> is specified then the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages in the source document to insert. If only <b>nEnd</b> is specified then the range of pages is 0 to <b>nEnd</b> .

**Returns**

Nothing

**Example**

Insert a cover page to the current document.

```

this.insertPages ({
    nPage: -1,
    cPath: "/c/temp/myCoverPage.pdf",
    nStart: 0
});

```

**mailDoc**

4.0			<b>S</b>	
-----	--	--	----------	--

Saves the current PDF document and mails it as an attachment to all recipients, with or without user interaction.


See also [app.mailGetAddrs](#), [app.mailMsg](#), [doc.mailForm](#), [fdf.mail](#) and [report.mail](#).

**NOTES:** (Security **S**, version 7.0) When this method is executed in a non-privileged context, the **bUI** parameter is not honored and defaults to **true**. See [Privileged versus Non-privileged Context](#).

(Save Rights **S**) For Adobe Reader 5.1 and beyond, this method is commonly allowed, but document Save rights are required in case the document is changed.

On Windows, the client machine must have its default mail program configured to be MAPI enabled in order to use this method.

## Parameters

<b>bUI</b>	(optional) If <b>true</b> (the default), the rest of the parameters are used in a compose-new-message window that is displayed to the user. If <b>false</b> , the <b>cTo</b> parameter is required and all others are optional. <b>NOTE:</b> (Security  , version 7.0) When this method is executed in a non-privileged context, the <b>bUI</b> parameter is not honored and defaults to <b>true</b> . See <a href="#">Privileged versus Non-privileged Context</a> .
<b>cTo</b>	(optional) The semicolon-delimited list of recipients for the message.
<b>cCc</b>	(optional) The semicolon-delimited list of CC recipients for the message.
<b>cBcc</b>	(optional) The semicolon-delimited list of BCC recipients for the message.
<b>cSubject</b>	(optional) The subject of the message. The length limit is 64k bytes.
<b>cMsg</b>	(optional) The content of the message. The length limit is 64k bytes.

## Returns

Nothing

## Example

This pops up the compose-new-message window.

```
this.mailDoc(true);
```

This sends out mail with the attached PDF file to fun1@fun.com and fun2@fun.com. Beginning with version 7.0, the code below would have to be executed in a privileged context if the **bUI** parameter (set to **false**) is to be honored.

```
this.mailDoc({
  bUI: false,
  cTo: "apstory@ap.com",
  cCC: "dpsmith@ap.com",
  cSubject: "The Latest News",
  cMsg: "A.P., attached is my latest news story in PDF."
});
```

## mailForm

4.0				
-----	--	--	---	--

Exports the form data and mails the resulting FDF file as an attachment to all recipients, with or without user interaction. The method does not support signed signature fields.

See also `app.mailGetAddrs`, `app.mailMsg`, `doc.mailDoc`, `fdf.mail` and `report.mail`.

**NOTE:** On Windows, the client machine must have its default mail program configured to be MAPI enabled in order to use this method.

**Parameters**

<b>bUI</b>	If <b>true</b> , the rest of the parameters are used in a compose-new-message window that is displayed to the user. If <b>false</b> , the <b>cTo</b> parameter is required and all others are optional.
<b>cTo</b>	(required if <b>bUI</b> is <b>true</b> ) A semicolon-delimited list of recipients for the message.
<b>cCc</b>	(optional) A semicolon-delimited list of CC recipients for the message.
<b>cBcc</b>	(optional) A semicolon-delimited list of BCC recipients for the message.
<b>cSubject</b>	(optional) The subject of the message. The length limit is 64k bytes.
<b>cMsg</b>	(optional) The content of the message. The length limit is 64k bytes.

**Returns**

Nothing

**Example**

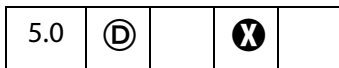
This pops up the compose new message window.

```
this.mailForm(true);
```

This sends out the mail with the attached FDF file to fun1@fun.com and fun2@fun.com.

```
this.mailForm(false, "fun1@fun.com; fun2@fun.com", "", "", "This is the subject", "This is the body of the mail.");
```

**movePage**



Moves a page within the document.

**Parameters**

<b>nPage</b>	(optional) The 0-based index of the page to move. Default is 0.
<b>nAfter</b>	(optional) The 0-based index of the page after which to move the specified page. Use -1 to move the page before the first page of the document. Default is the last page in the document.

**Returns**

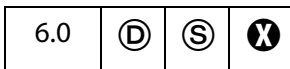
Nothing

**Example**

Reverse the pages in the document.

```
for (i = this.numPages - 1; i >= 0; i--) this.movePage(i);
```

**newPage**



Adds a new page to the active document in the Acrobat Viewer.

**NOTES:** (Security ): This method can only be executed during batch, console or menu events. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Parameters**

<b>nPage</b>	(optional) The page after which to add the new page in a 1-based page numbering system. The default is the last page of the document. Use 0 to add a page before the first page. An invalid page range is truncated to the valid range of pages.
<b>nWidth</b>	(optional) The width of the page in points. The default value is 612.
<b>nHeight</b>	(optional) The height of the page in points. The default value is 792.

**Returns**

Nothing

**Example**

Add a new page to match the page size of the doc.

```
var Rect = this.getPageBox("Crop");
this.newPage(0, Rect[2], Rect[1]);
```

## openDataObject

7.0				
-----	--	--	--	--

This method returns the [Doc Object](#) of a PDF document that is an embedded data object (i.e. an attachment) within the document that this method is being called for. The document is opened as a PDDoc.

The method can throw an exception instead of returning a doc object if

1. the document that this method is being called for does not contain the requested embedded data object;
2. the data object is not a PDF document;
3. permissions forbid opening attachments via JavaScript.

The document should be closed (using `doc.closeDoc`) after it is no longer needed.

### Parameters

---

<b>cName</b>	The name of the data object.
--------------	------------------------------

---

The name of a data object is a property of the [Data Object](#). A name is given to the object when it is embedded, automatically by the Acrobat UI, or programmatically by the JavaScript methods `doc.createDataObject` or `doc.importDataObject`.

### Returns

[Doc Object](#) or an exception is thrown

Related objects, properties and methods are the [Data Object](#), `doc.dataObjects`, `doc.setDataObjectContents`, `doc.getDataObjectContents`, `doc.createDataObject` and `doc.importDataObject`.

### Example

Open a PDF attachment and extract form data from it.

```
var oDoc = this.openDataObject("myAttachment");
try {
    var myField = this.getField("myTextField");
    // get the value of "yourTextField" in PDF attachment
    var yourField = oDoc.getField("yourTextField");
    // view this value in "myTextField"
    myField.value = yourField.value;
    oDoc.closeDoc();
} catch(e) { app.alert("Operation failed");}
```

See also ["Example 5 \(Version 7.0\)" on page 315](#) following the `doc.submitForm` method.



## print

3.01		Ⓢ		
------	--	---	--	--

Prints all or a specific number of pages of the document.

Beginning with Acrobat 6.0, the method can print the document using the settings contained in a [printParams Object](#), rather than through the other parameters. The permanent print settings are not altered.

**NOTES:** (Security Ⓢ, version 6.0) When printing to a file, the path must be a [Safe Path](#). The **print** method will not overwrite an existing file.

(Security Ⓢ, version 7.0) Non-interactive printing can only be executed during batch and console events. Printing is made non-interactive by setting **bUI** is to **false** or by setting the [interactive](#) property to silent, e.g.,

```
var pp = this.getPrintParams();
pp.interactive = pp.constants.interactionLevel.silent;
```

Outside of batch and console events, the values of **bUI** and of [interactive](#) are ignored, and a print dialog will always be presented.

**Doc.print** can only be executed during batch or console events when printing is set to non-interactive: **bUI** is set to **false**, or

See also [Privileged versus Non-privileged Context](#).

**NOTES:** On a Windows platform, the file name must include an extension of `.ps` or `.prn` (case insensitive). Additionally, the **print** method will not create a file directly in the root directory, the windows directory, or the windows system directory.

An **InvalidArgsError** (see the [Error Objects](#)) exception will be thrown and **print** will fail if any of the above security restrictions are not met.

### Parameters

<b>bUI</b>	(optional) If <b>true</b> (the default), will cause a UI to be presented to the user to obtain printing information and confirm the action.
<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages. If <b>nStart</b> and <b>nEnd</b> are not specified, prints all pages in the document. If only <b>nStart</b> is specified then the range of pages is the single page specified by <b>nStart</b> . If <b>nStart</b> and <b>nEnd</b> parameters are used, <b>bUI</b> must be <b>false</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of page. If <b>nStart</b> and <b>nEnd</b> are not specified, prints all pages in the document. If only <b>nEnd</b> is specified then the range of a pages is 0 to <b>nEnd</b> . If <b>nStart</b> and <b>nEnd</b> parameters are used, <b>bUI</b> must be <b>false</b> .

<b>bSilent</b>	(optional) If <b>true</b> , suppresses the cancel dialog box while the document is printing. The default is <b>false</b>
<b>bShrinkToFit</b>	(optional, version 5.0) If <b>true</b> , the page is shrunk (if necessary) to fit within the imageable area of the printed page. If <b>false</b> , it is not. The default is <b>false</b> .
<b>bPrintAsImage</b>	(optional, version 5.0) If <b>true</b> , print pages as an image. The default is <b>false</b> .
<b>bReverse</b>	(optional, version 5.0) If <b>true</b> , print from <b>nEnd</b> to <b>nStart</b> . The default is <b>false</b> .
<b>bAnnotations</b>	(optional, version 5.0) If <b>true</b> (the default), annotations are printed.
<b>printParams</b>	(optional, version 6.0) The <a href="#">printParams Object</a> containing the settings to use for printing. If this parameter is passed, any other parameters are ignored.

**Returns**

Nothing

**Example 1**

This example prints current page the document is on.

```
this.print(false, this.pageNum, this.pageNum);
// print a file silently
this.print({bUI: false, bSilent: true, bShrinkToFit: true});
```

**Example 2 (Version 6.0)**

```
var pp = this.getPrintParams();
pp.interactive = pp.constants.interactionLevel.automatic;
pp.printerName = "hp officejet d series";
this.print(pp);
```

**NOTE:** When [printerName](#) is an empty string and [fileName](#) is nonempty the current document is saved to disk as a PostScript file.

**Example 3 (Version 6.0)**

Save the current document as a PostScript file.

```
var pp = this.getPrintParams();
pp.fileName = "/c/temp/myDoc.ps";
pp.printerName = "";
this.print(pp);
```

## removeDataObject

5.0	ⓓ		ⓕ	
-----	---	--	---	--

Deletes the data object corresponding to the specified name from the document.

Related objects, properties and methods are the [Data Object](#), `doc.dataObjects`, `doc.getDataObject`, `doc.openDataObject`, `doc.createDataObject`, `doc.removeDataObject`, `doc.importDataObject`, `doc.getDataObjectContents`, and `doc.setDataObjectContents`.

### Parameters

<b>cName</b>	The name of the data object to remove.
--------------	--

The name of a data object is a property of the [Data Object](#). A name is given to the object when it is embedded, automatically by the Acrobat UI, or programmatically by the JavaScript methods `doc.createDataObject` or `doc.importDataObject`.

### Returns

Nothing

### Example

```
this.removeDataObject("MyData");
```

## removeField

5.0	ⓓ		ⓕ	
-----	---	--	---	--

Removes the specified field from the document. If the field appears on more than one page then all representations are removed.

**NOTE:** (ⓕ version 6.0): Beginning with version 6.0, `doc.removeField` can now be used from within Adobe Reader for documents with "Advanced Form Features".

### Parameters

<b>cName</b>	The field name to remove.
--------------	---------------------------

### Returns

Nothing

### Example

```
this.removeField("myBadField");
```

## removeIcon

5.0	Ⓓ		ⓧ
-----	---	--	---

Removes the specified named icon from the document.

See also [icons](#), [addIcon](#), [getIcon](#), and [importIcon](#), the `field` methods [buttonGetIcon](#), [buttonImportIcon](#), and [buttonSetIcon](#), and the [Icon Generic Object](#).

### Parameters

---

<b>cName</b>	The name of the icon to remove.
--------------	---------------------------------

---

The name of the icon is a property of the [Icon Generic Object](#). A name is given to the object either by `doc.importIcon`, when the icon file is imported into the document, or by `doc.addIcon`, which names an icon that is not in the document level named icons tree.

### Returns

Nothing

### Example

Remove all named icons from the document.

```
for ( var i = 0; i < this.icons.length; i++)
  this.removeIcon(this.icons[i].name);
```

## removeLinks

6.0	Ⓓ		ⓧ	
-----	---	--	---	--

Removes all the links on the specified page within the specified coordinates, if the user has permission to remove links from the document.

See also [addLink](#), [getLinks](#) and the [Link Object](#).

### Parameters

---

<b>nPage</b>	The 0-based index of the page from which to remove links.
<b>oCoords</b>	An array of four numbers in <i>rotated user space</i> , the coordinates of a rectangle listed in the following order: upper-left x, upper-left y, lower-right x and lower-right y.

---

### Returns

Nothing

**Example**

Remove all links from the document.

```
// remove all links from the document
for ( var p = 0; p < this.numPages; p++)
{
    var b = this.getPageBox("Crop", p);
    this.removeLinks(p, b);
}
```

Use [getLinks](#) to help count the number of links removed.

**removeScript**

7.0				
-----	--	--	--	--

This method removes a document level JavaScript—provided permissions for script removal is granted—specified by **cName**, the script name.

**Parameters**

<b>cName</b>	A string that specifies the name of the script to be removed.
--------------	---

**Returns**

The undefined value on success and throws an exception if the script is not found.

**Example**

Add a document level script, then remove it again.

```
this.addScript("myScript", "app.alert('A.C. Robot welcomes you!')");
```

Now remove this script:

```
this.removeScript("myScript");
```

**removeTemplate**

5.0	Ⓓ	Ⓔ	Ⓕ	
-----	---	---	---	--

Removes the named template from the document.

See also [templates](#), [createTemplate](#), [getTemplate](#), and the [Template Object](#).

**NOTES:** (Security Ⓔ): This method can only be executed during batch or console events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Parameters**

<b>cName</b>	The name of the template to remove.
--------------	-------------------------------------

The name of the template is a property of the [Template Object](#). A name is given to a template when it is created, either by the Acrobat UI or by the JavaScript method `doc.getTemplate`.

**Returns**

Nothing

**removeThumbnails**

5.0	Ⓓ		ⓧ	
-----	---	--	---	--

Deletes thumbnails for the specified pages in the document. See also [addThumbnails](#).

**Parameters**

<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document. If only <b>nStart</b> is specified, the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document. If only <b>nEnd</b> is specified, the range of pages is 0 to <b>nEnd</b> .

**Returns**

Nothing

**removeWeblinks**

5.0	Ⓓ		ⓧ	
-----	---	--	---	--

Scans the specified pages looking for links with actions to go to a particular URL on the web and deletes them. See also [addWeblinks](#).

**NOTE:** This method only removes weblinks authored in the application using the UI. Web links that are executed via JavaScript (for example, using [getURL](#)) are not removed.

**Parameters**

<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document. If only <b>nStart</b> is specified, the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document. If only <b>nEnd</b> is specified, the range of a pages is 0 to <b>nEnd</b> .

**Returns**




The number of web links removed from the document.

**Example**

Remove all web links from the document and report results to the console window.


```
var numWeblinks = this.removeWeblinks();
console.println("There were " + numWeblinks +
    " web links removed from the document.");
```

**replacePages**

5.0				
-----	---	---	---	--

Replaces pages in the current document with pages from the source document.

See also [deletePages](#), [extractPages](#), and [insertPages](#).

**NOTE:** (Security ): This method can only be executed during batch, console, or menu events. See the [Event Object](#) for a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Parameters**

<b>nPage</b>	(optional) The 0-based index of the page at which to start replacement. Default is 0.
<b>cPath</b>	The device-independent pathname to the PDF file that will provide the replacement pages. See Section 3.10.1 of the <a href="#">PDF Reference</a> for a description of the device-independent pathname format. The pathname may be relative to the location of the current document.

<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages in the source document to be used for replacement. If <b>nStart</b> and <b>nEnd</b> are not specified, gets all pages in the source document. If only <b>nStart</b> is specified, the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages in the source document to be used for replacement. If <b>nStart</b> and <b>nEnd</b> are not specified, gets all pages in the source document. If only <b>nEnd</b> is specified, the range of pages is 0 to <b>nEnd</b> .

**Returns**

Nothing

**resetForm**

3.01	ⓓ		
------	---	--	--

Resets the field values within a document.

**NOTE:** Resetting a field causes it to take on its default value, which in the case of text fields is usually blank.

**Parameters**

<b>aFields</b>	(optional) An array specifying the fields to reset. If not present or <b>null</b> , all fields in the form are reset. You can include non-terminal fields in the array.
----------------	---

**Returns**

Nothing

**Example 1**

Select fields to be reset and reset.

```
var fields = new Array();
fields[0] = "P1.OrderForm.Description";
fields[1] = "P1.OrderForm.Qty";
this.resetForm(fields);
```

or, the same fields can be reset using only one line of code:

```
this.resetForm(["P1.OrderForm.Description", "P1.OrderForm.Qty"]);
```

**Example 2**

This example illustrates how to reset a whole subtree. For example, if you pass "name" as part of the fields array then **name.first**, **name.last**, and so on, are reset.



```
this.resetForm(["name"]);
```

## saveAs

5.0		Ⓒ	Ⓓ	
-----	--	---	---	--

Saves the file to the device-independent path specified by the required parameter, **cPath**. The file is not saved in linearized format. Beginning with Acrobat 6.0, the document can be converted to another file type (other than PDF) and saved as specified by the value of the **cConvID** parameter.

**NOTES:** (Security Ⓒ): This method can only be executed during batch, console, or menu events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**NOTE:** (Adobe Reader Ⓓ): This method is available in the Adobe Reader for documents that have "Save rights".

### Parameters

<b>cPath</b>	The device-independent path in which to save the file.  <b>NOTE:</b> (Security Ⓒ): The parameter <b>cPath</b> is required to have a <a href="#">Safe Path</a> and have an extension appropriate to the value of <b>cConvID</b> . See the <a href="#">Values of cConvID and Valid Extensions</a> table below. This method will throw a <b>NotAllowedError</b> (see the <a href="#">Error Objects</a> ) exception if these security conditions are not met, and the method will fail.
<b>cConvID</b>	(optional, version 6.0) A conversion ID string that specifies the conversion file type. Currently supported values for <b>cConvID</b> are listed by the <code>app.fromPDFConverters</code> . If <b>cConvID</b> is not specified, then PDF is assumed.
<b>cFS</b>	(optional, version 7.0) A string that specifies the source file system name. Two values are supported: "" (the empty string) representing the default file system and "CHTTP". The default is the default file system. This parameter is only relevant if the web server supports WebDAV.
<b>bCopy</b>	(optional, version 7.0) A boolean which, if <b>true</b> , saves the PDF file as a copy. The default is <b>false</b> .
<b>bPromptToOverwrite</b>	(optional, version 7.0) A boolean which, if <b>true</b> , prompts the user if the destination file already exists. The default is <b>false</b> .

**Returns**

The value `undefined` is returned on success. An exception is thrown if an error occurs. For example, this method will throw a `NotAllowedError` (see the [Error Objects](#)) if the users disallows an overwrite.

**NOTE:** Prior to Version 7.0, this method had no return value.

**Values of cConvID and Valid Extensions**

cConvID	Valid Extensions
<code>com.adobe.acrobat.eps</code>	eps
<code>com.adobe.acrobat.html-3-20</code>	html, htm
<code>com.adobe.acrobat.html-4-01-css-1-00</code>	html, htm
<code>com.adobe.acrobat.jpeg</code>	jpeg, jpg, jpe
<code>com.adobe.acrobat.jp2k</code>	jpf, jpx, jp2, j2k, j2c, jpc
<code>com.adobe.acrobat.doc</code>	doc
<code>com.adobe.acrobat.png</code>	png
<code>com.adobe.acrobat.ps</code>	ps
<code>com.adobe.acrobat.rtf</code>	rft
<code>com.adobe.acrobat.accesstext</code>	txt
<code>com.adobe.acrobat.plain-text</code>	txt
<code>com.adobe.acrobat.tiff</code>	tiff, tif
<code>com.adobe.acrobat.xml-1-00</code>	xml

**NOTES:** When the conversion ID corresponds to `jpeg`, `jp2k`, `png`, or `tiff`, this method saves each page individually under a filename obtained by appending "`_Page_#`" to the basename of the filename provided. For example, if the value of the `cPath` is `"/C/temp/mySaveAsDocs/myJPGs.jpg"`, then names of the files generated will be `myJPGs_Page_1.jpg`, `myJPGs_Page_2.jpg`, and so on.

**Example 1**

The following code could appear as a batch sequence. Assume there is a PDF file already open containing form files that needs to be populated from a database and saved. Below is an outline of the script:

```
// code lines to read from a database and populate the form with data
// now save file to a folder; use customerID from database record
// as name
var row = statement.getRow();
```

```
.....
this.saveAs("/c/customer/invoices/" + row.customerID + ".pdf");
```

### Example 2

You can use [newDoc](#) and [addField](#) to dynamically layout a form, then populate it from a database and save.

```
var myDoc = app.newDoc()
// layout some dynamic form fields
// connect to database, populate with data, perhaps from a database
.....
// save the doc and/or print it; print it silently this time
// to default printer
myDoc.saveAs("/c/customer/invoices/" + row.customerID + ".pdf");
myDoc.closeDoc(true);           // close the doc, no notification
```

### Example 3 (Version 6.0)

Save the current document in rich text format:

```
this.saveAs("/c/myDocs/myDoc.rtf", "com.adobe.acrobat.rtf");
```

See [fromPDFConverters](#) for a listing of supported conversion ID strings.

### Example 3 (Version 7.0)

Save the document to a WebDAV folder.

```
this.saveAs({
  cPath: "http://www.myCom.com/WebDAV/myDoc.pdf",
  bPromptToOverwrite: true,
  cFS: "CHTTP"
});
```

## scroll

3.01			
------	--	--	--

Scrolls the specified point on the current page into middle of the current view. These coordinates must be defined in *rotated user space*. See the [PDF Reference](#) for details on the user space coordinate system.

### Parameters

<b>nX</b>	The x-coordinate for the point to scroll.
<b>nY</b>	The y-coordinate for the point to scroll.

### Returns

Nothing

## selectPageNthWord

5.0			
-----	--	--	--

Changes the current page number and selects the specified word on the page.

See also [getPageNthWord](#), [getPageNthWordQuads](#) and [getPageNumWords](#).

### Parameters

<b>nPage</b>	(optional) The 0-based index of the page to operate on. Default is 0, the first page in the document.
<b>nWord</b>	(optional) The 0-based index of the word to obtain. Default is 0, the first word on the page.
<b>bScroll</b>	(optional) Whether to scroll the selected word into the view if it is not already viewable. Default is <b>true</b> .

### Returns

Nothing

### Example

Get and select a particular word.

```
// get the 20th word on page 2 (page 1, 0-based)
var cWord = this.getPageNthWord(1, 20);
// Select that word (highlight) for the user to see, change page if
// necessary.
this.selectPageNthWord(1, 20);
```

## setAction

6.0	Ⓧ		Ⓧ	
-----	---	--	---	--

Sets the JavaScript action of the document for a given trigger.

See also [doc.addScript](#), [doc.setAction](#), [bookmark.setAction](#), and [field.setAction](#).

**NOTE:** This method will overwrite any action already defined for the selected trigger.

**Parameters**

<b>cTrigger</b>	The name of the trigger point to which to attach the action. Values are: WillClose WillSave DidSave WillPrint DidPrint
<b>cScript</b>	The JavaScript expression to be executed when the trigger is activated.

**Returns**

Nothing

**Example**

This example insert **WillSave** and **DidSave** actions. The code gets the filesize before saving and after saving, and compares the two.

```
// WillSave Script
var myWillSave = 'var filesizeBeforeSave = this.filesize;\r'
  + 'console.println("File size before saving is " + '
  + ' filesizeBeforeSave );';

// DidSave Script
var myDidSave = 'var filesizeAfterSave = this.filesize;\r'
  + 'console.println("File size after saving is "'
  + 'filesizeAfterSave);\r'
  + 'var difference = filesizeAfterSave - filesizeBeforeSave;\r'
  + 'console.println("The difference is " + difference );\r'
  + 'if ( difference < 0 )\r\t'
  + 'console.println("Reduced filesize!");\r'
  + 'else\r\t'
  + 'console.println("Increased filesize!");'

// Set Document Actions...
this.setAction("WillSave", myWillSave);
this.setAction("DidSave", myDidSave);
```

**setDataObjectContents**

7.0			<b>D</b>	
-----	--	--	----------	--

This method replaces the file attachment specified by the parameter **cName** with the contents of the **oStream** parameter.

**Parameters**

<b>cName</b>	The name associated with the <a href="#">Data Object</a> that is to be replaced with <b>oStream</b> .
<b>oStream</b>	A <a href="#">ReadStream Object</a> representing the contents of the file attachment.
<b>cCryptFilter</b>	(optional) The language independent name of a crypt filter to use when encrypting this data object. This crypt filter must have previously been added to the document's list of crypt filters, using the <a href="#">doc.addRecipientListCryptFilter</a> method, otherwise, an exception will be thrown. The predefined "Identity" crypt filter can be used if it is desired that this data object not be encrypted in a file that is otherwise encrypted by the <a href="#">doc.encryptForRecipients</a> method.

**Returns**

Nothing

**Example 1**

See the [Example](#) on [page 264](#).

**Example 2**

This document has a file attachment named `acrobat.xml`. The attachment is opened, the XML data is updated, then the new XML document is saved back to the attachment. It is possible to submit this XML file attachment. See ["Example 5 \(Version 7.0\)" on page 315](#), following the [Doc.submitForm\(\)](#) method. This example uses the XML data defined in the [Example](#) following [XMLData.applyXPath\(\)](#).

```
// get the file stream object of the attachment
var acrobat = this.getDataObjectContents("acrobat.xml");

// convert to a string
var cAcrobat = util.stringFromStream(acrobat, "utf-8");

// parse this and get XFAObject
var myXML = XMLData.parse(cAcrobat, false);

// change the value of grandad's income
myXML.family.grandad.personal.income.value = "300000";

// save XML document as string, cAcrobat
var cAcrobat = myXML.saveXML('pretty');

// convert to a file stream
var acrobat = util.streamFromString(cAcrobat, "utf-8");

// now "update" the attachment acrobat.xml with this file stream
```

```
this.setDataObjectContents("acrobat.xml", acrobat);
```

Related objects, properties and methods are the [Data Object](#), [doc.dataObjects](#), [doc.getDataObject](#), [doc.openDataObject](#), [doc.createDataObject](#), [doc.importDataObject](#), [doc.getDataObjectContents](#) and [doc.removeDataObject](#).

## setOCGOrder

7.0	Ⓓ		ⓧ	ⓧ
-----	---	--	---	---

Sets this document's OCGOrder array. This array represents how layers are displayed in the UI.

The simplest order array is a flat array of OCG objects. In this case the listed OCGs are displayed in the UI as a flat list in the same order. If a subarray is present in the order array, and the first element of the array is a string, then the string will be listed with the rest of the array nested underneath it. If the first element of the array is not a string, then the entire array will appear nested underneath the OCG preceding the subarray.

Related methods are [doc.getOCGs](#) and [doc.getOCGOrder](#), and the [OCG Object](#).

### Parameters

---

**oOrderArray** The array to be used as this document's OCG Order array.

---

### Returns

Nothing

### Example

Reverse the order of OCGs as listed in the UI.

```
var ocgOrder = this.getOCGOrder();
var newOrder = new Array();
for (var j=0; j < ocgOrder.length; j++)
    newOrder[j] = ocgOrder[ocgOrder.length-j-1];
this.setOCGOrder(newOrder);
```

## setPageAction

6.0	Ⓓ		ⓧ	
-----	---	--	---	--

Sets the action of a page in a document for a given trigger.

See also [doc.setAction](#), [doc.addScript](#), [bookmark.setAction](#), and [field.setAction](#).

**NOTE:** This method will overwrite any action already defined for the chosen page and trigger.

**Parameters**

<b>nPage</b>	The 0-based index of the page in the document to which an action is added.
<b>cTrigger</b>	The trigger for the action. Values are: Open Close
<b>cScript</b>	The JavaScript expression to be executed when the trigger is activated.

**Returns**

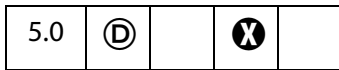
Nothing

**Example**

This example causes the application to beep when the first page is opened.

```
this.setPageAction(0, "Open", "app.beep(0);");
```

**setPageBoxes**



Sets a rectangle that encompasses the named box for the specified pages.

See also [getPageBox](#).

**Parameters**

<b>cBox</b>	(optional) The box type value, one of: Art Bleed Crop Media Trim  Note that the <b>BBox</b> box type is read-only and only supported in <a href="#">getPageBox</a> . For definitions of these boxes see Section 8.6.1, "Page Boundaries" in the <a href="#">PDF Reference</a> .
<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages in the document to be operated on. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document. If only <b>nStart</b> is specified, the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages in the document to be operated on. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document.



---

<b>rBox</b>	(optional) An array of four numbers in <i>rotated user space</i> to which to set the specified box. If not provided, the specified box is removed.
-------------	--

---

**Returns**

Nothing

**setPageLabels**

5.0	Ⓓ		ⓧ	
-----	---	--	---	--

Establishes the numbering scheme for the specified page and all pages following it until the next page with an attached label is encountered.

See also [getPageLabel](#).

**Parameters**


---

<b>nPage</b>	(optional) The 0-based index for the page to be labelled.
<b>aLabel</b>	(optional) An array of three required items [ <b>cStyle</b> , <b>cPrefix</b> , <b>nStart</b> ]. <ul style="list-style-type: none"> <li>● <b>cStyle</b> is the style of page numbering. Can be: <ul style="list-style-type: none"> <li><b>D</b>: decimal numbering</li> <li><b>R</b> or <b>r</b>: roman numbering, upper or lower case</li> <li><b>A</b> or <b>a</b>: alphabetic numbering, upper or lower case</li> </ul>           See the <a href="#">PDF Reference</a>, Section 7.3.1, for the exact definitions of these styles. </li> <li>● <b>cPrefix</b> is a string to prefix the numeric portion of the page label.</li> <li>● <b>nStart</b> is the ordinal with which to start numbering the pages. If not supplied, any page numbering is removed for the specified page and any others up to the next specified label. The value of <b>aLabel</b> cannot be <b>null</b>.</li> </ul>

---

**Returns**

Nothing

**Example 1**

10 pages in the document, label the first 3 with small roman numerals, the next 5 with numbers (starting at 1) and the last 2 with an "Appendix- prefix" and alphabetic.

```
this.setPageLabels(0, [ "r", "", 1]);
this.setPageLabels(3, [ "D", "", 1]);
this.setPageLabels(8, [ "A", "Appendix-", 1]);
var s = this.getPageLabel(0);
for (var i = 1; i < this.numPages; i++)
```

```
s += ", " + this.getPageLabel(i);
console.println(s);
```

The example will produce the following output on the console:

```
i, ii, iii, 1, 2, 3, 4, 5, Appendix-A, Appendix-B
```

**Example 2**

Remove all page labels from a document.

```
for (var i = 0; i < this.numPages; i++) {
  if (i + 1 != this.getPageLabel(i)) {
    // Page label does not match ordinal page number.
    this.setPageLabels(i);
  }
}
```

**setPageRotations**



Rotates the specified pages in the current document.

See also [getPageRotation](#).

**Parameters**

<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages in the document to be operated on. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document. If only <b>nStart</b> is specified, the range of pages is the single page specified by <b>nStart</b> .
<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages in the document to be operated on. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document. If only <b>nEnd</b> is specified, the range of pages is 0 to <b>nEnd</b> .
<b>nRotate</b>	(optional) The amount of rotation that should be applied to the target pages. Can be 0, 90, 180, or 270. Default is 0.

**Returns**

Nothing

**Example**

Rotate pages 0 through 10 of the current document.

```
this.setPageRotations(0, 10, 90);
```

## setPageTabOrder

6.0	Ⓧ		Ⓧ	
-----	---	--	---	--

Sets the tab order of the form fields on a page. The tab order can be set by row, by column, or by structure.

If a PDF 1.4 documents is viewed in Acrobat 6.0, tabbing between fields is in the same order as it is in Acrobat 5.0. Similarly, if a PDF 1.5 document is opened in Acrobat 5.0, the tabbing order for fields is the same as it is in Acrobat 6.0.

### Parameters

<b>nPage</b>	The 0-based index of the page number on which the tabbing order is to be set.
<b>cOrder</b>	The order to be used. Values are: rows columns structure

### Returns

Nothing

### Example

Set the page tab order for all pages to **rows**.

```
for (var i = 0; i < this.numPages; i++)
  this.setPageTabOrder(i, "rows");
```

## setPageTransitions

5.0	Ⓧ		Ⓧ	
-----	---	--	---	--

Sets the page transition for a specific range of pages.

See also [getPageTransition](#).

### Parameters

<b>nStart</b>	(optional) A 0-based index that defines the start of an inclusive range of pages in the document to be operated on. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document. If only <b>nStart</b> is specified, the range of pages is the single page specified by <b>nStart</b> .
---------------	---

---

<b>nEnd</b>	(optional) A 0-based index that defines the end of an inclusive range of pages in the document to be operated on. If <b>nStart</b> and <b>nEnd</b> are not specified, operates on all pages in the document. If only <b>nEnd</b> is specified, the range of pages is 0 to <b>nEnd</b> .
<hr/>	
<b>aTrans</b>	(optional) The page transition array consists of three values: [ <b>nDuration</b> , <b>cTransition</b> , <b>nTransDuration</b> ]. <ul style="list-style-type: none"> <li>● <b>nDuration</b> is the maximum amount of time the page is displayed before the viewer automatically turns to the next page. Set to -1 to turn off automatic page turning.</li> <li>● <b>cTransition</b> is the name of the transition to apply to the page. See <b>fullScreen.transitions</b> for a list of valid transitions.</li> <li>● <b>nTransDuration</b> is the duration (in seconds) of the transition effect.</li> </ul> If <b>aTrans</b> is not present, any page transitions for the pages are removed.

---

**Returns**

Nothing

**Example**

Put document into fullscreen mode, and apply some transitions.

```

this.setPageTransitions({ aTrans: [-1, "Random", 1] });
app.fs.isFullScreen=true;
    
```

**spawnPageFromTemplate**



Spawns a page in the document using the given template, as returned by [getNthTemplate](#).

See [templates](#), [createTemplate](#), and [template.spawn](#), which supersede this method in later versions.

**NOTE:** The template feature does not work in Adobe Reader.

**Parameters**

---

<b>cTemplate</b>	The template name.
<hr/>	
<b>nPage</b>	(optional) The 0-based page number before which or into which the template is spawned, depending on the value of <b>bOverlay</b> . If <b>nPage</b> is omitted, a new page is created at the end of the document.
<hr/>	
<b>bRename</b>	(optional) Whether fields should be renamed. The default is <b>true</b> .

---

<b>bOverlay</b>	(optional, version 4.0) If <b>false</b> , the template is inserted before the page specified by <b>nPage</b> . When <b>true</b> (the default) it is overlaid on top of that page.
<b>oXObject</b>	(optional, version 6.0) The value of this parameter is the return value of an earlier call to <b>spawnPageFromTemplate</b> .

**Returns**

Prior to Acrobat 6.0, this method returned nothing. Now, this method returns an object representing the page contents of the page spawned. This return object can then be used as the value of the optional parameter **oXObject** for subsequent calls to **spawnPageFromTemplate**.

**NOTE:** Repeatedly spawning the *same* page can cause a large inflation in the file size. To avoid this file size inflation problem, **spawnPageFromTemplate** now returns an object that represents the page contents of the spawned page. This return value can be used as the value of the **oXObject** parameter in subsequent calls to the **spawnPageFromTemplate** method to spawn the same page.

**Example 1**

```
var n = this.numTemplates;
var cTempl;
for (i = 0; i < n; i++) {
    cTempl = this.getNthTemplate(i);
    this.spawnPageFromTemplate(cTempl);
}
```

**Example 2 (version 6.0)**

The following example spawns the same template 31 times using the **oXObject** parameter and return value. Using this technique avoids overly inflating the file size.

```
var t = this.getNthTemplate(0)
var XO = this.spawnPageFromTemplate(t, this.numPages, false, false);
for (var i=0; i < 30; i++)
    this.spawnPageFromTemplate(t,this.numPages, false, false, XO);
```

**submitForm**

3.01			
------	--	--	--

Submits the form to a specified URL. To call this method , you must be running inside a web browser or have the Acrobat Web Capture plug-in installed (unless the URL uses the "mailto" scheme, in which case it will be honored even if not running inside a web browser, as long as the SendMail plug-in is present). Beginning with Adobe Reader 6.0, you need not be inside a web browser to call this method.

**NOTE:** (Version 6.0) Depending on the parameters passed, there are restrictions on the use of submitForm. See the notes embedded in the description of the parameters.

The `https` protocol is supported for secure connections.

## Parameters

<b>cURL</b>	The URL to submit to. This string must end in <b>#FDF</b> if the result from the submission is FDF or XFDF (that is, the value of <b>cSubmitAs</b> is "FDF" or "XFDF") and the document is being viewed inside a browser window.
<b>bFDF</b>	⊗ (optional) Whether to submit as FDF or HTML. If <b>true</b> , the default, submits the form data as FDF. If <b>false</b> , submits it as URL-encoded HTML. This option has been deprecated, use <b>cSubmitAs</b> instead.
<b>bEmpty</b>	(optional) When <b>true</b> , submit all fields, including those that have no value. When <b>false</b> (the default), exclude fields that currently have no value. <b>NOTE:</b> If data is submitted as XDP, XML or XFD (see the <b>cSubmitAs</b> parameter, below), the <b>bEmpty</b> parameter is ignored. All fields are submitted, even fields that are empty. See <b>aFields</b> below.
<b>aFields</b>	(optional) An array of field names to submit or a string containing a single field name. <ul style="list-style-type: none"> <li>● If supplied, only the fields indicated are submitted, except those excluded by <b>bEmpty</b>.</li> <li>● If omitted or <b>null</b>, all fields are submitted, except those excluded by <b>bEmpty</b>.</li> <li>● If an empty array, no fields are submitted. A submitted FDF might still contain data if <b>bAnnotations</b> is <b>true</b>.</li> </ul> You can specify non-terminal field names to export an entire subtree of fields. <b>NOTE:</b> If data is submitted as XDP, XML or XFD (see the <b>cSubmitAs</b> parameter, below), the <b>aFields</b> parameter is ignored. All fields are submitted, even fields that are empty. See <b>bEmpty</b> above.
<b>bGet</b>	(optional, version 4.0) When <b>true</b> , submit using the HTTP GET method. When <b>false</b> (the default), use a POST. GET is only allowed if using Acrobat Web Capture to submit (the browser interface only supports POST) and only if the data is sent as HTML (that is, <b>cSubmitAs</b> is <b>HTML</b> ).
<b>bAnnotations</b>	(optional, version 5.0) When <b>true</b> , annotations are included in the submitted FDF or XML. The default is <b>false</b> . Only applicable if <b>cSubmitAs</b> is <b>FDF</b> or <b>XFDF</b> .

<b>bXML</b>	<p>ⓧ (optional, version 5.0) If <b>true</b>, submit as XML. The default is <b>false</b>.</p> <p>This option has been deprecated, use <b>cSubmitAs</b> instead.</p>
<b>bIncrChanges</b>	<p>(optional, version 5.0) When <b>true</b>, include the incremental changes to the PDF in the submitted FDF. The default is <b>false</b>. Only applicable if <b>cSubmitAs</b> is <b>FDF</b>. Not available in the Adobe Reader.</p>
<b>bPDF</b>	<p>ⓧ (optional, version 5.0) If <b>true</b>, submit the complete PDF document. The default is <b>false</b>. When <b>true</b>, all other parameters except <b>cURL</b> are ignored. Not available in the Adobe Reader.</p> <p>This option has been deprecated, use <b>cSubmitAs</b> instead.</p>
<b>bCanonical</b>	<p>(optional, version 5.0) When <b>true</b>, convert any dates being submitted to standard format (that is, <b>D:YYYYMMDDHHmmSSOHH' mm'</b>; see the <a href="#">PDF Reference</a> for details). The default is <b>false</b>.</p>
<b>bExclNonUserAnnots</b>	<p>(optional, version 5.0) A boolean that indicates, if <b>true</b>, to exclude any annotations that are not owned by the current user. The default is <b>false</b>.</p>
<b>bExclFKey</b>	<p>(optional, version 5.0) When <b>true</b>, exclude the "F" key. The default is <b>false</b>.</p>
<b>cPassword</b>	<p>(optional, Version 5.0) The password to use to generate the encryption key, if the FDF needs to be encrypted before getting submitted.</p> <p>Pass the value <b>true</b> (no quotes), to use the password that the user has previously entered (within this Acrobat session) for submitting or receiving an encrypted FDF. If no password has been entered, prompts the user to enter a password.</p> <p>Regardless of whether the password is passed in or requested from the user, this new password is remembered within this Acrobat session for future outgoing or incoming encrypted FDFs.</p> <p>Only applicable if <b>cSubmitAs</b> is <b>FDF</b>.</p>
<b>bEmbedForm</b>	<p>(optional, version 6.0) When <b>true</b>, the call embeds the entire form from which the data is being submitted in the FDF.</p> <p>Only applicable if <b>cSubmitAs</b> is <b>FDF</b>.</p>


---

<b>oJavaScript</b>	<p>(optional, version 6.0) Can be used to include <code>Before</code>, <code>After</code>, and <code>Doc</code> JavaScripts in a submitted FDF. If present, the value is converted directly to an analogous <code>CosObj</code> and used as the <code>/JavaScript</code> attribute in the FDF. For example:</p> <pre>oJavaScript: {   Before: 'app.alert("before!")',   After: 'app.alert("after!")',   Doc: [ "MyDocScript1", "myFunc1()",         "MyDocScript2", "myFunc2()" ] }</pre>
	Only applicable if <code>cSubmitAs</code> is <b>FDF</b> .
<b>cSubmitAs</b>	<p>(optional, version 6.0) This parameter indicates the format for submission. Values are</p> <pre>FDF (default) XFDF HTML XDP XML XFD PDF</pre> <p><b>NOTE:</b> Additional notes on the values of this parameter:</p> <ul style="list-style-type: none"> <li>● <b>PDF</b> means submit the complete PDF document; in this case, all other parameters except <code>cURL</code> are ignored.</li> <li>● Save rights required (S): The <b>PDF</b> choice is not available in Adobe Reader, unless the document has save rights.</li> <li>● (version 7.0) If <b>XML</b> is the value of this parameter, the form data for the current document is submitted in XML format, unless the parameter <code>oXML</code> (new to version 7.0) contains a valid <a href="#">XMLData Object</a>, in which case that is what gets submitted instead.</li> </ul> <p>This parameter supercedes the individual format parameters; however, they are considered in the following priority order, from high to low: <code>cSubmitAs</code>, <code>bPDF</code>, <code>bXML</code>, <code>bFDF</code>.</p>
<b>bInclNMKey</b>	<p>(optional, version 6.0) When <code>true</code>, include the "NM" key of any annotations. The default is <code>false</code>.</p>

---



---

<b>aPackets</b>	<p>(optional, version 6.0) An array of strings, specifying which packets to include in an XDP submission. Possible strings are:</p> <pre> template datasets stylesheet xfdf sourceSet pdf config *</pre> <p>This parameter is only applicable if <b>cSubmitAs</b> is <b>XDP</b>. <b>pdf</b> means that the PDF should be embedded; if <b>pdf</b> is not included here, only a link to the PDF is included in the XDP. <b>xfdf</b> means to include annotations in the XDP (since that packet uses XFDF format). <b>*</b> means that all packets should be included in the XDP. The default for <b>pdf</b> is to include it as a <i>reference</i>. To embed the PDF file in the XDP, <i>explicitly</i> specify <b>pdf</b> as one of the packets.</p> <p><b>NOTE:</b> (Save rights required  </p>
<b>cCharset</b>	<p>(optional, version 6.0) The encoding for the values submitted. String values are:</p> <pre> utf-8 utf-16 Shift-JIS BigFive GBK UHC</pre> <p>If not passed, the current Acrobat behavior applies. For XML-based formats, <b>utf-8</b> is used. For other formats, Acrobat tries to find the best host encoding for the values being submitted.</p> <p>XFDF submission ignores this value and always uses <b>utf-8</b>.</p>
<b>oXML</b>	<p>(optional, version 7.0) This parameter is only applicable if <b>cSubmitAs</b> equals <b>XML</b>. It should be an <a href="#">XMLData Object</a>, which will get submitted.</p>

---

<b>cPermID</b>	(optional, version 7.0) Specifies a permanent ID to assign to the PDF that is submitted if either the value of <b>cSubmitAs</b> is <b>PDF</b> or <b>bEmbedForm</b> is <b>true</b> . This permanent ID is the first entry in the <b>doc.docID</b> array ( <b>doc.docID[0]</b> ). Does not affect the current document.
<b>cInstID</b>	(optional, version 7.0) Specifies an instance ID to assign to the PDF that is submitted if either the value of <b>cSubmitAs</b> is <b>PDF</b> or <b>bEmbedForm</b> is <b>true</b> . This instance ID is the second entry in the <b>doc.docID</b> array ( <b>doc.docID[1]</b> ). Does not affect the current document.
<b>cUsageRights</b>	(optional, version 7.0) Specifies the additional usage rights to be applied to the PDF that is submitted if either the value of <b>cSubmitAs</b> is <b>PDF</b> or <b>bEmbedForm</b> is <b>true</b> . The only valid value is <b>submitFormUsageRights.RMA</b> . Does not affect the current document.

**Returns**

Nothing

**Example 1**

Submit the form to the server.

```
this.submitForm("http://myserver/cgi-bin/myscript.cgi#FDF");
```

**Example 2**

```
var aSubmitFields = new Array( "name", "id", "score" );
this.submitForm({
    cURL: "http://myserver/cgi-bin/myscript.cgi#FDF",
    aFields: aSubmitFields,
    cSubmitAs: "FDF" // the default, not needed here
});
```

**Example 3**

This example illustrates a shortcut to submitting a whole subtree. Passing "name" as part of the **field** parameter, submits "name.title", "name.first", "name.middle" and "name.last".

```
this.submitForm("http://myserver/cgi-bin/myscript.cgi#FDF",
    true, false, "name");
```

**Example 4**

```
this.submitForm({
    cURL: "http://myserver/cgi-bin/myscript.cgi#FDF",
    cSubmitAs: "XFDF"
});
```

**Example 5 (Version 7.0)**

A PDF file contains several XFA forms as attachments, the following script gathers the XML data from each attachment and concatenates them. The combined data is then submitted.

```
var oParent = event.target;
var oDataObjects = oParent.dataObjects;
if (oDataObjects == null)
    app.alert("This form has no attachments!");
else {
    var nChildren = oDataObjects.length;
    var oFirstChild = oParent.openDataObject(oDataObjects[0].name);
    var oSubmitData = oFirstChild.xfa.data.nodes.item(0).clone(true);
    for (var iChild = 1; iChild < nChildren; iChild++) {
        var oNextChild = oParent.openDataObject(
            oDataObjects[iChild].name);
        oSubmitData.nodes.append(oNextChild.xfa.data.nodes.item(0));
        oNextChild.closeDoc();
    }
    oParent.submitForm({
        cURL: "http://www.myCom.com/cgi-bin/myCGI.pl#FDF",
        cSubmitAs: "XML",
        oXML: oSubmitData
    });
    oFirstChild.closeDoc();
}
```

This example uses `doc.dataObjects`, `doc.openDataObject` and properties and method of the `XFAObject Object`.

**Example 6 (Version 7.0)**

This script illustrates `cPermID`, `cInstID` and `cUsageRights`.

```
this.submitForm({
    cURL: myURL,
    cSubmitAs: "PDF",
    cPermID: someDoc.docID[0],
    cInstID: someDoc.docID[1],
    cUsageRights: submitFormUsageRights.RMA
});
```

**syncAnnotScan**

5.0			<b>A</b>	<b>X</b>
-----	--	--	----------	----------

Guarantees that all annotations will be scanned by the time this method returns.

In order to show or process annotations for the entire document, all annotations must have been detected. Normally, a background task runs that examines every page and looks for annotations during idle time, as this scan is a time consuming task. Much of the annotation

behavior works gracefully even when the full list of annotations is not yet acquired by background scanning.

In general, you should call this method if you want the entire list of annotations.

See also [getAnnots](#).

### Parameters

None

### Returns

Nothing

### Example

The second line of code will not be executed until **syncAnnotScan** returns and this will not occur until the annot scan of the document is completed.

```
this.syncAnnotScan();
annots = this.getAnnots({nSortBy:ANSE_Author});
// now, do something with the annotations.
```

---

## Doc.media Object

The **doc.media** of each document contains multimedia properties that are specific to that document, and methods that apply to the document.

---

## Doc.media Object Properties

### canPlay

6.0				
-----	--	--	--	--

The **doc.media.canPlay** property indicates whether multimedia playback is allowed for a document. Playback depends on the user's Trust Manager preferences and other factors. For example, playback is not allowed in authoring mode.

**doc.media.canPlay** returns an object that contains both a yes/no indication and a reason why playback is not allowed, if that is the case.

*Type: Object*

*Access: R.*

If playback is allowed, then **canPlay.yes** exists to indicate this. (It is an empty object, but it may contain other information in the future.) You can make a simple test like this:

```

if( doc.media.canPlay.yes )
{
    // We can play back multimedia for this document
}

```

If playback is not allowed, **canPlay.no** object exists instead. As with **canPlay.yes**, you can simply test for the existence of **canPlay.no**, or you can look inside it for information about why playback is not allowed. At least one of these properties or other properties that may be added in the future will exist within **canPlay.no**:

Properties of canPlay.no	
Property	Description
authoring	can't play when in authoring mode
closing	can't play because the document is closing
saving	can't play because the document is saving
security	can't play because of security settings
other	can't play for some other reason

In addition, **canPlay.canShowUI** indicates whether any alert boxes or other user interface are allowed in response to this particular playback rejection.

#### Example:

```

var canPlay = doc.media.canPlay;
if( canPlay.no )
{
    // We can't play, why not?
    if( canPlay.no.security )
    {
        // The user's security settings prohibit playback,
        // are we allowed to put up alerts right now?
        if( canPlay.canShowUI )
            app.alert( "Security prohibits playback" );
        else
            console.println( "Security prohibits playback" );
    }
    else
    {
        // Can't play for some other reason, handle it here
    }
}

```

## Doc.media Object Methods

### deleteRendition

6.0				
-----	--	--	--	--

The `doc.media.deleteRendition()` method deletes the named Rendition from the document. The Rendition is no longer accessible with JavaScript. It does nothing if the Rendition is not present.

#### Parameters

<b>cName</b>	<b>cName</b> , a string, is the name of the Rendition.
--------------	--

#### Returns

Nothing

#### Example

```

this.media.deleteRendition("myMedia");
if ( this.media.getRendition("myMedia") == null)
    console.println( "Rendition successfully deleted" );
    
```

### getAnnot

6.0				
-----	--	--	--	--

`Doc.media.getAnnot()` looks for and returns a [ScreenAnnot Object](#) in the document by page number and either name or title, or returns `null` if there is no matching ScreenAnnot. If both name and title are specified, both must match.

#### Parameters

<b>args</b>	An object containing the properties to be passed to this method. The properties are described below.
-------------	--

#### Properties of args

<b>nPage</b>	The page number (base 0) on which the Annot resides
<b>cAnnotName</b>	(optional) The name of the ScreenAnnot. <b>NOTE:</b> <b>cAnnotName</b> is never used in pdf generated by Acrobat.
<b>cAnnotTitle</b>	(optional) The title of the ScreenAnnot

**NOTE:** The parameters for this method must be passed as an object literal, and not as an ordered listing of parameters.

## Returns

[ScreenAnnot Object](#)

## Example

The Acrobat user interface allows you to specify the title for a [ScreenAnnot](#) but not its name, so a typical use of `getAnnot` would be:

```
var annot= myDoc.media.getAnnot
    ({ nPage: 0,cAnnotTitle: "My Annot Title" });
```

See the example following [getRendition\(\)](#) for an additional example.

## getAnnots

6.0				
-----	--	--	--	--

The `doc.media.getAnnots()` method returns an Array of all the [ScreenAnnot Objects](#) on the specified page of the document, or all the [ScreenAnnot Objects](#) on all pages of the document if `nPage` is omitted. The array is empty if there are no such [ScreenAnnots](#).

## Parameters

<code>nPage</code>	The page number (base 0) on which the Annots reside
--------------------	---

## Returns

Array of [ScreenAnnot Objects](#)

## Example

Get a listing of the [ScreenAnnots](#) on page 0, then play a media clip in a [ScreenAnnot](#) randomly chosen from the list.

```
var annots = this.media.getAnnots({ nPage: 0 });
var rendition = this.media.getRendition("myClip");
var settings = { windowType: app.media.windowType.docked }
var l = annots.length
var i = Math.floor( Math.random() * l ) % l
var args = { rendition:rendition, annot:annots[i], settings:settings };
app.media.openPlayer( args );
```

## getOpenPlayers

7.0				
-----	--	--	--	--

This method returns an array of [MediaPlayer Objects](#), one for each currently open media player. The players in the array are listed in the order in which they were opened. Using this array, some or all of the open players can be manipulated. For example, you can stop or close all players that the document has opened, without having to keep a list of them yourself.

Each time **getOpenPlayers** is called, it returns a new copy of the array, listing the players open at that moment. New players that are subsequently opened don't show up in an array already gotten. If a player that is in the array is closed, the player object remains in the array and **player.isOpen** becomes **false**. The **doc.media.getOpenPlayers()** can be called again at any time to get a new, up to date player array.

Do not write code that iterates directly over **doc.media.getOpenPlayers**:

```
for( var i in doc.media.getOpenPlayers() ) // Wrong!
```

Instead, get a copy of the player array and iterate over that:

```
var players = doc.media.getOpenPlayers();
for( var i in players ) {
    ....
}
```

This insures that the loop works correctly even if players are opened or closed during the loop.

### Parameters

None

### Returns

Array of [MediaPlayer Objects](#).

### Example

The following two functions take a doc object as a parameter and operate on the running players associated with that doc object.

```
// Stop all running players.
function stopAllPlayers( doc ) {
    var players = doc.media.getOpenPlayers();
    for( var i in players ) players[i].stop();
}
// Close all running players. Closing a player does not remove it from
// the array.
function closeAllPlayers( doc ) {
    var players = doc.media.getOpenPlayers();
    for( var i in players )
        players[i].close( app.media.closeReason.general );
}
```



## getRendition

6.0				
-----	--	--	--	--

`doc.media.getRendition()` looks up a Rendition in the document by name and returns it, or returns `null` if there is no Rendition with that name.

### Parameters

<b>cName</b>	<b>cName</b> , a string, is the name of the Rendition.
--------------	--

### Returns

[Rendition Object](#)

### Example

The following script is executed from a mouse up action of a form button. It plays a docked media clip in a ScreenAnnot.

```
app.media.openPlayer({
  rendition: this.media.getRendition( "myClip" ),
  annot: this.media.getAnnot( {nPage:0,cAnnotTitle:"myScreen"} ),
  settings: { windowType: app.media.windowType.docked }
});
```

## newPlayer

6.0				
-----	--	--	--	--

The `doc.media.newPlayer()` method creates and returns a [MediaPlayer Object](#). The `args` parameter must contain a `settings` property and optionally can contain an `events` property. It can also contain any number of additional user-defined properties. All the properties of `args` are copied into the new [MediaPlayer Object](#). This is a shallow copy: The properties of `args` are copied into the new player, but if any of those properties are objects themselves, those objects are shared between `args` and the new player.

The `newPlayer()` method creates a bare-bones player which does not have any of the standard event listeners required for standard Acrobat media player behavior. Use `app.media.addStockEvents()` to add the necessary event listeners.

In most cases it is better to use `app.media.createPlayer()` to create a media player instead of `doc.media.newPlayer()`. The `createPlayer()` sets up the standard event listeners and other player properties automatically. If you do call `newPlayer()` directly, the source code for `createPlayer()` in `media.js` should be reviewed for sample code.

**Parameters**


---

<b>args</b>	<b>args</b> is a PlayerArgs object. See <a href="#">PlayerArgs Object</a> .
-------------	---

---

**Returns**

[MediaPlayer Object](#)

**Example:**

See `Events.dispatch()` for a rough example.

**Error Objects**

**Error** objects are dynamically created whenever an exception is thrown from methods or properties implemented in Acrobat JavaScript. Several sub-classes of the **Error** object can be thrown by core JavaScript (**EvalError**, **RangeError**, **SyntaxError**, **TypeError**, **ReferenceError**, **URIError**). They all have the **Error** object as prototype. Acrobat JavaScript can throw some of these exceptions, or implement subclasses of the **Error** object at its convenience. If your scripts are using the mechanism of **try/catch** error handling, the object thrown should be one of the types listed in the following table.

<b>Error Object</b>	<b>Brief Description</b>
<b>RangeError</b>	Argument value is out of valid range
<b>TypeError</b>	Wrong type of argument value
<b>ReferenceError</b>	Reading a variable that does not exist
<b>MissingArgError</b>	Missing required argument
<b>NumberOfArgsError</b>	Invalid number of arguments to a method
<b>InvalidSetError</b>	A property set is not valid or possible
<b>InvalidGetError</b>	A property get is not valid or possible
<b>OutOfMemoryError</b>	Out of memory condition occurred
<b>NotSupportedError</b>	Functionality not supported in this configuration (for example, Reader)
<b>NotSupportedHFTError</b>	HFT is not available (a plug-in may be missing)
<b>NotAllowedError</b>	Method or property is not allowed for security reasons
<b>GeneralError</b>	Unspecified error cause
<b>RaiseError</b>	Acrobat internal error

Error Object	Brief Description
<b>DeadObjectError</b>	Object is dead
<b>HelpError</b>	User requested for help with a method

**Error** object types implemented by Acrobat JavaScript inherit properties and methods from the core **Error** object. Some Acrobat Javascript objects may implement their own specific types of exception. A description of the **Error** subclass (with added methods and properties, if any) should be provided in the documentation for the particular object.

### Example

Print all properties of the **Error** object to the console.

```
try {
    app.alert(); // one argument is required for alert
} catch(e) {
    for (var i in e)
        console.println( i + ": " + e[i])
}
```

## Error Properties

### fileName

6.0			
-----	--	--	--

The name of the script which caused the exception to be thrown.

*Type: String*

*Access: R.*

### lineNumber

6.0			
-----	--	--	--

The offending line number from where an exception was thrown in the JavaScript code.

*Type: Integer*

*Access: R.*

### extMessage

7.0			
-----	--	--	--

An message providing additional details about the exception.

*Type: String**Access: R.*

## message

6.0			
-----	--	--	--

The error message providing details about the exception.

*Type: String**Access: R.*

## name

6.0			
-----	--	--	--

The name of the **Error** object subclass, indicating the type of the **Error** object instance.

*Type: String**Access: R/W.*

---

## Error Methods

### toString

6.0			
-----	--	--	--

Gets the error message providing details about the exception.

#### Parameters

None

#### Returns

The error message string. (See [message](#).)

---

## Event Object

All JavaScripts are executed as the result of a particular event. Each event has a *type* and a *name*. The events detailed here are listed as type/name pairs.

For each of these events, Acrobat JavaScript creates an *event object*. During the occurrence of each event, you can access this event object to get, and possibly manipulate, information about the current state of the event.

It is important for JavaScript writers to know when these events occur and in what order they are processed. Some methods or properties can only be accessed during certain events; therefore, a knowledge of these events will prove useful.

## Event Type/Name Combinations

### App/Init

When the Viewer is started, the *Application Initialization Event* occurs. Script files, called **Folder Level JavaScripts**, are read in from the application and user JavaScript folders. They load in the following order: `config.js`, `glob.js`, all other files, then any user files.

This event defines the **name** and **type** properties for the **event** object.

This event does not listen to the **rc** return code.

### Batch/Exec

5.0			
-----	--	--	--

A batch event occurs during the processing of each document of a batch sequence. JavaScripts that authored as part of a batch sequence can access the event object upon execution.

This event defines the **name**, **target**, and **type** properties for the **event** object. The **target** in this event is the document object.

This event listens to the **rc** return code. If the return code is set to **false**, the batch sequence is stopped.

### Bookmark/Mouse Up

5.0			
-----	--	--	--

This event occurs whenever a user clicks on a bookmark that executes a JavaScript.

This event defines the **name**, **target**, and **type** properties for the event object. The **target** in this event is the bookmark object that was clicked.

This event does not listen to the **rc** return code.

### Console/Exec

5.0			
-----	--	--	--

A console event occurs whenever a user evaluates a JavaScript in the console.

This event defines the **name**, and **type** properties for the event object.

This event does not listen to the **rc** return code.

**Doc/DidPrint**

5.0			
-----	--	--	--

This event is triggered after a document has printed.

This event defines the **name**, **target**, and **type** properties for the event object. The **target** in this event is the document object.

This event does not listen to the **rc** return code.

**Doc/DidSave**

5.0			
-----	--	--	--

This event is triggered after a document has been saved.

This event defines the **name**, **target**, and **type** properties for the event object. The **target** in this event is the document object.

This event does not listen to the **rc** return code.

**Doc/Open**

4.0			
-----	--	--	--

This event is triggered whenever a document is opened. When a document is opened, the document level script functions are scanned and any exposed scripts are executed.

This event defines the **name**, **target**, **targetName**, and **type** properties for the event object. The **target** in this event is the document object.

This event does not listen to the **rc** return code.

**Doc/WillClose**

5.0			
-----	--	--	--

This event is triggered before a document is closed.

This event defines the **name**, **target**, and **type** properties for the event object. The **target** in this event is the document object.

This event does not listen to the **rc** return code.

**Doc/WillPrint**

5.0			
-----	--	--	--

This event is triggered before a document is printed.

This event defines the **name**, **target**, and **type** properties for the event object. The **target** in this event is the document object.

This event does not listen to the **rc** return code.

**Doc/WillSave**

5.0			
-----	--	--	--

This event is triggered before a document is saved.

This event defines the **name**, **target**, and **type** properties for the event object. The **target** in this event is the document object.

This event does not listen to the **rc** return code.

**External/Exec**

5.0			
-----	--	--	--

This event is the result of an external access, for example, through OLE, AppleScript, or loading an FDF.

This event defines the **name** and **type** properties for the event object.

This event does not listen to the **rc** return code.

**Field/Blur**

4.05			
------	--	--	--

The **blur** event occurs after all other events just as the field loses focus. This event is generated regardless of whether or not a mouse click is used to deactivate the field (for example, tab key).

This event defines the **modifier**, **name**, **shift**, **target**, **targetName**, **type**, and **value** properties for the event object. The **target** in this event is the field whose validation script is being executed.

This event does not listen to the **rc** return code.

**Field/Calculate**

3.01			
------	--	--	--

This event is defined when a change in a form requires that all fields that have a calculation script attached to them be executed. All fields that depend on the value of the changed field will now be re-calculated. These fields may in turn generate additional **Field/Validate**, **Field/Blur**, and **Field/Focus** events.

Calculated fields may have dependencies on other calculated fields whose values must be determined beforehand. The **calculation order array** contains an ordered list of all the fields in a document that have a calculation script attached. When a full calculation is needed, each of the fields in the array is calculated in turn starting with the zeroth index of the array and continuing in sequence to the end of the array.

To change the calculation order of fields, use the **Advanced>Forms>Set Field Calculation Order...** menu item in Adobe Acrobat.

This event defines the **name**, **source**, **target**, **targetName**, **type**, and **value** properties for the event object. The **target** in this event is the field whose calculation script is being executed.

This event does listen to the **rc** return code. If the return code is set to **false**, the field's value is not changed. If true, the field takes on the value found in the **value**.

### Field/Focus

4.05			
------	--	--	--

The **focus** event occurs after the mouse down but before the mouse up after the field gains the focus. This routine is called whether or not a mouse click is used to activate the field (for example, tab key) and is the best place to perform processing that must be done before the user can interact with the field.

This event defines the **modifier**, **name**, **shift**, **target**, **targetName**, **type**, and **value** properties for the event object. The **target** in this event is the field whose validation script is being executed.

This event does not listen to the **rc** return code.

### Field/Format

3.01			
------	--	--	--

Once all dependent calculations have been performed the **format** event is triggered. This event allows the attached JavaScript to change the way that the data value appears to a user (also known as its presentation or appearance). For example, if a data value is a number and the context in which it should be displayed is currency, the formatting script can add a dollar sign (\$) to the front of the value and limit it to two decimal places past the decimal point.

This event defines the **commitKey**, **name**, **target**, **targetName**, **type**, **value**, and **willCommit** properties for the event object. The **target** in this event is the field whose format script is being executed.

This event does not listen to the **rc** return code. However, the resulting **value** is used as the fields formatted appearance.

### Field/Keystroke

3.01			
------	--	--	--

The **keystroke** event occurs whenever a user types a keystroke into a **text box** or **combobox** (this includes cut and paste operations), or selects an item in a **combobox** drop down or **listbox** field. A keystroke script may want to limit the type of keys allowed. For example, a numeric field might only allow numeric characters.

The user interface for Acrobat allows the author to specify a **Selection Change** script for listboxes. The script is triggered every time an item is selected. This is implemented as the keystroke event where the keystroke value is equivalent to the user selection. This behavior



is also implemented for the combobox—the "keystroke" could be thought to be a paste into the text field of the value selected from the drop down list.

There is a final call to the keystroke script before the validate event is triggered. This call sets the `willCommit` to `true` for the event. With keystroke processing, it is sometimes useful to make a final check on the field value before it is committed (pre-commit). This allows the script writer to gracefully handle particularly complex formats that can only be partially checked on a keystroke by keystroke basis.

The `keystroke` event of text fields is called in situations other than when the user is entering text with the keyboard or committing the field value. It is also called to validate the default value of a field when set through the UI or by JavaScript, and to validate entries provided by autofill. In these situations not all properties of the event are defined. Specifically `event.target` will be `undefined` when validating default values and `event.richChange` and `event.richValue` will be `undefined` when validating autofill entries.

This event defines the `commitKey`, `change`, `changeEx`, `keyDown`, `modifier`, `name`, `selEnd`, `selStart`, `shift`, `target` (except when validating default values), `targetName`, `type`, `value`, and `willCommit` properties for the event object. The `target` in this event is the field whose keystroke script is being executed.

This event does listen to the `rc` return code. If set to `false`, the keystroke is ignored. The resulting `change` is used as the keystroke if the script desires to replace the keystroke code. The resultant `selEnd` and `selStart` properties can change the current text selection in the field.

### Field/Mouse Down

3.01			
------	--	--	--

The `mouse down` event is triggered when a user starts to click on a form field and the mouse button is still down. It is advised that you perform very little processing (that is, play a short sound) during this event. A mouse down event will not occur unless a `mouse enter` event has already occurred.

This event defines the `modifier`, `name`, `shift`, `target`, `targetName`, and `type` properties for the event object. The `target` in this event is the field whose validation script is being executed.

This event does not listen to the `rc` return code.

### Field/Mouse Enter

3.01			
------	--	--	--

The `mouse enter` event is triggered when a user moves the mouse pointer inside the rectangle of a field. This is the typical place to open a text field to display help text, and so on.

This event defines the `modifier`, `name`, `shift`, `target`, `targetName`, and `type` properties for the event object. The `target` in this event is the field whose validation script is being executed.

This event does not listen to the **rc** return code.

### Field/Mouse Exit

3.01			
------	--	--	--

The **mouse exit** event is the opposite of the **mouse enter** event and occurs when a user moves the mouse pointer outside of the rectangle of a field. A **mouse exit** event will not occur unless a **mouse enter** event has already occurred.

This event defines the **modifier**, **name**, **shift**, **target**, **targetName**, and **type** properties for the event object. The **target** in this event is the field whose validation script is being executed.

This event does not listen to the **rc** return code.

### Field/Mouse Up

3.01			
------	--	--	--

The **mouse up** event is triggered when the user clicks on a form field and releases the mouse button. This is the typical place to attach routines such as the submit action of a form. A **mouse up** event will not occur unless a **mouse down** event has already occurred.

This event defines the **modifier**, **name**, **shift**, **target**, **targetName**, and **type** properties for the event object. The **target** in this event is the field whose validation script is being executed.

This event does not listen to the **rc** return code.

### Field/Validate

3.01			
------	--	--	--

Regardless of the field type, user interaction with a field may produce a new value for that field. After the user has either clicked outside a field, tabbed to another field, or pressed the enter key, the user is said to have **committed** the new data value.

The **validate** event is the first event generated for a field after the value has been committed so that a JavaScript can verify that the value entered was correct. If the validate event is successful, the next event triggered is the **calculate** event.

This event defines the **change**, **changeEx**, **keyDown**, **modifier**, **name**, **shift**, **target**, **targetName**, **type**, and **value** properties for the event object. The **target** in this event is the field whose validation script is being executed.

This event does listen to the **rc** return code. If the return code is set to **false**, the field value is considered to be invalid and the value of the field is unchanged.

### Link/Mouse Up

5.0			
-----	--	--	--

This event is triggered when a link containing a JavaScript action is activated by the user.

This event defines the **name**, **target**, and **type** properties for the event object. The **target** in this event is the document object.

This event does not listen to the **rc** return code.

**Menu/Exec**

5.0			
-----	--	--	--

A menu event occurs whenever JavaScript that has been attached to a menu item is executed. In Acrobat 5.0, the user can add a menu item and associate JavaScript actions with it. For example,

```
app.addItem({ cName: "Hello", cParent: "File",
              cExec: "app.alert('Hello',3);", nPos: 0});
```

The script **app.alert('Hello', 3)** will execute during a **menu event**. There are two ways for this to occur:

1. Through the user interface, the user can click on that menu item and the script will execute; and
2. Programmatically, when **app.execMenuItem("Hello")** is executed (perhaps, during a mouse up event of a button field), the script will execute.

This event defines the **name**, **target**, **targetName**, and **type** properties for the event object. The **target** in this event is the currently active document, if one is open.

This event listens to the **rc** return code in the case of the enable and marked proc for menu items. (See the **cEnabled** and **cMarked** parameters of **app.addItem()**.) A return code of **false** will disable or unmark a menu item. A return code of **true** enable or mark a menu item.

**Page/Open**

4.05			
------	--	--	--

This event happens whenever a new page is viewed by the user and after page drawing for the page has occurred.

This event defines the **name**, **target**, and **type** properties for the event object. The **target** in this event is the document object.

This event does not listen to the **rc** return code.

**Page/Close**

4.05			
------	--	--	--

This event happens whenever the page being viewed is no longer the current page; that is, the user switched to a new page or closed the document.

This event defines the **name**, **target**, and **type** properties for the event object. The **target** in this event is the document object.

This event does not listen to the **rc** return code.

**Screen/InView**

6.0			
-----	--	--	--

This event happens whenever a new page first comes into view by the user. When the page layout is set to “Continuous” or “Continuous - Facing”, this event occurs before the Screen/Open event.

This event defines the [modifier](#), [name](#), [shift](#), [target](#), [targetName](#), and [type](#) properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the [rc](#) return code.

**Screen/OutView**

6.0			
-----	--	--	--

This event happens whenever a page first goes out of view from the user. When the page layout is set to “Continuous” or “Continuous - Facing”, this event occurs after the Screen/Close event.

This event defines the [modifier](#), [name](#), [shift](#), [target](#), [targetName](#), and [type](#) properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the [rc](#) return code.

**Screen/Open**

6.0			
-----	--	--	--

This event happens whenever a new page is viewed by the user and after page drawing for the page has occurred.

This event defines the [modifier](#), [name](#), [shift](#), [target](#), [targetName](#), and [type](#) properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the [rc](#) return code.

**Screen/Close**

6.0			
-----	--	--	--

This event happens whenever the page being viewed is no longer the current page; that is, the user switched to a new page or closed the document.

This event defines the [modifier](#), [name](#), [shift](#), [target](#), [targetName](#), and [type](#) properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the [rc](#) return code.

**Screen/Focus**

6.0			
-----	--	--	--

The **focus** event occurs after the mouse down but before the mouse up after the field gains the focus. This routine is called whether or not a mouse click is used to activate the ScreenAnnot (for example, tab key) and is the best place to perform processing that must be done before the user can interact with the field.

This event defines the [modifier](#), [name](#), [shift](#), [target](#), [targetName](#), and [type](#) properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the [rc](#) return code.

**Screen/Blur**

6.0			
-----	--	--	--

The **blur** event occurs after all other events just as the ScreenAnnot loses focus. This event is generated regardless of whether or not a mouse click is used to deactivate the ScreenAnnot (for example, tab key).

This event defines the [modifier](#), [name](#), [shift](#), [target](#), [targetName](#), and [type](#) properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the [rc](#) return code.

**Screen/Mouse Up**

6.0			
-----	--	--	--

The **mouse up** event is triggered when the user clicks on a ScreenAnnot and releases the mouse button. This is the typical place to attach routines such as the starting a Multimedia clip. A **mouse up** event will not occur unless a **mouse down** event has already occurred.

This event defines the [modifier](#), [name](#), [shift](#), [target](#), [targetName](#), and [type](#) properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the [rc](#) return code.

**Screen/Mouse Down**

6.0			
-----	--	--	--

The **mouse down** event is triggered when a user starts to click on a ScreenAnnot and the mouse button is still down. It is advised that you perform very little processing (that is, play a short sound) during this event. A mouse down event will not occur unless a **mouse enter** event has already occurred.

This event defines the **modifier**, **name**, **shift**, **target**, **targetName**, and **type** properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the **rc** return code.

### Screen/Mouse Enter

6.0			
-----	--	--	--

The **mouse enter** event is triggered when a user moves the mouse pointer inside the rectangle of an ScreenAnnot.

This event defines the **modifier**, **name**, **shift**, **target**, **targetName**, and **type** properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the **rc** return code.

### Screen/Mouse Exit

6.0			
-----	--	--	--

The **mouse exit** event is the opposite of the **mouse enter** event and occurs when a user moves the mouse pointer outside of the rectangle of a ScreenAnnot. A **mouse exit** event will not occur unless a **mouse enter** event has already occurred.

This event defines the **modifier**, **name**, **shift**, **target**, **targetName**, and **type** properties for the event object. The **target** in this event is the ScreenAnnot (see [ScreenAnnot Object](#)) that initiated this event, **targetName** is the title of the ScreenAnnot.

This event does not listen to the **rc** return code.

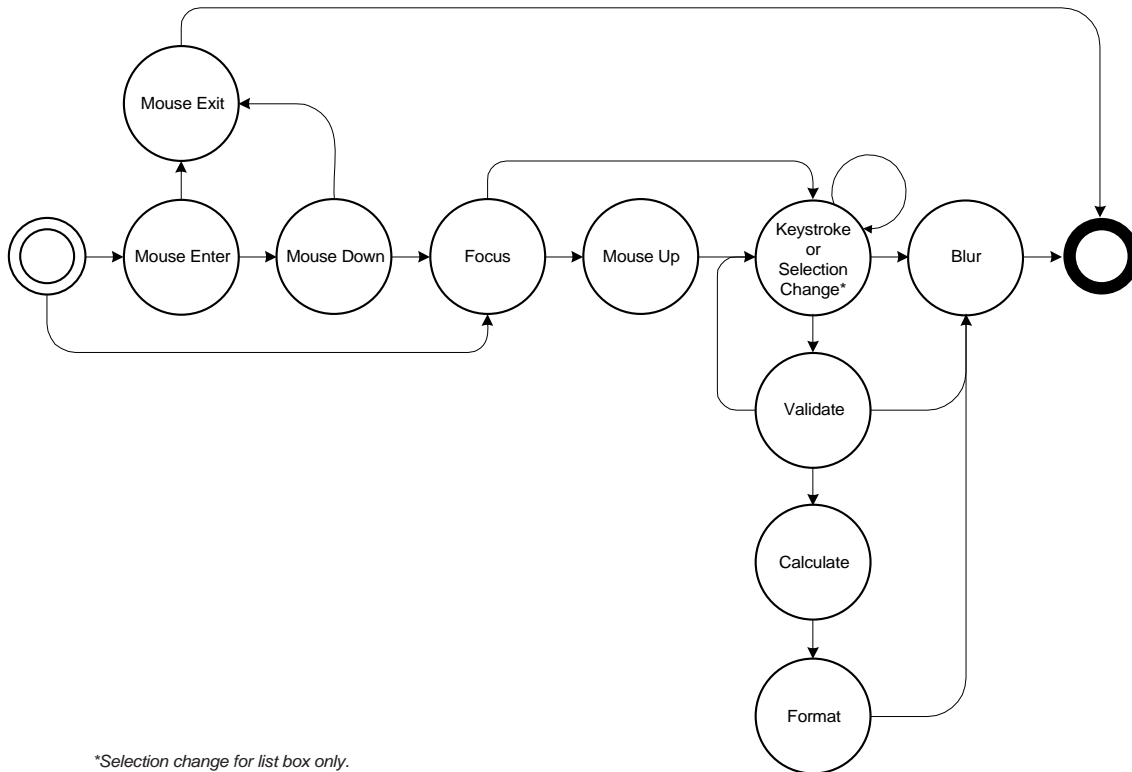
## Document Event Processing

When a document is opened, the [Doc/Open](#) event occurs; functions are scanned, and any exposed (top-level) scripts are executed. Next, if the **NeedAppearances** key in the PDF file is set to **true** in the **AcroForm** dictionary, the formatting scripts of all form fields in the document are executed. (See Section 3.6.1 and 7.6.1 of the [PDF Reference](#).) Finally, the [Page/Close](#) event occurs.

**NOTE:** For user's who create PDF files containing form fields with the **NeedAppearances** key set to **true**, be sure to do a "Save As" before posting such files on the Web. Performing a "Save As" on a file generates the form appearances, which are saved with the file. This increases the performance of Reader when it loads the file within a Web browser.

## Form Event Processing

The order in which the form events occur is illustrated in the state diagram below. This illustrates certain dependencies that are worth noting, for example, the Mouse Up event cannot occur if the Focus event did not occur.



\*Selection change for list box only.

## Multimedia Event Processing

Whenever an event fires and is dispatched to an event listener, a (multimedia) [Event Object](#) is passed as a parameter to the event listener. This object is similar to the event object used elsewhere in Acrobat, and it has the properties listed below.

Multimedia Event objects fired by rendition actions (e.g. in custom JavaScript entered from the Actions tab in the Multimedia Properties panel) also include these properties:

<code>action.annot</code>	The Screen Annotation for this event (See <a href="#">ScreenAnnot Object</a> )
<code>action.rendition</code>	The Rendition for this event (See <a href="#">Rendition Object</a> )

Multimedia Event objects that have been dispatched by the standard multimedia event dispatcher also include these properties. These are not present if you provide your own `events.dispatch()` method:

<code>media.doc</code>	The document, same as <code>target.doc</code>
<code>media.events</code>	The events object, same as <code>target.events</code>
<code>media.id</code>	A copy of <code>event.name</code> with spaces removed

Individual events may have additional properties; see the description of each [EventListener Object](#) method for details.

An event method called by the standard event dispatcher may set either of these properties to stop further event dispatching:

```
stopDispatch
stopAllDispatch
```

To stop the current event from being dispatched to any remaining event listeners, an event method can set `event.stopDispatch` to `true`. If this is done in an “on” event method, no more “on” methods will be called for the event, but “after” methods will still be called. If you set `event.stopAllDispatch`, then no more event methods of either type will be called. Read about the [EventListener Object](#) for a description of the “on” and “after” event listeners

---

## Event Properties

### change

3.01			
------	--	--	--

Specifies the change in value that the user has just typed. This is replaceable such that if the JavaScript wishes to substitute certain characters, it may. The change may take the form of an individual keystroke or a string of characters (for example if a paste into the field is performed).

*Type: String*

*Access: R/W.*

#### Example

Change all keystrokes to upper case.

```
// Custom Keystroke for text field
event.change = event.change.toUpperCase();
```



## changeEx

5.0			
-----	--	--	--

Contains the export value of the change and is available only during a [Field/Keystroke](#) event for **listbox** and **combobox**.

For the **listbox**, the keystroke script, if any, is entered under the **Selection Change** tab in the properties dialog.

For the **combobox**, **changeEx** is only available if the pop-up part of the combo is used, that is, a selection (with the mouse or the keyboard) is being made from the pop-up. If the combo is editable and the user types in an entry, the [Field/Keystroke](#) event behaves as for a **text** field (that is, there are no **changeEx** or [keyDown](#) event properties).

Beginning with Acrobat 6.0, **event.changeEx** is defined for text fields. When **event.fieldFull** is **true**, **changeEx** is set to the entire text string the user attempted to enter and **event.change** is the text string cropped to what fits within the field. Use **event.richChangeEx** (and **event.richChange**) to handle rich text fields.

Type: various

Access: R.

### Example 1

This example illustrates the **combobox**, **event.changeEx** and **app.launchURL**. The example illustrates a simple html online help file system.

Here is a **combobox**, which is described programmatically.

```
var c = this.addField({
  cName: "myHelp",
  cFieldType: "combobox",
  nPageNum: 0,
  oCoords: [72,12+3*72, 3*72, 0+3*72]
});
```

Now set the items in the **combobox**.

```
c.setItems([
  ["Online Help", "http://www.myhelp.com/myhelp.html"],
  ["How to Print", "http://www.myhelp.com/myhelp.html#print"],
  ["How to eMail", "http://www.myhelp.com/myhelp.html#email"]
]);
```

Set the action.

```
c.setAction("Keystroke", "getHelp()");
```

This function is a defined at the document level.

```
function getHelp() {
  if ( !event.willCommit && (event.changeEx != "") )
    app.launchURL(event.changeEx);
}
```

**Example 2**

For an example of the use of **changeEx** with text fields, see the example following **fieldFull**.

**commitKey**

4.0			
-----	--	--	--

Determines how a form field will lose focus. Values are:

- 0 : Value was not committed (for example, escape key was pressed).
- 1: Value was committed because of a click outside the field using the mouse.
- 2: Value was committed because of hitting the enter key.
- 3: Value was committed by tabbing to a new field.

Type: Number

Access: R.

**Example**

To automatically display an alert dialog after a field has been committed add the following to the field's format script:

```
if (event.commitKey != 0)
    app.alert("Thank you for your new field value.");
```

**fieldFull**

6.0			
-----	--	--	--

Only available in keystroke events for text fields. Set to **true** when the user attempts to enter text which does not fit in the field due to either a space limitation (the property **Field.doNotScroll** is set to **true**) or the maximum character limit (the property **Field.charLimit** set to a positive value). When **fieldFull** is **true**, **event.changeEx** is set to the entire text string the user attempted to enter and **event.change** is the text string cropped to what fits within the field.

Type: Boolean

Access: R

Events: **Keystroke**.

**Example 1**

Below is custom keystroke script for a text field that has a character limit, for example. Then the field gets filled, or if the user commits the data entered, the focus moves to another field.

```
if ( event.fieldFull || event.willCommit )
    this.getField("NextTabField").setFocus();
```

**Example 2**

Test whether user has overfilled the text field. Custom Keystroke script for a text field. Initially, the field is set so that text does not scroll.

```
if ( event.fieldFull )
{
  app.alert("You've filled the given space with text,"
+ " and as a result, you've lost some text. I'll set the field to"
+ " scroll horizontally, and paste in the rest of your"
+ " missing text.");
  this.resetForm([event.target.name]); // reset field to lose focus
  event.target.doNotScroll = false; // make changes
  event.change = event.changeEx;
}
```

Field properties generally cannot be changed during a keystroke event, so it is necessary for the field to lose focus as a way to commit the data. The user then has to reset the focus and continue entering data.

**keyDown**

5.0			
-----	--	--	--

Available only during a keystroke event for **listbox** and **combobox**. For a **listbox** or the pop-up part of a **combobox**, the value is **true** if the arrow keys were used to make a selection, **false** otherwise.

For the **combobox**, **keyDown** is only available if the pop-up part of it is used, that is, a selection (with the mouse or the keyboard) is being made from the pop-up. If the combo is editable and the user types in an entry, the **Field/Keystroke** event behaves as for a **text** field (that is, there are no **changeEx** or **keyDown** event properties).

Type: Boolean

Access: R.

**modifier**

3.01			
------	--	--	--

Whether the modifier key is down during a particular event. The modifier key on the Microsoft Windows platform is **Control** and on the Macintosh platform is **Option** or **Command**. The **modifier** is not supported on UNIX.

Type: Boolean

Access: R.

**name**

4.05			
------	--	--	--

The name of the current event as a text string. The **type** and name together uniquely identify the event. Valid names are:

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

*Type: String*

*Access: R*

*Events: all.*

**rc**

3.01			
------	--	--	--

Used for validation. Indicates whether a particular event in the event chain should succeed. Set to **false** to prevent a change from occurring or a value from committing. By default **rc** is **true**.

*Type: Boolean*

*Access: R/W*

*Events: Keystroke, Validate, Menu.*

**richChange**

6.0			
-----	--	--	--

Specifies the change in value that the user has just typed. The **richChange** property is only defined for rich text fields and mirrors the behavior of the **event.change** property. The value of **richChange** is an array of **Span Objects** which specify both the text entered into the field and the formatting. Keystrokes are represented as single member arrays, while rich text pasted into a field is represented as an array of arbitrary length.

When **event.fieldFull** is **true**, **richChangeEx** is set to the entire rich formatted text string the user attempted to enter and **event.richChange** is the rich formatted text string cropped to what fits within the field. Use **event.changeEx** (and **event.change**) to handle (plain) text fields.

*Type: Array of Span Objects* *Access: R/W*

*Events: Keystroke.*

Related objects and properties are the [Span Object](#), [field.defaultStyle](#), [field.richText](#), [field.richValue](#), [event.richValue](#), and [annot.richContents](#).

### Example

This example changes the keystroke to upper case, alternately colors the text blue and red, and switches underlining off and on.

```
// Custom Keystroke event for text rich field.
var span = event.richChange;
for ( var i=0; i<span.length; i++)
{
    span[i].text = span[i].text.toUpperCase();
    span[i].underline = !span[i].underline;
    span[i].textColor = (span[i].underline) ? color.blue : color.red;
}
event.richChange = span;
```

## richChangeEx

6.0			
-----	--	--	--

The **richChangeEx** property is only defined for rich text fields and mirrors the behavior of the [event.changeEx](#) property for text fields. The value of **richChangeEx** is an array of [Span Objects](#) which specify both the text entered into the field and the formatting. Keystrokes are represented as single member arrays, while rich text pasted into a field is represented as an array of arbitrary length.

When [event.fieldFull](#) is **true**, **richChangeEx** is set to the entire rich formatted text string the user attempted to enter and [event.richChange](#) is the rich formatted text string cropped to what fits within the field. Use [event.changeEx](#) (and [event.change](#)) to handle (plain) text fields.

Type: Array of [Span Objects](#) Access: R/W

Events: **Keystroke**.

Related objects and properties are the [Span Object](#), [field.defaultStyle](#), [field.richText](#), [field.richValue](#), [event.richChange](#), [event.richValue](#), and [annot.richContents](#).

### Example

If the text field is filled up by the user, allow additional text by setting the field to scroll.

```
if ( event.fieldFull )
{
    app.alert("You've filled the given space with text,"
    + " and as a result, you've lost some text. I'll set the field to"
    + " scroll horizontally, and paste in the rest of your"
    + " missing text.");
    this.resetForm([event.target.name]); // reset field to lose focus
    event.target.doNotScroll = false; // make changes
```

```

    if ( event.target.richText )
        event.richChange = event.richChangeEx
    else
        event.change = event.changeEx;
}

```

See also [event.fieldFull](#).

## richValue

6.0			
-----	--	--	--

This property mirrors the [field.richValue](#) property of the field and the [event.value](#) property for each event.

Type: Array of [Span Objects](#) Access: R/W

Events: **Keystroke**.

Related objects and properties are the [Span Object](#), [field.defaultStyle](#), [field.richText](#), [field.richValue](#), [event.richChange](#), [event.richChangeEx](#), and [annot.richContents](#).

### Example

This example turns all bold text into red underlined text.

```

// Custom Format event for a rich text field.
var spans = event.richValue;
for ( var i = 0; i < spans.length; i++ )
{
    if( spans[i].fontWeight >= 700 )
    {
        spans[i].textColor = color.red;
        spans[i].fontWeight = 400; // change to default weight
        spans[i].underline = true;
    }
}
event.richValue = spans;

```

## selEnd

3.01			
------	--	--	--

The ending position of the current text selection during a keystroke event.

Type: *Integer*

Access: R/W.

### Example

This is the function [AFMergeChange](#) taken from the file `AForms.js` in the application JavaScripts folder. This function merges the last change (of a text field) with the uncommitted change. This function uses both [selEnd](#) and [selStart](#).

```
function AFMergeChange(event)
{
    var prefix, postfix;
    var value = event.value;

    if(event.willCommit) return event.value;
    if(event.selStart >= 0)
        prefix = value.substring(0, event.selStart);
    else prefix = "";
    if(event.selEnd >= 0 && event.selEnd <= value.length)
        postfix = value.substring(event.selEnd, value.length);
    else postfix = "";
    return prefix + event.change + postfix;
}
```

## selStart

3.01			
------	--	--	--

The starting position of the current text selection during a keystroke event.

*Type: Integer*

*Access: R/W.*

### Example

See the example following [selEnd](#).

## shift

3.01			
------	--	--	--

Whether the shift key is down during a particular event.

*Type: Boolean*

*Access: R.*

### Example

The following is a mouse up button action.

```
if (event.shift)
    this.gotoNamedDest("dest2");
else
    this.gotoNamedDest("dest1");
```

**source**

5.0			
-----	--	--	--

The [Field Object](#) that triggered the calculation event. This is usually different from the [target](#) of the event, that is, the field that is being calculated.

*Type: object*

*Access: R.*

**target**

3.01			
------	--	--	--

The target object that triggered the event. In all mouse, focus, blur, calculate, validate, and format events it is the [Field Object](#) that triggered the event. In other events, such as page open and close, it is the [Doc Object](#) or [this Object](#).

*Type: object*

*Access: R.*

**targetName**

5.0			
-----	--	--	--

Tries to return the name of the JavaScript being executed. Can be used for debugging purposes to help better identify the code causing exceptions to be thrown. Common values of **targetName** include:

- the folder-level script file name for [App/Init](#) events;
- the Doc-level script name for [Doc/Open](#) events;
- the PDF file name being processed for [Batch/Exec](#) events;
- the Field name for [Field/Blur](#), [Field/Calculate](#), [Field/Focus](#), [Field/Format](#), [Field/Keystroke](#), [Field/Mouse Down](#), [Field/Mouse Enter](#), [Field/Mouse Exit](#), [Field/Mouse Up](#) and [Field/Validate](#) events.
- the Menu item name for [Menu/Exec](#) events.

If there is an identifiable name, Acrobat EScript reports **targetName** when an exception is thrown.

*Type: String*

*Access: R.*

**Example**

The first line of the folder level JavaScript file `conserve.js` has an error in it, when the Acrobat Viewer started, an exception is thrown. The standard message reveals quite clearly the source of the problem.

```
MissingArgError: Missing required argument.
```



```
App.alert:1:Folder-Level:App:conserve.js
==> Parameter cMsg.
```

## type

5.0			
-----	--	--	--

The type of the current event as a text string. The type and [name](#) together uniquely identify the event. Valid types are:

Batch	External
Console	Bookmark
App	Link
Doc	Field
Page	Menu

*Type: String*

*Access: R.*

## value

3.01			
------	--	--	--

This property has different meanings for different **field** events.

### Field/Validate event

For the [Field/Validate](#) event, this is the value that the field contains when it is committed. For a **combobox**, this is the **face value**, not the **export value** (see [changeEx](#) for the export value).

#### Example

For example, the following JavaScript verifies that the field value is between zero and 100.

```
if (event.value < 0 || event.value > 100) {
    app.beep(0);
    app.alert("Invalid value for field " + event.target.name);
    event.rc = false;
}
```

### Field/Calculate event

For a [Field/Calculate](#) event, JavaScript should set this property. It is the value that the field should take upon completion of the event.

#### Example

For example, the following JavaScript sets the calculated value of the field to the value of the SubTotal field plus tax.

```
var f = this.getField("SubTotal");
event.value = f.value * 1.0725;
```

**Field/Format event**

For a **Field/Format** event, JavaScript should set this property. It is the value used when generating the appearance for the field. By default, it contains the value that the user has committed. For a **combobox**, this is the **face value**, not the **export value** (see **changeEx** for the export value).

**Example**

For example, the following JavaScript formats the field as a currency type of field.

```
event.value = util.printf("$%.2f", event.value);
```

**Field/Keystroke event**

The current value of the field. If modifying a text field, for example, this is the text in the text field before the keystroke is applied.

**Field/Blur and Field/Focus events**

The current value of the field. During these two events, **event.value** is read-only, that is, the field value cannot be changed by setting **event.value**.

Beginning with Acrobat 5.0, for a **listbox** that allows multiple selections (see **field.multipleSelection**), if the field value is an array (that is, there are multiple selections currently selected), **event.value** returns an empty string when getting, and does not accept setting.

*Type: various*

*Access: R/W.*

**willCommit**

3.01			
------	--	--	--

Verifies the current keystroke event before the data is committed. This is useful to check the target form field values and for example verify if character data instead of numeric data was entered. JavaScript sets this property to **true** after the last **keystroke** event and before the field is validated.

*Type: Boolean*

*Access: R.*

**Example**

This example illustrates the structure of a keystroke event.

```
var value = event.value
if (event.willCommit)
    // Final value checking.
else
    // Keystroke level checking.
```

## Events Object

A multimedia Events object (whose constructor is `app.media.Events`) is a collection of event listener objects. The events property of a [MediaPlayer Object](#) or a [ScreenAnnot Object](#) is an Events object.

### Example:

This following is executed as rendition action

```
console.println("Ready to play \"" + event.action.rendition.uiName
    + "\" from screen annot \"" + event.targetName + "\".");
// Create a simple app.media.Events object
var events = new app.media.Events({
    // The Event object is passed as a parameter to all event
    // listeners, this is a the parameter "e" below/
    // Called immediately during a Play event:
    onPlay: function( e ) { console.println( "onPlay: media.id = "
        + e.media.id ); },
    // Called during idle time after the Play event:
    afterPlay: function() { console.println( "afterPlay" ); },
});
var player = app.media.openPlayer({ events: events });
```

## Events Object Methods

### add

6.0				
-----	--	--	--	--

Adds any number of [EventListener Objects](#) to the dispatch table for this [Events Object](#). Any previous listeners are preserved, and when an event is fired, all matching listener methods are called.

The standard event dispatcher first calls any [onEveryEvent](#) methods in the order they were added, then calls any "on" events (see the description of "on" and "after" events in the introductory paragraphs to [EventListener Object](#)) for the specific event being dispatched, also in the order they were added. Finally, it sets a very short timer (one millisecond) to call any "after" events. When that timer fires, the "after" events are called in the same order described for on events.

**NOTE:** If you try to add the same event listener twice, the second attempt is ignored.

If you add an event listener from inside an event method, the new listener's methods will be called as part of the dispatching for the current event.

### Parameters

Any number of parameters, each one an [EventListener Object](#)

**Returns**

Nothing

**Example:**

```
// Add an event listener for the onPlay event, here, player is a
// MediaPlayer object.
player.events.add
({
  onPlay: function() { console.println( "onPlay" ); }
});
```

See also Events.[remove](#)..**dispatch**

6.0				
-----	--	--	--	--

When a MediaPlayer fires an event, the Multimedia plug-in creates an [Event Object](#) and calls `MediaPlayer.events.dispatch(event)`. Similarly, a ScreenAnnot calls `ScreenAnnot.events.dispatch(event)`.

The dispatch method is the only part of the event dispatching system that the Acrobat Multimedia plugin calls directly. You can substitute your own, entirely different event dispatching system by providing your own `MediaPlayer.events` object with its own `dispatch()` method.

The `dispatch()` method is responsible for calling each of the event listeners associated with the event, as identified by `oMediaEvent.name`. In most cases, a PDF file will not provide its own `dispatch()` method but will use the standard event dispatching system.

**Parameters**


---

<code>oMediaEvent</code>	A <a href="#">Event Object</a>
--------------------------	--------------------------------

---

**Returns**

Nothing

If you write your own `dispatch()` method, note that `oMediaEvent.name` may contain spaces. The standard `dispatch()` method makes a copy of `oMediaEvent.name` in `oMediaEvent.media.id` with the spaces removed, to allow the name to be used directly as part of a JavaScript event method name.

Also, note that if you write your own `dispatch()`, it will be called synchronously when each event occurs, and any processing you do will be subject to the same limitations as described for “on” event methods in the [EventListener](#) section (see [EventListener Object](#)). In particular, it cannot make any calls to a [MediaPlayer Object](#) nor do anything that can indirectly cause a MediaPlayer method to be called. See the source code for the standard `dispatch()` in `media.js` for a way to work around this using a timer.

The `dispatch()` method is not usually called directly from JavaScript code, although it can be.

**Example:**

```
// Create a new media player with a custom event dispatcher.
// This is an advanced technique that would rarely be used in
// typical PDF JavaScript.
var player = doc.media.newPlayer(
{
  events:
  {
    dispatch: function( e )
    {
      console.println( 'events.dispatch' + e.toSource() );
    }
  }
});
// Synthesize and dispatch a Script event, as if one had been
// encountered while the media was playing. With the standard event
// dispatcher, this will call any and all event listeners that have been
// added for this event. With the custom dispatcher above, it will log a
// message to the console.
var event = new Event;
event.name = "Script";
event.media = { command: "test", param: "value" };
player.events.dispatch( event );
```

**remove**

6.0				
-----	--	--	--	--

The method removes one or more event listeners that were previously added with `Events.add()`. If you use an object literal directly in `Events.add()`, you will not be able to remove that listener using `Media.remove()` because there is no way to pass a reference to the same object. If you want to be able to remove an event listener, pass it to `add()` in a variable instead of an object literal, so that you can pass the same variable to `remove()`, as in the example below.

The `remove()` method may be called from inside an event method to remove any event listener, even the listener that the current event method is part of. The current event method continues executing, but no other event methods in the same event listener object will be called.

**Parameters**

Any number of parameters, each one an [EventListener Object](#)

**Returns**

Nothing

**Example:**

Assume **player** is a MediaPlayer object.

```
var listener = { afterStop: function() { app.alert("Stopped!"); } }
player.events.add( listener );           // add listener
.....
player.events.remove( listener );       // later, remove it
```

---

**EventListener Object**

An EventListener object is a collection of event method functions along with optional local data. Event method names begin with “on” or “after” followed by the event name, e.g. **onPause** or **afterPause**. When an event is dispatched, matching “on” event methods are called immediately, and matching “after” event methods are called a short while later, at the next idle time.

There are severe restrictions on what an “on” event method can do. In particular, an “on” event method for a MediaPlayer cannot call any of that MediaPlayer’s methods, nor can it call any other Acrobat method that may indirectly cause a method of the MediaPlayer to be called. For example, an “on” method must not close the document, save it, change the active page, change the focus, or anything else that may eventually call a method of the MediaPlayer.

An “after” event method does not have these restrictions. For most purposes, “after” event method are more versatile. Use an “on” event method only when the event must be processed synchronously at the time that it occurs, such as an **onGetRect()** method.

A note about reentrancy: “on” event methods are never reentered, but “after” event methods may be reentered.

Inside an event method, **this** is the event listener object. The document is available in **event.media.doc**, and the event target (MediaPlayer or ScreenAnnot) is in **event.target**.

**Events.add()** installs EventListener objects for dispatching, **Events.dispatch()** dispatches an event to the matching event methods, and **Events.remove()** removes EventListener objects from the dispatch table.

**Example:**

```
// Create a simple MediaEvents object
var events = new app.media.Events
({
  // Called immediately during a Play event:
  onPlay: function() { console.println( "onPlay" ); },

  // Called during idle time after the Play event:
  afterPlay: function() { console.println( "afterPlay" ); },
});
var player = app.media.createPlayer({events: events});
player.events.add({
```

```

        afterPlay: function( e ) {
            app.alert("Playback started, doc.URL = " + e.media.doc.URL );
        }
    });
    player.open();

```

---

## EventListener Object Methods

The events listed here are specific to multimedia. In addition to these events, a ScreenAnnot may receive the standard events used elsewhere in Acrobat (Destroy, Mouse Up, Mouse Down, Mouse Enter, Mouse Exit, Page Open, Page Close, Page Visible, Page Invisible, Focus, and Blur). Please see the Events section of the main Acrobat JavaScript documentation for details on those events.

### afterBlur

6.0				
-----	--	--	--	--

The Blur event fires when a MediaPlayer or ScreenAnnot loses the keyboard focus after having it.

#### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

#### Returns

Nothing

See the [onBlur](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

#### Example

The following script is executed as a Rendition action. The user clicks on the ScreenAnnot to open, but not play the movie clip. Clicking outside the ScreenAnnot (a Blur event) plays the movie. Clicking on the ScreenAnnot (a Focus event) while movie is playing pauses the movie. To continue, the user clicks outside the ScreenAnnot again.

```

var playerEvents = new app.media.Events
({
    afterBlur: function () { player.play(); },
    afterFocus: function () { player.pause(); }
});
var settings = { autoPlay: false };
var args = { settings: settings, events: playerEvents};
var player = app.media.openPlayer(args);

```

See also [afterFocus](#).

## afterClose

6.0				
-----	--	--	--	--

The Close event fires when a MediaPlayer is closed for any reason.

If you want to start another media player from the Close event, be sure to test `doc.media.canPlay` first to make sure playback is allowed. For example, playback may not be allowed because the document is closing.

The [Event Object](#) for a Close event includes these properties in addition to the standard Event properties:

---

<code>media.closeReason</code>	Why the player was closed, from <code>app.media.closeReason</code>
<code>media.hadFocus</code>	Did the player have the focus when it was closed?

---

When a player closes while it has the focus, it first receives a Blur event and then the Close event. In the Close event, `media.hadFocus` indicates whether the player had the focus before closing.

When the `afterClose` event method is called, the MediaPlayer has already been deleted and its JavaScript object is dead.

### Parameters

---

<code>oMediaEvent</code>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------------	---

---

### Returns

Nothing

See the [onClose](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

### Example

See [onClose](#) for a representative example.

## afterDestroy

6.0				
-----	--	--	--	--

The Destroy event fires when a ScreenAnnot is destroyed.

When the `afterDestroy` event method is called, the ScreenAnnot has already been deleted from the document and its JavaScript object is dead.



**Parameters**

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [onDestroy](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**afterDone**

6.0				
-----	--	--	--	--

The Done event fires when media playback reaches the end of media.

**Parameters**

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [onDone](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**afterError**

6.0				
-----	--	--	--	--

The Error event fires when an error occurs in a MediaPlayer.

The Event object for an Error event includes these properties in addition to the standard Event properties:

---

<b>media.code</b>	Status code value
<b>media.serious</b>	True for serious errors, false for warnings
<b>media.text</b>	Error message text

---

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [onError](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**afterEscape**

6.0				
-----	--	--	--	--

The Escape event fires when the user presses the Escape key while a MediaPlayer is open and has the keyboard focus. A MediaPlayer may receive an Escape event before it receives the Ready event.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [onEscape](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**afterEveryEvent**

6.0				
-----	--	--	--	--

If an [Events Object](#) contains an [onEveryEvent](#) or [afterEveryEvent](#) property, its event listener function(s) are called for every event, not just a specific one.

The event listener function(s) in an [onEveryEvent](#) or [afterEveryEvent](#) property are called before any listener functions that name the specific event.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [onEveryEvent](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**Example:**

```
var events = new app.media.Events(
{
  // This is called immediately during every event:
  onEveryEvent: function( e )
  { console.println( 'onEveryEvent, event = ' + e.name ); },

  // This is called during a Play event, after onEveryEvent is
  // called:
  onPlay: function() { console.println( "onPlay" ); },

  // This is called for every event, but later during idle time:
  afterEveryEvent: function( e )
  { console.println( "afterEveryEvent, event = " + e.name ); },

  // This is called during idle time after a Play event,
  // and after afterEveryEvent is called:
  afterPlay: function() { console.println( "afterPlay" ); },
});
```

**afterFocus**

6.0				
-----	--	--	--	--

The Focus event fires when a MediaPlayer or ScreenAnnot gets the keyboard focus.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [onFocus](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

### Example

See [afterBlur](#) for an example of usage.

## afterPause

6.0				
-----	--	--	--	--

The Pause event fires when media playback pauses, either because of user interaction or when the [pause \(\)](#) method is called.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [onPause](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## afterPlay

6.0				
-----	--	--	--	--

The Play event fires when media playback starts or resumes, either because of user interaction or when the [play \(\)](#) method is called.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [onPlay](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## afterReady

6.0				
-----	--	--	--	--

The Ready event fires when a newly-created MediaPlayer is ready for use. Most methods of a [MediaPlayer Object](#) cannot be called until the Ready event fires.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [onReady](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

See [afterScript](#), below, [Markers.get](#) and the [MediaOffset Object](#).

### Example

This (document level) script plays multiple media clips. For each ScreenAnnot, a media (OpenPlayer) player is opened. When it is ready, the **afterReady** script signals this fact to Multiplayer.

```
// Parameters: doc, page, rendition/annot name, multiPlayer instance
function OnePlayer( doc, page, name, multiPlayer )
{
    var player = app.media.openPlayer({
        annot: doc.media.getAnnot(
            { nPage: page, cAnnotTitle: name } ),
        rendition: doc.media.getRendition( name ),
        settings: { autoPlay: false },
        events: {
            afterReady: function( e ) {
                multiPlayer.afterReady( player );
            },
        }
    });
    return player;
}
// Parameters: doc, page, list of rendition/annot names
function MultiPlayer( doc, page )
{
    var nPlayersCueing = 0; // number of players cueing up
    var players = [];      // the SinglePlayers

    this.afterReady = function( player ) {
        if( ! player.didAfterReady ) {
            player.didAfterReady = true;
        }
    }
}
```

```

        nPlayersCueing--;
        if( nPlayersCueing == 0 ) this.play();
    }
}
this.play = function() {
    for( var i = 0; i < players.length; i++ ) players[i].play();
}
for( var i = 2; i < arguments.length; i++ ) {
    players[i-2] = new OnePlayer( doc, page, arguments[i], this );
    nPlayersCueing++;
}
}

```

Playing multiple media clips is accomplished by executing the code

```
var myMultiPlayer = new MultiPlayer( this, 0, "Clip1", "Clip2" );
```

from, for example, a mouse up action of a form button.

See **afterScript** for another example of **afterReady**.

## afterScript

6.0				
-----	--	--	--	--

The Script event fires when a script trigger is encountered in the media during playback.

The [Event Object](#) for a Script event includes these properties in addition to the standard Event properties:

<b>media.command</b>	Command name
<b>media.param</b>	Command parameter string

These two strings can contain any values that the media clip provides. They do not necessarily contain executable JavaScript code it is up to the onScript or afterScript event listener to interpret them.

### Parameters

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

### Returns

Nothing

See the [onScript](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## Example

The following is part of a complete example presented after `MediaPlayer.seek()`. The media is an audio clip (.wma), which does support markers and scripts, of (famous) quotations. The `afterReady` listener counts the number of markers, one at the beginning of each quotation. At the end of each quotation, there is also an embedded command script, the `afterScript` listener watches for these commands, and if it is a "pause" command, it pauses the player.

```
var nMarkers=0;
var events = new app.media.Events;
events.add({
  // count the number of quotes in this audio clip, save as nMarkers
  afterReady: function() {
    var g = player.markers;
    while ( (index = g.get( { index: nMarkers } ) ) != null )
      nMarkers++;
  },
  // Each quote should be followed by a script, if the command is to
  // pause, then pause the player.
  afterScript: function( e ) {
    if ( e.media.command == "pause" ) player.pause();
  }
});
var player = app.media.openPlayer({
  rendition: this.media.getRendition( "myQuotes" ),
  settings: { autoPlay: false },
  events: events
});
```

## afterSeek

6.0				
-----	--	--	--	--

The Seek event fires when a MediaPlayer is finished seeking to a playback offset as a result of a `seek()` call. Note that not all media players fire Seek events.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [onSeek](#) and the explanation of the differences between an "on" event and an "after" event in [EventListener Object](#).

## afterStatus

6.0				
-----	--	--	--	--

The Status event fires on various changes of status that a MediaPlayer reports.

The [Event Object](#) for a Status event includes these properties in addition to the standard Event properties:

---

**media.code** Status code value, defined in [app.media.status](#)

---

**media.text** Status message text

---

The following values are used only by some media players, and only when **media.code** == [app.media.status.buffering](#). They are zero otherwise.

---

**media.progress** Progress value from 0 to media.total

---

**media.total** Maximum progress value

---

### Parameters

---

**oMediaEvent** An [Event Object](#) which is automatically passed to this event listener.

---

### Returns

Nothing

See the [onStatus](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

### Example

The following code would monitor the status of the player, as executed from a Rendition event associated with a ScreenAnnot.

```
var events = new app.media.Events
events.add({
  afterStatus: function ( e ) {
    console.println( "Status code " + e.media.code +
      ", description: " + e.media.text);
  }
});
app.media.openPlayer({ events: events });
```



## afterStop

6.0				
-----	--	--	--	--

The Stop event fires when media playback stops, either because of user interaction or when the [stop \(\)](#) method is called.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [onStop](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## onBlur

6.0				
-----	--	--	--	--

The Blur event fires when a MediaPlayer or ScreenAnnot loses the keyboard focus after having it.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [afterBlur](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## onClose

6.0				
-----	--	--	--	--

The Close event fires when a MediaPlayer is closed for any reason.

If you want to start another media player from the Close event, be sure to test `doc.media.canPlay` first to make sure playback is allowed. For example, playback may not be allowed because the document is closing.

The Event object for a Close event includes these properties in addition to the standard Event properties:

<b>media.closeReason</b>	Why the player was closed, from <b>app.media.closeReason</b>
<b>media.hadFocus</b>	Did the player have the focus when it was closed?

When a player closes while it has the focus, it first receives a Blur event and then the Close event. In the Close event, `media.hadFocus` indicates whether the player had the the focus before closing.

When the `afterClose` event method is called, the `MediaPlayer` has already been deleted and its JavaScript object is dead.

### Parameters

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

### Returns

Nothing

See the [afterClose](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

### Example

This script gets information about why the media clip closed, executed from a Rendition action. See [app.media.closeReason](#),

```
var playerEvents = new app.media.Events({
  onClose: function (e) {
    var eReason, r = app.media.closeReason;
    switch ( e.media.closeReason )
    {
      case r.general: eReason = "general"; break;
      case r.error: eReason = "error"; break;
      case r.done: eReason = "done"; break;
      case r.stop: eReason = "stop"; break;
      case r.play: eReason = "play"; break;
      case r.uiGeneral: eReason = "uiGeneral"; break;
      case r.uiScreen: eReason = "uiScreen"; break;
      case r.uiEdit: eReason = "uiEdit"; break;
      case r.docClose: eReason = "Close"; break;
      case r.docSave: eReason = "docSave"; break;
      case r.docChange: eReason = "docChange"; break;
    }
    console.println("Closing...The reason is " + eReason );
  }
});
app.media.openPlayer({ events: playerEvents });
```

## onDestroy

6.0				
-----	--	--	--	--

The Destroy event fires when a ScreenAnnot is destroyed.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [afterDestroy](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## onDone

6.0				
-----	--	--	--	--

The Done event fires when media playback reaches the end of media.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [afterDone](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## onError

6.0				
-----	--	--	--	--

The Error event fires when an error occurs in a MediaPlayer.

The Event object for an Error event includes these properties in addition to the standard Event properties:

---

<b>media.code</b>	Status code value
<b>media.serious</b>	<b>true</b> for serious errors, <b>false</b> for warnings

---

---

<b>media.text</b>	Error message text
-------------------	--------------------

---

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [afterError](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**onEscape**

6.0				
-----	--	--	--	--

The Escape event fires when the user presses the Escape key while a MediaPlayer is open and has the keyboard focus. A MediaPlayer may receive an Escape event before it receives the Ready event.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [afterEscape](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**onEveryEvent**

6.0				
-----	--	--	--	--

If an [Events Object](#) contains an onEveryEvent or afterEveryEvent property, its event listener function(s) are called for every event, not just a specific one.

The event listener function(s) in an onEveryEvent or [afterEveryEvent](#) property are called before any listener functions that name the specific event.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [afterEveryEvent](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**onFocus**

6.0				
-----	--	--	--	--

The Focus event fires when a MediaPlayer or ScreenAnnot gets the keyboard focus.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [afterFocus](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**onGetRect**

6.0				
-----	--	--	--	--

The GetRect event fires whenever the multimedia plug-in needs to get the display rectangle for a docked MediaPlayer.

The Event object for an GetRect event includes this property in addition to the standard Event properties:

---

<b>media.rect</b>	Player rectangle, an array of four numbers in device space
-------------------	--

---

The **onGetRect ()** method must set this property in the **oMediaEvent** before returning.

Although you can write an **afterGetRect** listener, there is no useful purpose for it if it returns a rect property it will be ignored. The **onGetRect** listener is where the rect property must be set.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

**Example**

Page 0 has a series of (thumbnail-size) ScreenAnnots, and page 1 is a blank page. Put the viewer into continuous facing mode so that both pages are seen side-by-side. Below is a typical Rendition action or mouse up button JavaScript action.

```
var rendition = this.media.getRendition("Clip1");
var settings = rendition.getPlaySettings();
var annot = this.media.getAnnot({ nPage:0,cAnnotTitle:"ScreenClip1" });
var player = app.media.openPlayer({
    rendition: rendition,
    annot: annot,
    settings: { windowType: app.media.windowType.docked },
    events:
    {
        onGetRect: function (e) {
            var width = e.media.rect[2] - e.media.rect[0];
            var height = e.media.rect[3] - e.media.rect[1];
            width *= 3; // triple width and height
            height *= 3;
            e.media.rect[0] = 36; // move left, upper to
            e.media.rect[1] = 36; // upper left-hand corner
            e.media.rect[2] = e.media.rect[0]+width;
            e.media.rect[3] = e.media.rect[1]+height;
            return e.media.rect; // return this
        }
    }
});
player.page = 1; // show on page 1, this triggers an onGetRect event.
```

See [MediaPlayer.page](#) and [MediaPlayer.triggerGetRect](#) for a variation on this same example.

**onPause**

6.0				
-----	--	--	--	--

The Pause event fires when media playback pauses, either because of user interaction or when the [play\(\)](#) method is called.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [afterPause](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**onPlay**

6.0				
-----	--	--	--	--

The Play event fires when media playback starts or resumes, either because of user interaction or when the [pause \(\)](#) method is called.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [afterPlay](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

**onReady**

6.0				
-----	--	--	--	--

The Ready event fires when a newly-created MediaPlayer is ready for use. Most methods of a [MediaPlayer Object](#) cannot be called until the Ready event fires.

**Parameters**


---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

**Returns**

Nothing

See the [afterReady](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## onScript

6.0				
-----	--	--	--	--

The Script event fires when a script trigger is encountered in the media during playback.

The Event object for a Script event includes these properties in addition to the standard Event properties:

---

<b>media.command</b>	Command name
<b>media.param</b>	Command parameter string

---

These two strings can contain any values that the media clip provides. They do not necessarily contain executable JavaScript code it is up to the onScript or afterScript event listener to interpret them.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [afterScript](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## onSeek

6.0				
-----	--	--	--	--

The Seek event fires when a MediaPlayer is finished seeking to a playback offset as a result of a [seek](#) () call. Note that not all media players fire Seek events.

### Parameters

---

<b>oMediaEvent</b>	An <a href="#">Event Object</a> which is automatically passed to this event listener.
--------------------	---

---

### Returns

Nothing

See the [afterSeek](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).



## onStatus

6.0				
-----	--	--	--	--

The Status event fires on various changes of status that a MediaPlayer reports.

The [Event Object](#) for a Status event includes these properties in addition to the standard Event properties:

---

**media.code** Status code value, defined in `app.media.status`

---

**media.text** Status message text

---

The following values are used only by some media players, and only when `media.code == app.media.status.buffering`. They are zero otherwise.

---

**media.progress** Progress value from 0 to `media.total`

---

**media.total** Maximum progress value

---

### Parameters

---

**oMediaEvent** An [Event Object](#) which is automatically passed to this event listener.

---

### Returns

Nothing

See the [afterStatus](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#).

## onStop

6.0				
-----	--	--	--	--

The Stop event fires when media playback stops, either because of user interaction or when the `stop ()` method is called.

### Parameters

---

**oMediaEvent** An [Event Object](#) which is automatically passed to this event listener.

---

### Returns

Nothing

See the [afterStop](#) and the explanation of the differences between an “on” event and an “after” event in [EventListener Object](#)

---

## FDF Object

6.0		Ⓢ		
-----	--	---	--	--

This object corresponds to a PDF-encoded data exchange file. The most familiar use of FDF files is to contain forms data that is exported from a PDF file. FDF files can also be used as general purpose data files. It is for this later purpose that the FDF object exists.

(Security Ⓢ): All methods and properties marked with Ⓢ in its quickbar are available only during batch, console, application initialization and menu events.

**NOTE:** Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

---

## FDF Properties

### deleteOption

6.0	Ⓓ	Ⓢ	ⓧ	
-----	---	---	---	--

Indicates whether the FDF file should be automatically deleted after it is processed. This is a generic value that may or may not be used, depending on the content of the FDF file and how it is processed. It is used for embedded files beginning in Acrobat 6.0. Allowed values are

0 (default): Acrobat will automatically delete the FDF file after processing

1: Acrobat will not delete the FDF file after processing (however a web or email browser may still delete the file).

2: Acrobat will prompt the user to determine whether to delete the FDF file after processing (however a web or email browser may still delete the file).

*Type: Integer*

*Access: R/W.*

### isSigned

6.0	Ⓓ		ⓧ	
-----	---	--	---	--

Returns **true** if the FDF data file is signed.

*Type: Boolean*

*Access: R.*

**Example**

See if the fdf is signed.

```
var fdf = app.openFDF("/C/temp/myDoc.fdf");
console.println( "It is "+ fdf.isSigned + " that this FDF is signed");
fdf.close();
```

See a more complete example following `fdf.signatureSign`

**numEmbeddedFiles**

6.0	Ⓓ		ⓧ	
-----	---	--	---	--

The number of files embedded in the FDF file. If the **FDF** object is a valid FDF file, no exceptions will be thrown.

A file may be embedded in an FDF file with the `fdf.addEmbeddedFile` method.

*Type: Integer*

*Access: R.*

**Example**

Create a new FDF object, embed a PDF doc, save the FDF, open the FDF again, and count the number of embedded files.

```
var fdf = app.newFDF();
fdf.addEmbeddedFile("/C/myPDFs/myDoc.pdf");
fdf.save("/c/temp/myDocWrapper.fdf");
fdf = app.openFDF("/c/temp/myDocWrapper.fdf");
console.println("The number of embedded files = "
    + fdf.numEmbeddedFiles);
fdf.close();
```

**FDF Methods****addContact**

6.0	Ⓓ	Ⓔ	ⓧ	
-----	---	---	---	--

Adds a contact to the FDF file.

**Parameters**


---

**oUserEntity** This is a [UserEntity Generic Object](#) which list the contact to be added to the FDF file.

---

**Returns**

Throws an exception on failure.

**Example**

```
var oEntity={firstName:"Fred", lastName:"Smith", fullName:"Fred Smith"};
var f = app.newFDF();
f.addContact( oEntity );
f.save( "/c/temp/FredCert.fdf" );
```

**addEmbeddedFile**

6.0	Ⓓ	Ⓒ	ⓧ	
-----	---	---	---	--

Add the specified file to the end of the array of embedded files in the FDF file. Anyone opening the FDF file will be instructed to save the embedded file or files according to **nSaveOption**. If the embedded file is a PDF file, the file will be opened and displayed in the viewer. If the embedded file is an FDF file, the file will be opened by the viewer for processing. FDF files containing embedded files were supported beginning with Acrobat 4.05. An example use for embedding PDF files is when these files are hosted on an HTTP server and it is desired that the user clicks to download and save the PDF file, rather than viewing the file in the browser. There is no relationship between these embedded files and files that are associated with forms data that is stored in an FDF file.

**Parameters**

<b>cDIPath</b>	(optional) A device-independent absolute path to a file on the user's hard drive. If not specified, the user is prompted to locate a file. See <i>File Specification Strings</i> in the <a href="#">PDF Reference</a> for the exact syntax of the path.
<b>nSaveOption</b>	<p>(optional) How the embedded file will be presented to the person opening this FDF file, where the file will be saved, and whether the file will be deleted after it is saved. Values are:</p> <ul style="list-style-type: none"> <li>● 0: The file will be automatically saved to the Acrobat document folder.</li> <li>● 1 (the default): The user will be prompted for a filename to which to save the embedded file.</li> <li>● 2: Should not be used.</li> <li>● 3: The file will be automatically saved as a temporary file and deleted during cleanup (when Acrobat is closed).</li> </ul> <p>In Acrobat 4.05 through 5.05, for values of 0 and 3, the user is prompted for the location of the save folder if they have not already set this value.</p> <p>For all values of <b>nSaveOption</b>, if the file is a PDF or FDF file it is automatically opened by Acrobat once it is saved.</p>

**Returns**

Throws an exception if this operation could not be completed, otherwise returns the number of embedded files that are now in the FDF file.

**Example**

Create a new FDF, embed a PDF doc, then save.

```
var fdf = app.newFDF();
fdf.addEmbeddedFile("/C/myPDFs/myDoc.pdf");
fdf.save("/c/temp/myDocs.fdf");
```

**addRequest**

6.0	Ⓓ	Ⓔ	ⓧ	
-----	---	---	---	--

Adds a request to the FDF file. There can be only one request in an FDF file. If the FDF file already contains a request, it is replaced with this new request.

**Parameters**

<b>cType</b>	What is being requested. Currently the only valid value is the string "CMS", which is a request for contact information.
<b>cReturnAddress</b>	The return address string for the request. This must begin with <b>mailto:</b> , <b>http:</b> or <b>https:</b> and be of the form <b>"http://www.acme.com/cgi.pl"</b> or <b>"mailto:jdoe@adobe.com"</b> .
<b>cName</b>	(optional) The name of the person or organization that has generated the request.

**Returns**

Throws an exception if there is an error.

**Example**

```
var f = app.newFDF();
f.addRequest("CMS", "http://www.acme.com/cgi.pl", "Acme Corp");
f.save("/c/tmp/request.fdf");
```

**close**

6.0		Ⓔ	ⓧ	
-----	--	---	---	--

Immediately closes the FDF file.

**Parameters**

None

**Returns**

Throws an exception if there is an error.

See the `fdf.save` method, which also closes an FDF file.**Example**The example following [addEmbeddedFile](#) illustrates `fdf.close`.**mail**

6.0		Ⓢ	ⓧ	
-----	--	---	---	--

This method saves the [FDF Object](#) as a temporary FDF file and mails this file as an attachment to all recipients, with or without user interaction. The temporary file is deleted once it is no longer needed.

See also `app.mailGetAddrs`, `app.mailMsg`, `doc.mailDoc`, `doc.mailForm` and `report.mail`.

**NOTE:** On Windows, the client machine must have its default mail program configured to be MAPI enabled in order to use this method.

**Parameters**

<b>bUI</b>	(optional) Whether to display a user interface. If <b>true</b> (the default) the rest of the parameters are used to seed a compose-new-message window that is displayed to the user. If <b>false</b> , the <b>cTo</b> parameter is required and all others are optional.
<b>cTo</b>	(optional) A semicolon-separated list of recipients for the message.
<b>cCc</b>	(optional) A semicolon-separated list of CC recipients for the message.
<b>cBcc</b>	(optional) A semicolon-separated list of BCC recipients for the message.
<b>cSubject</b>	(optional) The subject of the message. The length limit is 64k bytes.
<b>cMsg</b>	(optional) The content of the message. The length limit is 64k bytes.

**Returns**

Throws an exception if there is an error.

**Example**

```
var fdf = app.openFDF( "/c/temp/myDoc.fdf" );
```

```

/* This will pop up the compose new message window */
fdf.mail();

/* This will send out the mail with the attached FDF file to
fun1@fun.com and fun2@fun.com */
fdf.mail( false, "fun1@fun.com", "fun2@fun.com", "",
        "This is the subject", "This is the body.");

```

## save

6.0	Ⓓ	Ⓔ	ⓧ	
-----	---	---	---	--

Save the [FDF Object](#) as a file. A save will always occur. The file is closed when it is saved, and the **FDF** object no longer contains a valid object reference.

See the `fdf.close` method, which also closes a FDF file.

### Parameters

---

<b>cDIPath</b>	The device-independent path of the file to be saved.
	<b>NOTE:</b> (Security Ⓔ): <b>cDIPath</b> must be a <a href="#">Safe Path</a> and must have an extension of <code>.fdf</code> .

---

### Returns

Throws an exception if there is an error.

### Example

Create a new FDF, embed a PDF doc, then save.

```

var fdf = app.newFDF()
fdf.addEmbeddedFile("/C/myPDFs/myDoc.pdf");
fdf.save("/c/temp/myDocs.fdf");

```

## signatureClear

6.0	Ⓓ	Ⓔ	ⓧ	
-----	---	---	---	--

If the [FDF Object](#) is signed, clears the signature and returns `true` if successful. Does nothing if the **FDF** object is not signed. Does not save the file.

### Parameters

None

### Returns

`true` on success.

## signatureSign

6.0	Ⓓ	Ⓒ	ⓧ	
-----	---	---	---	--

Sign the **FDF Object** with the specified security object. **FDF** objects can be signed only once. The **FDF** object is signed in memory and is not automatically saved as a file to disk. Call **save** to save the **FDF** object after it is signed. Call **signatureClear** to clear FDF signatures.

### Parameters

<b>oSig</b>	The <b>SecurityHandler Object</b> that is to be used to sign. Security objects normally require initialization before they can be used for signing. Check the documentation for your security handler to see if it is able to sign FDF files. The <b>signFDF</b> property of the SecurityHandler Object will indicate whether a particular security object is capable of signing FDF files.
<b>oInfo</b>	(optional) A <b>SignatureInfo Object</b> containing the writable properties of the signature.
<b>nUI</b>	(optional) The type of dialog to show when signing. Values are: 0: Show no dialog. 1: Show a simplified dialog with no editable fields (fields can be provided in <b>oInfo</b> ). 2: Show a more elaborate dialog that includes editable fields for reason, location and contact information. The default is 0.
<b>cUISignTitle</b>	(optional) The title to use for the sign dialog. This is only used if <b>nUI</b> is non-zero.
<b>cUISelectMsg</b>	(optional) A message to display when a user is required to select a resource for signing, such as selecting a credential. It is used only when <b>nUI</b> is non-zero.

### Returns

**true** if the signature was applied successfully, **false** otherwise.

### Example

Open existing FDF data file and sign.

```
var eng = security.getHandler( "Adobe.PPKLite" );
eng.login( "myPassword" , "/c/test/Acme.pfx" );
var myFDF = app.openFDF( "/c/temp/myData.fdf" );
if( !myFDF.isSigned ) {
    myFDF.signatureSign({
        oSig: eng,
```



```

        nUI: 1,
        cUISignTitle: "Sign Embedded File FDF",
        cUISelectMsg: "Please select a Digital ID to use to "
            + "sign your embedded file FDF."
    });
    myPDF.save( "/c/temp/myData.fdf" );
};

```

## signatureValidate

6.0			X	
-----	--	--	---	--

Validate the signature of an [PDF Object](#) and return a [SignatureInfo Object](#) specifying the properties of the signature.

### Parameters

<b>oSig</b>	<p>(optional) The security handler to be used to validate the signature. Can be either a <a href="#">SecurityHandler Object</a> or a generic object with the following properties:</p> <ul style="list-style-type: none"> <li>● <b>oSecHdlr</b>: The <a href="#">SecurityHandler Object</a> to use to validate this signature.</li> <li>● <b>bAltSecHdlr</b>: A boolean. If <b>true</b>, an alternate security handler, selected based on user preference settings, may be used to validate the signature. The default is <b>false</b>, meaning that the security handler returned by the signature's <b>handlerName</b> property is used to validate the signature. This parameter is not used if <b>oSecHdlr</b> is provided.</li> </ul> <p>If <b>oSig</b> not supplied, the security handler returned by the signature's <b>handlerName</b> property is used to validate the signature.</p>
<b>bUI</b>	<p>(optional) When <b>true</b>, allow UI to be shown, if necessary, when validating the data file. UI may be used to select a validation handler if none is specified.</p>

### Returns

A [SignatureInfo Object](#). The signature status is described in **status** property.

### Example

```

fdf = app.openPDF("/c/temp/myDoc.fdf");
eng = security.getHandler( "Adobe.PPKLite" );
if (fdf.isSigned)
{
    var oSigInfo = fdf.signatureValidate({
        oSig: eng,
        bUI: true
    });
}

```

```
        console.println("Signature Status: " + oSigInfo.status);
        console.println("Description: " + oSigInfo.statusText);
    } else {
        console.println("FDF not signed");
    }
}
```

---

## Field Object

The **field** object represents an Acrobat form field (that is, a field created using the Acrobat form tool or **doc.addField**). In the same manner that an author might want to modify an existing field's properties like the border color or font, the **field** object gives the JavaScript user the ability to perform the same modifications.

### Field Access from JavaScript

Before a field can be accessed, it must be "bound" to a JavaScript variable through a method provided by the **Doc Object** method interface. More than one variable may be bound to a field by modifying the field's object properties or accessing its methods. This affects all variables bound to that field.

```
var f = this.getField("Total");
```

This example allows the script to now manipulate the form field **Total** by using the variable **f**.

Fields can be arranged hierarchically within a document. For example, form fields can have names like "FirstName" and "LastName". These are called **flat names**, there is no association between these fields. By changing the field names slightly, a hierarchy of fields within the document can be created. For example, if "FirstName" and "LastName" are changed to "Name.First" and "Name.Last", a tree of fields is formed. The period (".") separator in Acrobat Forms is used to denote a hierarchy shift. The "Name" portion of these fields is the parent, and "First" and "Last" are the children. There is no limit to the depth of a hierarchy that can be constructed but it is important that the hierarchy remain manageable. It is also important to clarify some terminology: the field "Name" is known as an **internal** field (that is, it has no visible manifestation) and the fields "First" and "Last" are **terminal** fields (and show up on the page).

A useful property about Acrobat Form fields is that fields that share *the same name* also share *the same value*. Terminal fields can have different presentations of that data; they can appear on different pages, be rotated differently, have a different font or background color, and so on, but they have the same value. This means that if the value of one presentation of a terminal field is modified, all others with the same name get updated automatically. We refer to each presentation of a terminal field as a *widget*.

Individual widgets do not have names. Each individual widget is identified by index (0-based) within its terminal field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). You can easily determine what the index is for a specific widget by looking at the "Fields" panel in Acrobat.

It is the number that follows the '#' sign in the field name shown (in Acrobat 6 or later, the widget index is only displayed if the field has more than one widget). You can double-click an entry in the "Fields" panel to go to the corresponding widget in the document. Alternatively, if you select a field in the document, the corresponding entry in the "Fields" panel is highlighted.

### Doc.getField() Extended to Widgets

A new notation is available when calling `getField` which can be used to retrieve the Field object of one individual widget of a field. This new notation consists of appending a '.' (a dot) followed by the widget index to the field name passed. When this approach is used, the `field` object returned by `getField` encapsulates only one individual widget. You can use the `field` objects returned this way in any place you would use a `field` object returned by simply passing the unaltered field name. However, the set of nodes that are affected may vary, as shown in the following table..

Action	Field Object that Represents All Widgets	Field Object that Represents One Specific Widget
Get a widget property	Gets property of widget # 0	Gets property of that widget
Set a widget property	Sets property of all widgets that are children of that field <sup>a</sup>	Sets property of that widget
Get a field property	Gets property of that field	Gets property of parent field
Set a field property	Sets property of that field	Sets property of parent field

a. Except for the `rect` property and the `setFocus` method. For these cases it applies to widget # 0.

The following example changes the `rect` property of the second radio button (the first would have index 0) of the field "my radio".

```
var f = this.getField("my radio.1");
f.rect = [360, 677, 392, 646];
```

### Field versus Widget Attributes

Some of the properties of the `field` object in JavaScript truly live at the field level, and apply to all widgets that are children of that field. A good example is `value`. Other

properties are, in fact, widget-specific. A good example is `rect`. The following table shows which attributes live at the field level and which at the widget level.

Field Object Properties and Methods that Affect Widget-Level Attributes	Field Object Properties and Methods that Affect Field-Level Attributes
<code>alignment, borderStyle, buttonAlignX, buttonAlignY, buttonPosition, buttonScaleHow, buttonScaleWhen, display, fillColor, hidden, highlight, lineWidth, print, rect, strokeColor, style, textColor, textFont, textSize, buttonGetCaption, buttonGetIcon, buttonImportIcon, buttonSetCaption, buttonSetIcon, checkThisBox<sup>a</sup>, defaultIsChecked<sup>a</sup>, isBoxChecked<sup>a</sup>, isDefaultChecked<sup>a</sup>, setAction<sup>b</sup>, setFocus</code>	<code>calcOrderIndex, charLimit, comb, currentValueIndices, defaultValue, doNotScroll, doNotSpellCheck, delay, doc, editable, exportValues, fileSelect, multiline, multipleSelection, name, numItems, page, password, readonly, required, submitName, type, userName, value, valueAsString, clearItems, browseForFileToSubmit, deleteItemAt, getItemAt, insertItemAt, setAction<sup>b</sup>, setItems, signatureInfo, signatureSign, signatureValidate</code>

- a. These methods take a widget index, `nWidget`, as parameter. If you invoke these methods on a Field object `"f"` that represents one specific widget, then the `nWidget` parameter is optional (and is ignored if passed), and the method acts on the specific widget encapsulated by `"f"`.
- b. Some actions live at the field level, and some at the widget level. The former includes `"Keystroke"`, `"Validate"`, `"Calculate"`, `"Format"`. The latter includes `"MouseUp"`, `"MouseDown"`, `"MouseEnter"`, `"MouseExit"`, `"OnFocus"`, `"OnBlur"`.

## Field Properties

**NOTE:** Some property values are stored in the PDF document as names (see section 3.2.4 on name objects in the [PDF Reference](#)), while others are stored as strings (see section 3.2.3 on string objects in the [PDF Reference](#)). For a property that is stored as a name, there is a 127 character limit on the length of the string.

Examples of properties that have a 127 character limit include `Field.value` and `Field.defaultValue` for a checkbox and radiobutton. The [PDF Reference](#) documents all Annotation properties as well as how they are stored.

### alignment

3.01	Ⓧ		
------	---	--	--

Controls how the text is laid out within the text field. Values are

- left
- center
- right

Type: String

Access: R/W

Fields: **text**.

**Example**

```
var f = this.getField("MyText");
f.alignment = "center";
```

**borderStyle**

3.01	ⓓ		
------	---	--	--

The border style for a field. Valid border styles are

- solid
- dashed
- beveled
- inset
- underline

The border style determines how the border for the rectangle is drawn. The **border** object is a static convenience constant that defines all the border styles of a field, as shown in the following table:

Type	Keyword	Description
<b>solid</b>	<b>border.s</b>	Strokes the entire perimeter of the rectangle with a solid line.
<b>beveled</b>	<b>border.b</b>	Equivalent to the <b>solid</b> style with an additional beveled (pushed-out appearance) border applied inside the solid border.
<b>dashed</b>	<b>border.d</b>	Strokes the perimeter with a dashed line
<b>inset</b>	<b>border.i</b>	Equivalent to the <b>solid</b> style with an additional inset (pushed-in appearance) border applied inside the solid border.
<b>underline</b>	<b>border.u</b>	Strokes the bottom portion of the rectangle's perimeter.

Type: String

Access: R/W

Fields: all.

**Example**

The following example illustrates how to set the border style of a field to **solid**:

```
var f = this.getField("MyField");
f.borderStyle = border.s; /* border.s evaluates to "solid" */
```

## buttonAlignX

5.0	Ⓧ		
-----	---	--	--

Controls how space is distributed from the left of the button face with respect to the icon. It is expressed as a percentage between 0 and 100, inclusive. The default value is 50.

If the icon is scaled anamorphically (which results in no space differences) then this property is not used.

*Type: Integer*

*Access: R/W*

*Fields: **button***

## buttonAlignY

5.0	Ⓧ		
-----	---	--	--

Controls how unused space is distributed from the bottom of the button face with respect to the icon. It is expressed as a percentage between 0 and 100 inclusive. The default value is 50.

If the icon is scaled anamorphically (which results in no space differences) then this property is not used.

*Type: Integer*

*Access: R/W*

*Fields: **button***

### Example

This is an elevator animation. The field "myElevator" is a button form field that has small width and large height. An icon is imported as the appearance face.

```
function MoveIt()
{
  if ( f.buttonAlignY == 0 ) {
    f.buttonAlignY++;
    run.dir = true;
    return;
  }
  if ( f.buttonAlignY == 100 ) {
    f.buttonAlignY--;
    run.dir = false;
    return;
  }
  if (run.dir) f.buttonAlignY++;
  else f.buttonAlignY--;
}
var f = this.getField("myElevator");
f.buttonAlignY=0;
run = app.setInterval("MoveIt()", 100);
run.dir=true;
toprun = app.setTimeout(
```

```
"app.clearInterval(run); app.clearTimeOut(toprun)", 2*20000+100);
```

## buttonFitBounds

6.0	ⓓ		
-----	---	--	--

When **true**, the extent to which the icon may be scaled is set to the bounds of the button field; the additional icon placement properties are still used to scale/position the icon within the button face.

In previous versions of Acrobat, the width of the field border was always taken into consideration when scaling an icon to fit a button face, even when no border color was specified. Setting this property to **true** when a border color has been specified for the button will cause an exception to be raised.

Type: Boolean

Access: R/W

Fields: **button**

## buttonPosition

5.0	ⓓ		
-----	---	--	--

Controls how the text and the icon of the button are positioned with respect to each other within the button face. The convenience **position** object defines all of the valid alternatives:

Icon/Text Placement	Keyword
Text Only	<b>position.textOnly</b>
Icon Only	<b>position.iconOnly</b>
Icon top, Text bottom	<b>position.iconTextV</b>
Text top, Icon bottom	<b>position.textIconV</b>
Icon left, Text right	<b>position.iconTextH</b>
Text left, Icon right	<b>position.textIconH</b>
Text in Icon (overlaid)	<b>position.overlay</b>

Type: Integer

Access: R/W

Fields: **button**

## buttonScaleHow

5.0	ⓓ		
-----	---	--	--

Controls how the icon is scaled (if necessary) to fit inside the button face. The convenience **scaleHow** object defines all of the valid alternatives:

How is Icon Scaled	Keyword
Proportionally	<b>scaleHow.proportional</b>
Non-proportionally	<b>scaleHow.anamorphic</b>

Type: *Integer*

Access: *R/W*

Fields: **button**.

## buttonScaleWhen

5.0	ⓓ		
-----	---	--	--

Controls when an icon is scaled to fit inside the button face. The convenience **scaleWhen** object defines all of the valid alternatives:

When is Icon Scaled	Keyword
Always	<b>scaleWhen.always</b>
Never	<b>scaleWhen.never</b>
If icon is too big	<b>scaleWhen.tooBig</b>
If icon is too small	<b>scaleWhen.tooSmall</b>

Type: *Integer*

Access: *R/W*

Fields: **button**.

## calcOrderIndex

3.01	ⓓ		
------	---	--	--

Changes the calculation order of fields in the document. When a computable **text** or **combobox** field is added to a document, the field's name is appended to the calculation order array. The calculation order array determines the order fields are calculated in the document. The **calcOrderIndex** property works similarly to the **Calculate** tab used by the Acrobat Form tool.

Type: *Integer*

Access: *R/W*

Fields: **combobox**, **text**.



**Example**

```
var a = this.getField("newItem");
var b = this.getField("oldItem");
a.calcOrderIndex = b.calcOrderIndex + 1;
```

In this example, [getField](#) gets the "newItem" field that was added after "oldItem" field. It then changes the [calcOrderIndex](#) of the "oldItem" field so that it is calculated before "newItem" field.

**charLimit**

3.01	Ⓣ		
------	---	--	--

Limits the number of characters that a user can type into a text field.

See [event.fieldFull](#) to detect when the maximum number of characters is reached.

Type: *Integer*

Access: *R/W*

Fields: **text**.

**Example**

Set a limit on the number of characters that can be typed into a field.

```
var f = this.getField("myText");
f.charLimit = 20;
```

**comb**

6.0	Ⓣ		
-----	---	--	--

If set to **true**, the field background is drawn as series of boxes (one for each character in the value of the field) and the each character of the content is drawn within those boxes. The number of boxes drawn is determined from the [field.charLimit](#) property.

It applies only to text fields. The setter will also raise if any of the following field properties are also set [multiline](#), [password](#), and [fileSelect](#). A side-effect of setting this property is that the [doNotScroll](#) property is also set.

Type: *Boolean*

Access: *R/W*

Fields: **text**.

**Example**

Create a comb field in the upper left corner of a newly created document.

```
var myDoc = app.newDoc(); // create a blank doc
var Bbox = myDoc.getPageBox("Crop"); // get crop box
var inch = 72;

// add a text field at the top of the document
var f = myDoc.addField("Name.Last", "text", 0,
    [ inch, Bbox[1]-inch, 3*inch, Bbox[1]- inch - 14 ] );
```

```
// add some attributes to this text field
f.strokeColor = color.black;
f.textColor = color.blue;
f.fillColor = ["RGB",1,0.66,0.75];

f.comb = true; // declare this is a comb field
f.charLimit = 10; // Max number of characters
```

## commitOnSelChange

6.0	ⓓ		
-----	---	--	--

Controls whether a field value is committed after a selection change. When **true**, the field value is committed immediately when the selection is made. When **false**, the user can change the selection multiple times without committing the field value; the value is committed only when the field loses focus, that is, when the user clicks outside the field.

Type: Boolean

Access: R/W

Fields: **combobox**, **listbox**

## currentValueIndices

5.0	ⓓ		
-----	---	--	--

Reads and writes single or multiple values of a **listbox** or **combobox**.

### Read

Returns the options-array indices of the strings that are the value of a **listbox** or **combobox** field. These indices are 0-based. If the value of the field is a single string then it returns an integer. Otherwise, it returns an array of integers sorted in ascending order. If the current value of the field is not a member of the set of offered choices (as could happen in the case of an editable combobox) it returns -1.

### Write

Sets the value of a **listbox** or **combobox**. It accepts either a single integer, or an array of integers, as an argument. To set a single string as the value, pass an integer which is the 0-based index of that string in the options array. Note that in the case of an editable **combobox**, if the desired value is not a member of the set of offered choices, then you must set the **value** instead. Except for this case, **currentValueIndices** is the preferred way to set the value of a **listbox** or **combobox**.

To set a multiple selection for a **listbox** that allows it, pass an array as argument to this property, containing the indices (sorted in ascending order) of those strings in the options array. This is the *only* way to invoke multiple selection for a **listbox** from JavaScript. The ability for a **listbox** to support multiple section can be set through [multipleSelection](#).

Related Field methods and properties include [numItems](#), [getItemAt](#), [insertItemAt](#), [deleteItemAt](#) and [setItems](#).

Type: *Integer | Array*

Access: *R/W*

Fields: **combobox**, **listbox**

### Example (Read)

The script below is a mouse up action of a button. The script gets the current value of a list box.

```
var f = this.getField("myList");
var a = f.currentValueIndices;
if (typeof a == "number") // a single selection
    console.println("Selection: " + f.getItemAt(a, false));
else { // multiple selections
    console.println("Selection:");
    for (var i = 0; i < a.length; i ++)
        console.println(" " + f.getItemAt(a[i], false));
}
```

### Example (Write)

The following code, selects the second and fourth (0-based index values, 1 and 3, respectively) in a listbox.

```
var f = this.getField("myList");
f.currentValueIndices = [1,3];
```

## defaultStyle

6.0			
-----	--	--	--

This property defines the default style attributes for the form field. If the user clicks into an empty field and begins entering text without changing properties using the property toolbar, these are the properties that will be used. This property is a single [Span Object](#) without a [text](#) property. Some of the properties in the default style span mirror the properties of the field object. Changing these properties also modifies the **defaultStyle** property for the field and vice versa.

The following table details the properties of the field object that are also in the default style and any differences between their values.

Field Properties	defaultStyle (Span Properties)	Description
<a href="#">alignment</a>	<a href="#">alignment</a>	The <b>alignment</b> property has the same values for both the default style and the field object.

Field Properties	defaultStyle (Span Properties)	Description
<code>textFont</code>	<code>fontFamily</code> <code>fontStyle</code> <code>fontWeight</code>	The value of this field property is a complete font name which represents the font family, weight and style. In the default style property each property is represented separately. If an exact match for the font properties specified in the default style cannot be found a similar font will be used or synthesized.
<code>textColor</code>	<code>textColor</code>	The <code>textColor</code> property has the same values for both the default style and the field objec.
<code>textSize</code>	<code>textSize</code>	The <code>textSize</code> property has the same values for both the default style and the field object.

**NOTES:** When a field is empty, `defaultStyle` is the style used for newly entered text. If a field already contains text when the `defaultStyle` is changed the text will not pick up any changes to `defaultStyle`; newly entered text will use the attributes of the text it is inserted into (or specified with the toolbar).

When pasting rich text into a field any unspecified attributes in the pasted rich text will be filled with those from the `defaultStyle`.

Superscript and Subscript are ignored in the `defaultStyle`.

Type: *Span Object*

Access: *R/W*

Fields: **rich text.**

### Example

Change the default style for a text field.

```
var style = this.getField("Text1").defaultStyle;
style.textColor = color.red;
style.textSize = 18;

// if Courier Std is not found on the user's system, use a monospace
style.fontFamily = ["Courier Std", "monospace" ];

this.getField("Text1").defaultStyle = style;
```

### defaultValue

3.01	Ⓣ		
------	---	--	--

Exposes the default value of a field. This is the value that the field is set to when the form is reset. For **comboboxes** and **listboxes** either an export or a user value can be used to set the default. In the case of a conflict (for example, the field has an export value and a user

value with the same string but these apply to different items in the list of choices), the export value is matched against first.

Type: *String*

Access: *R/W*

Fields: *all except button, signature.*

**Example**

```
var f = this.getField("Name");
f.defaultValue = "Enter your name here.";
```

**doNotScroll**

5.0	Ⓧ		
-----	---	--	--

When **true**, the text field does not scroll and the user, therefore, is limited by the rectangular region designed for the field. Setting this property to **true** or **false** corresponds to checking or unchecking the "Scroll long text" field in the Options tab of the field.

Type: *Boolean*

Access: *R/W*

Fields: **text**.

**doNotSpellCheck**

5.0	Ⓧ		
-----	---	--	--

When **true**, spell checking is *not* performed on this editable text field. Setting this property to **true** or **false** corresponds to unchecking or checking the "Check spelling" attribute in the Options tab of the Field Properties dialog.

Type: *Boolean*

Access: *R/W*

Fields: **combobox (editable), text**.

**delay**

3.01			
------	--	--	--

Delays the redrawing of a field's appearance. It is generally used to buffer a series of changes to the properties of the field before requesting that the field regenerate its appearance. Setting the property to **true** forces the field to wait until **delay** is set to **false**. The update of its appearance then takes place, redrawing the field with its latest settings.

There is a corresponding **doc.delay** flag if changes are being made to many fields at once.

Type: *Boolean*

Access: *R/W*

Fields: *all*.

**Example**

This example changes the appearance of a checkbox; it sets the **delay** to **true**, makes changes, and sets **delay** to **false**.

```
// Get the myCheckBox field
var f = this.getField("myCheckBox");
// set the delay and change the fields properties
// to beveled edge and medium thickness line.
f.delay = true;
f.borderStyle = border.b;
... // a number of other changes
f.strokeWidth = 2;
f.delay = false; // force the changes now
```

**display**

4.0	Ⓢ		
-----	---	--	--

Controls whether the field is hidden or visible on screen and in print. Values are:

Effect	Keyword
Field is visible on screen and in print	<b>display.visible</b>
Field is hidden on screen and in print	<b>display.hidden</b>
Field is visible on screen but does not print	<b>display.noPrint</b>
Field is hidden on screen but prints	<b>display.noView</b>

This property supersedes the older **hidden** and **print** properties.

Type: Integer

Access: R/W

Fields: all.

**Example**

```
// Set the display property
var f = getField("myField");
f.display = display.noPrint;

// Test whether field is hidden on screen and in print
if (f.display == display.hidden) console.println("hidden");
```

**doc**

3.01			
------	--	--	--

Returns the **Doc Object** of the document to which the field belongs.

*Type: object**Access: R/W**Fields: all.*

## editable

3.01	Ⓓ		
------	---	--	--

Controls whether a **combobox** is editable. When **true**, the user can type in a selection. When **false**, the user must choose one of the provided selections.

*Type: Boolean**Access: R/W**Fields: **combobox***

### Example

```
var f = this.getField("myComboBox");
f.editable = true;
```

## exportValues

5.0	Ⓓ		
-----	---	--	--

The array of export values defined for the field. For radio button fields, this is necessary to make the field work properly as a group with the one button checked at any given time giving its value to the field as a whole. For checkbox fields, unless an export value is specified, the default used when the field is checked is "Yes" (or the corresponding localized string). When it is unchecked, its value is "Off" (this is also true for a radio button field when none of its buttons are checked). This property contains an array of strings with as many elements as there are annotations in the field. The elements of the array are mapped to the individual annotations comprising the field in the order of creation (unaffected by tab-order).

*Type: Array**Access: R/W**Fields: **checkbox, radiobutton***

### Example

This example creates a radio button field and sets its export values.

```
var d = 40;
var f = this.addField("myRadio","radiobutton",0, [200, 510, 210, 500]);
this.addField("myRadio","radiobutton",0, [200+d, 510-d, 210+d, 500-d]);
this.addField("myRadio","radiobutton",0,[200, 510-2*d, 210, 500-2*d]);
this.addField("myRadio","radiobutton",0,[200-d, 510-d, 210-d, 500-d]);
f.strokeColor = color.black;
// now give each radio field an export value
f.exportValues = ["North", "East", "South", "West"];
```

## fileSelect

5.0	Ⓓ	Ⓔ	
-----	---	---	--

When **true**, sets the file-select flag in the **Options** tab of the **text** field (“Field is Used for File Selection”). This indicates that the value of the field represents a pathname of a file whose contents may be submitted with the form.

The pathname may be entered directly into the field by the user, or the user can browse for the file. (See the [browseForFileToSubmit](#).)

**NOTES:** The file select flag is mutually exclusive with the [multiline](#), [charLimit](#), [password](#), and [defaultValue](#). Also, on the Macintosh platform, when setting the file select flag, the field gets treated as read-only; hence, the user must browse for the file to enter into the field. (See the [browseForFileToSubmit](#).)

(SecurityⒺ): This property can only be set during batch, menu, or console events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

Type: Boolean

Access: R/W

Fields: **text**.

## fillColor

4.0	Ⓓ		
-----	---	--	--

Specifies the background color for a field. The background color is used to fill the rectangle of the field. Values are defined by using **transparent**, **gray**, **RGB** or **CMYK** color. See [Color Arrays](#) for information on defining color arrays and how values are used with this property.

In older versions of this specification, this property was named **bgColor**. The use of **bgColor** is now discouraged but for backwards compatibility is still valid.

Type: Array

Access: R/W

Fields: *all*.

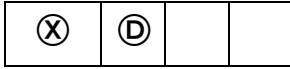
### Example

This code changes the background color of a text field: If the current color is red, change color to blue, otherwise, change color to yellow.

```
var f = this.getField("myField");
if (color.equal(f.fillColor, color.red))
    f.fillColor = color.blue;
else
    f.fillColor = color.yellow;
```



## hidden



Controls whether the field is hidden or visible to the user. If the value is **false** the field is visible, **true** the field is invisible. The default value for **hidden** is **false**.

See also the **display** which supersedes this property in later versions.

Type: Boolean

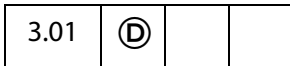
Access: R/W

Fields: all.

### Example

```
var f = this.getField("myField");
f.hidden = true; // Set the field to hidden
```

## highlight



Defines how a button reacts when a user clicks it. The four highlight modes supported are:

**none**: No visual indication that the button has been clicked.

**invert**: Click causes the region encompassing the button's rectangle to invert momentarily.

**push**: Click displays the down face for the button (if any) momentarily.

**outline**: Click causes the border of the rectangle to invert momentarily.

The convenience **highlight** object defines each state, as follows:

Type	Keyword
<b>none</b>	<b>highlight.n</b>
<b>invert</b>	<b>highlight.i</b>
<b>push</b>	<b>highlight.p</b>
<b>outline</b>	<b>highlight.o</b>

Type: String

Access: R/W

Fields: **button**.

### Example

The following example sets the **highlight** property of a button to "invert".

```
// set the highlight mode on button to invert
var f = this.getField("myButton");
f.highlight = highlight.i;
```

## lineWidth

4.0	ⓓ		
-----	---	--	--

Specifies the thickness of the border when stroking the perimeter of a field's rectangle. If the stroke color is transparent, this parameter has no effect except in the case of a beveled border. Values are:

- 0: none
- 1: thin
- 2: medium
- 3: thick

In older versions of this specification, this property was **borderWidth**. The use of **borderWidth** is now discouraged but for backwards compatibility is still valid.

*Type: Integer*

*Access: R/W*

*Fields: all.*

### Example

```
// Change the border width of the Text Box to medium thickness
f.lineWidth = 2
```

The default value for **lineWidth** is 1 (**thin**). Any integer value can be used; however, values beyond 5 may distort the field's appearance.

## multiline

3.01	ⓓ		
------	---	--	--

Controls how the text is wrapped within the field. When **false**, the default, the text field can be a single line only. When **true**, multiple lines are allowed and wrap to field boundaries.

*Type: Boolean*

*Access: R/W*

*Fields: text.*

### Example

See the [Example 1](#) following `doc.getField`.

## multipleSelection

5.0	ⓓ		
-----	---	--	--

If **true**, indicates that a listbox allows multiple selection of the items.

See also [type](#), [value](#), and [currentValueIndices](#).

*Type: Boolean*

*Access: R/W*

*Fields: listbox*

## name

3.01			
------	--	--	--

This property returns the fully qualified field name of the field as a string object.

Beginning with Acrobat 6.0, if the [Field Object](#) represents one individual widget, then the returned name includes an appended '!' followed by the widget index.

*Type: String*

*Access: R*

*Fields: all.*

### Example

Get a field object, and write the **name** of the field to the console.

```
var f = this.getField("myField");

// displays "myField" in console window
console.println(f.name);
```

## numItems

3.01			
------	--	--	--

The number of items in a **combobox** or **listbox**.

*Type: Integer*

*Access: R*

*Fields: combobox, listbox.*

### Example

Get the number of items in a list box.

```
var f = this.getField("myList");
console.println("There are " + f.numItems + " in this listbox");
```

Face names and values of a combobox or listbox can be access through the [getItemAt](#) method. See that method for an additional example of **numItems**.

## page

5.0			
-----	--	--	--

Returns the page number or an array of page numbers of a field. If the field has only one appearance in the document, the **page** property will return an integer representing the (0 based) page number of the page on which the field appears. If the field has multiple appearances, it will return an array of integers, each member of which is a (0 based) page number of an appearance of the field. The order in which the page numbers appear in the array is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). If an appearance of the field is on a hidden template page, **page** returns a value of -1 for that appearance.

*Type: Integer|Array**Access: R**Fields: all.***Example 1**

```
var f = this.getField("myField");
if (typeof f.page == "number")
    console.println("This field only occurs once on page " + f.page);
else
    console.println("This field occurs " + f.page.length + " times);
```

**Example 2 (Version 6.0)**

The page property is one way of discovering the number of widgets associated with a field name. For example, programmatically discover the number of radio buttons in a radio button field.

```
var f = this.getField("myRadio");
if ( typeof f.page == "object" )
    console.println("There are " + f.page.length
        + " radios in this field.");
```

**password**

3.01	ⓓ		
------	---	--	--

Causes the field to display asterisks for the data entered into the field. Upon submission, the actual data entered is sent. Fields that have the password attribute set will not have the data in the field saved when the document is saved to disk.

*Type: Boolean**Access: R/W**Fields: text.***print**

ⓧ	ⓓ		
---	---	--	--

Determines whether a given field prints or not. Set the **print** property to **true** to allow the field to appear when the user prints the document, set it to **false** to prevent printing. This property can be used to hide control buttons and other fields that are not useful on the printed page.

This property has been superseded by the [display](#) and its use is discouraged.

*Type: Boolean**Access: R/W**Fields: all.***Example**

Set a field so that it does not print.

```
var f = this.getField("myField");
f.print = false;
```

## radiosInUnison

6.0	ⓓ		
-----	---	--	--

When **false**, even if a group of radio buttons have the same name and export value, they behave in a mutually exclusive fashion, like HTML radio buttons. The default for new radio buttons is **false**.

When **true**, if a group of radio buttons have the same name and export value, they turn on and off in unison, as in Acrobat 4.0.

Type: Boolean

Access: R/W

Fields: **radiobutton**

## readonly

3.01	ⓓ		
------	---	--	--

Sets or gets the read-only characteristic of a field. If a field is read-only, the user can see the field but cannot change it.

Type: Boolean

Access: R/W

Fields: *all*.

### Example

Get a field and make it read-only.

```
var f = this.getField("myTextField");
f.value = "You can't change this message!";
f.readonly = true;
```

## rect

5.0	ⓓ		
-----	---	--	--

Sets or gets an array of four numbers in *Rotated User Space* that specifies the size and placement of the form field. These four numbers are the coordinates of the bounding rectangle and are listed in the following order: upper-left x, upper-left y, lower-right x and lower-right y.

**NOTE:** **Annot Object** also has a **rect**, but: 1) the coordinates are not in *Rotated User Space*, and 2) they are in different order than in **field.rect**.

Type: Array

Access: R/W

Fields: *all*.

### Example 1

Lay out a 2-inch-wide text field just to the right of the field "myText".

```
var f = this.getField("myText"); // get the field object
```

```

var myRect = f.rect;                // and get it's rectangle

// make needed coordinate adjustments for new field
myRect[0] = f.rect[2]; // the ulx for new = lrx for old
myRect[2] += 2 * 72; // move over two inches for lry
f = this.addField("myNextText", "text", this.pageNum, myRect);
f.strokeColor = color.black;

```

**Example 2**

Move a button field that already exists over 10 points to the right.

```

var b = this.getField("myButton");
var aRect = b.rect; // make a copy of b.rect
aRect[0] += 10;    // increment first x-coordinate by 10
aRect[2] += 10;    // increment second x-coordinate by 10
b.rect = aRect;    // update the value of b.rect

```

**required**

3.01	Ⓓ		
------	---	--	--

Sets or gets the *required* characteristic of a field. When **true**, the field's value must be non-**null** when the user clicks a submit button that causes the value of the field to be posted. If the field value is **null**, the user receives a warning message and the submit does not occur.

*Type: Boolean*

*Access: R/W*

*Fields: all except **button**.*

**Example**

Make "myField" into a required field.

```

var f = this.getField("myField");
f.required = true;

```

**richText**

6.0	Ⓓ		
-----	---	--	--

Gets and sets the rich text property of the text field. If **true**, the field will allow rich text formatting. The default is **false**.

*Type: Boolean*

*Access: R/W*

*Fields: **text**.*

Related objects and properties are the [Span Object](#), **field.richValue**, **field.defaultStyle**, **event.richValue**, **event.richChange**, **event.richChangeEx**, and **annot.richContents**.

**Example 1**

Get a field object, and set it for rich text formatting.

```
var f = this.getField("Text1");
f.richText = true;
```

See [Example 2](#) following [richValue](#) for a more complete example.

**Example 2**

Count the number of rich text fields in the document.

```
var count = 0;
for ( var i = 0; i < this.numFields; i++)
{
    var fname = this.getNthFieldName(i);
    var f = this.getField(fname);
    if ( f.type == "text" && f.richText ) count++
}
console.println("There are a total of "+ count + " rich text fields.");
```

**richValue**

6.0			
-----	--	--	--

This property gets the text contents and formatting of a rich text field. For field types other than rich text this property is **undefined**. The rich text contents are represented as an array of [Span Objects](#) containing the text contents and formatting of the field.

Type: Array of [Span Objects](#) Access: R/W

Fields: **rich text**.

Related objects and properties are the [Span Object](#), [field.richText](#), [field.defaultStyle](#), [event.richValue](#), [event.richChange](#), [event.richChangeEx](#), and [annot.richContents](#).

**Example 1**

This example turns all bold text into red underlined text.

```
var f = this.getField("Text1");
var spans = f.richValue;
for ( var i = 0; i < spans.length; i++ )
{
    if( spans[i].fontWeight >= 700 )
    {
        spans[i].textColor = color.red;
        spans[i].underline = true;
    }
}
f.richValue = spans;
```

**Example 2**

This example creates a text field, marks it for rich text formatting, and inserts rich text.

```

var myDoc = app.newDoc(); // create a blank doc
var Bbox = myDoc.getPageBox("Crop"); // get crop box
var inch = 72;

// add a text field at the top of the document
var f = myDoc.addField("Text1", "text", 0,
    [72, Bbox[1]-inch, Bbox[2]-inch, Bbox[1]-2*inch ] );
// add some attributes to this text field
f.strokeColor = color.black;
f.richText = true; // rich text
f.multiline = true; // multiline

// now build up an array of Span Objects
var spans = new Array();
spans[0] = new Object();
spans[0].text = "Attention:\r";
spans[0].textColor = color.blue;
spans[0].textSize = 18;

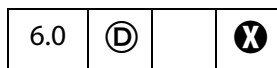
spans[1] = new Object();
spans[1].text = "Adobe Acrobat 6.0\r";
spans[1].textColor = color.red;
spans[1].textSize = 20;
spans[1].alignment = "center";

spans[2] = new Object();
spans[2].text = "will soon be here!";
spans[2].textColor = color.green;
spans[2].fontStyle = "italic";
spans[2].underline = true;
spans[2].alignment = "right";

// now give the rich field a rich value
f.richValue = spans;

```

## rotation



Determines the rotation of a widget in 90 degree counter-clockwise increments. Valid values are 0, 90, 180, 270.

*Type: Integer*

*Access: R/W*

*Fields: all.*

### Example

Create a rotated text field on each page and fill it with text.

```

for ( var i=0; i < this.numPages; i++) {
    var f = this.addField("myRotatedText"+i,"text",i,[6, 6+72, 18, 6]);
}

```



```
f.rotation = 90; f.value = "Confidential";
f.textColor = color.red; f.readonly = true;
}
```

## strokeColor

4.0	ⓓ		
-----	---	--	--

Specifies the stroke color for a field which is used to stroke the rectangle of the field with a line as large as the line width. Values are defined by using **transparent**, **gray**, **RGB** or **CMYK** color. See [Color Arrays](#) for information on defining color arrays and how values are used with this property.

In older versions of this specification, this property was **borderColor**. The use of **borderColor** is now discouraged but for backwards compatibility is still valid.

Type: Array

Access: R/W

Fields: all.

### Example

Change the stroke color of each text field in the document to red.

```
for ( var i=0; i < this.numFields; i++) {
  var fname = this.getNthFieldName(i);
  var f = this.getField(fname);
  if ( f.type == "text" ) f.strokeColor = color.red;
}
```

## style

3.01	ⓓ		
------	---	--	--

Allows the user to set the glyph style of a **checkbox** or **radiobutton**. The glyph style is the graphic used to indicate that the item has been selected.

The style values are associated with keywords as follows:

Style	Keyword
check	<b>style.ch</b>
cross	<b>style.cr</b>
diamond	<b>style.di</b>
circle	<b>style.ci</b>
star	<b>style.st</b>
square	<b>style.sq</b>

*Type: String**Access: R/W**Fields: **checkbox**, **radiobutton**.***Example**

The following example illustrates the use of this property and the style object, it sets the glyph style to circle.

```
var f = this.getField("myCheckbox");
f.style = style.ci;
```

**submitName**

5.0	Ⓓ		
-----	---	--	--

If nonempty, used during form submission instead of **name**. Only applicable if submitting in HTML format (that is, URLencoded).

*Type: String**Access: R/W**Fields: all.***textColor**

4.0	Ⓓ		
-----	---	--	--

This property determines the foreground color of a field. It represents the text color for **text**, **button**, or **listbox** fields and the check color for **checkbox** or **radio button** fields. Values are defined the same as the **fillColor**. See [Color Arrays](#) for information on defining color arrays and how values are set and used with this property.

In older versions of this specification, this property was **fgColor**. The use of **fgColor** is now discouraged but for backwards compatibility is still valid.

**NOTE:** An exception is thrown if a transparent color space is used to set **textColor**.

*Type: Array**Access: R/W**Fields: all.***Example**

This example sets the foreground color to red.

```
var f = this.getField("myField");
f.textColor = color.red;
```

**textFont**

3.01	Ⓓ		
------	---	--	--

Determines the font that is used when laying out text in a **text field**, **combobox**, **listbox** or **button**. Valid fonts are defined as properties of the [font Object](#). Beginning

with Acrobat 5.0, arbitrary fonts can also be used, see the paragraph on the [Use of arbitrary fonts](#).

Type: String

Access: R/W

Fields: **button**, **combobox**,  
**listbox**, **text**.

## font Object

Text Font	Keyword
Times-Roman	<code>font.Times</code>
Times-Bold	<code>font.TimesB</code>
Times-Italic	<code>font.TimesI</code>
Times-BoldItalic	<code>font.TimesBI</code>
Helvetica	<code>font.Helv</code>
Helvetica-Bold	<code>font.HelvB</code>
Helvetica-Oblique	<code>font.HelvI</code>
Helvetica-BoldOblique	<code>font.HelvBI</code>
Courier	<code>font.Cour</code>
Courier-Bold	<code>font.CourB</code>
Courier-Oblique	<code>font.CourI</code>
Courier-BoldOblique	<code>font.CourBI</code>
Symbol	<code>font.Symbol</code>
ZapfDingbats	<code>font.ZapfD</code>

## Use of arbitrary fonts

Beginning with Acrobat 5.0, an arbitrary font can be used when laying out a **text** field, **combobox**, **listbox** or **button** by setting the value of **textFont** to the **PDSysFont** font name, as returned by **PDSysFontGetName**, see the [Acrobat and PDF Library API Reference](#).

How to find the **PDSysFont** font name of a font:

1. Create a text field in a PDF document. Using the UI, set the text font for this field to the desired font.
2. Open the JavaScript Debugger Console and execute the script

```
this.getField("Text1").textFont
```

The above code assumes the name of the field is **Text1**.

- The string returned to the console is the font name needed to programmatically set the text font.

**NOTE:** Use of arbitrary fonts as opposed to those listed in the [font Object](#) creates compatibility problems with older versions of the Viewer.

**Example**

The following example illustrates the use of this property and the font object, it sets the font to Helvetica.

```
var f = this.getField("myField");
f.textFont = font.Helv;
```

**Example (Acrobat 5.0)**

Set the font of "myField" to Viva-Regular.

```
var f = this.getField("myField");
f.textFont = "Viva-Regular";
```

**textSize**

3.01	Ⓧ		
------	---	--	--

Controls the text size (in points) to be used in all controls. In **checkbox** and **radiobutton** fields, the text size determines the size of the check. Valid text sizes range from 0 to 32767 inclusive. A text size of zero means to use the largest point size that will allow all text data to fit in the field's rectangle.

Type: Number

Access: R/W

Fields: all.

**Example**

Set the text size of "myField" to 28 points.

```
this.getField("myField").textSize = 28;
```

**type**

3.01			
------	--	--	--

Returns the type of the field as a string. Valid types are:

- button
- checkbox
- combobox
- listbox
- radiobutton
- signature
- text

*Type: String**Access: R**Fields: all.***Example**

Count the number of text field in the document.

```
var count = 0;
for ( var i=0; i<this.numFields; i++) {
    var fname = this.getNthFieldName(i);
    if ( this.getField(fname).type == "text" ) count++;
}
console.println("There are " + count + " text fields.");
```

**userName**

3.01	Ⓣ		
------	---	--	--

Gets or sets the user name (short description string) of the field. The user name is intended to be used as tooltip text whenever the mouse cursor enters a field. It can also be used as a user-friendly name when generating error messages instead of the field name.

*Type: String**Access: R/W**Fields: all.***Example**

Add a tooltip to a button field.

```
var f = this.getField("mySubmit");
f.userName = "Press this button to submit your data.";
```

**value**

3.01	Ⓣ		
------	---	--	--

Gets the value of the field data that the user has entered. Depending on the [type](#) of the field, may be a String, Date, or Number. Typically, the **value** is used to create calculated fields.

Beginning with Acrobat 6.0, if a field contains rich text formatting, modifying this property will discard the formatting and regenerate the field value and appearance using the [defaultStyle](#) and plain text value. To modify the field value and maintain formatting use the [richValue](#) property.

**NOTES:** For **signature** fields, if the field has been signed then a non-**null** string is returned as the value.

For Acrobat 5.0 or later, if the field is a **listbox** that accepts multiple selection (see [multipleSelection](#)), you can pass an array to set the **value** of the field, and **value** returns an array for a **listbox** with multiple values currently selected.

The **currentValueIndices** of a listbox that has multiple selections is the preferred and most efficient way to get and set the value of this type of field.

See also **field.valueAsString**, and **event.type**.

Type: *various*

Access: *R/W*

Fields: *all except button*

### Example

In this example, the **value** of the field being calculated is set to the sum of the "oil" and "filter" fields and multiplied by the state sales tax.

```
var oil = this.getField("Oil");
var filter = this.getField("Filter");
event.value = (oil.value + filter.value) *1.0825;
```

## valueAsString

5.0	Ⓧ		
-----	---	--	--

Returns the value of a field as a JavaScript string.

This differs from **value**, which attempts to convert the contents of a field contents to an accepted format. For example, for a field with a value of "020", **value** returns the integer 20, while **valueAsString** returns the string "020".

Type: *String*

Access: *R*

Fields: *all except button*

## Field Methods

### browseForFileToSubmit

5.0	Ⓧ		
-----	---	--	--

When invoked on a **text** field for which the **fileSelect** flag is set (checked), opens a standard file-selection dialog. The path entered through the dialog is automatically assigned as the value of the text field.

If invoked on a text field in which the **fileSelect** flag is clear (unchecked), an exception is thrown.

**Parameter**

None

**Returns**

Nothing

**Example**

The following code references a text field with the file select flag checked. This is a mouse up action of a button field.

```
var f = this.getField("resumeField");
f.browseForFileToSubmit();
```

**buttonGetCaption**

5.0			
-----	--	--	--

Gets the caption associated with a **button**.

Use **field.buttonSetCaption** to set the caption.

**Parameter**


---

<b>nFace</b>	(optional) If specified, gets a caption of the given type: 0: (default) normal caption 1: down caption 2: rollover caption
--------------	---

---

**Returns**

The caption string associated with the button.

**Example**

This example places pointing arrows to the left and right of the caption on a button field with icon and text.

```
// a mouse enter event
event.target.buttonSetCaption("> " + event.target.buttonGetCaption()
    + " <=");

// a mouse exit event
var str = event.target.buttonGetCaption();
str = str.replace(/=> | <=/g, "");
event.target.buttonSetCaption(str);
```

The same effect can be created by having the same icon for rollover, and have the same text, with the arrows inserted, for the rollover caption. This approach would be slower and cause the icon to flicker. The above code gives a very fast and smooth rollover effect because only the caption is changed, not the icon.

## buttonGetIcon

5.0			
-----	--	--	--

Gets the [Icon Generic Object](#) of a specified type associated with a button.

See also the [buttonSetIcon](#) method for assigning an icon to a button.

### Parameter

---

<b>nFace</b>	(optional) If specified, gets an icon of the given type: 0: (default) normal icon 1: down icon 2: rollover icon
--------------	--

---

### Returns

The [Icon Generic Object](#).


### Example

```
// Swap two button icons.  
var f = this.getField("Button1");  
var g = this.getField("Button2");  
var temp = f.buttonGetIcon();  
f.buttonSetIcon(g.buttonGetIcon());  
g.buttonSetIcon(temp);
```

See also [buttonSetIcon](#) and [buttonImportIcon](#).



## buttonImportIcon

3.01			
------	--	--	---

Imports the appearance of a button from another PDF file. If neither of the optional parameters are passed, the method prompts the user to select a file available on the system.

See also [buttonGetIcon](#), [buttonSetIcon](#), [addIcon](#), [getIcon](#), [importIcon](#), and [removeIcon](#).

### Parameter

<b>cPath</b>	(optional, version 5.0) The device-independent pathname for the file. See Section 3.10.1 of the <a href="#">PDF Reference</a> for a description of the device-independent pathname format. Beginning with version 6.0, Acrobat will first attempt to open <b>cPath</b> as a PDF. On failure, Acrobat will try to convert <b>cPath</b> to PDF from one of the known graphics formats (BMP, GIF, JPEG, PCX, PNG, TIFF) and then import the converted file as a button icon.
<b>nPage</b>	(optional, version 5.0) The 0-based page number from the file to turn into an icon. The default is 0.

### Returns

An integer, as follows:

- 1: The user cancelled the dialog
- 0: No error
- 1: The selected file couldn't be opened
- 2: The selected page was invalid

### Example (Acrobat 5.0)

It is assumed that we are connected to an employee information database. We communicate with the database using the [ADBC Object](#) and related objects. An employee's record is requested and three columns are utilized, *FirstName*, *SecondName* and *Picture*. The *Picture* column, from the database, contains a device-independent path to the employee's picture, stored in PDF format. The script might look like this:

```
var f = this.getField("myPicture");
f.buttonSetCaption(row.FirstName.value + " " + row.LastName.value);
if (f.buttonImportIcon(row.Picture.value) != 0)
    f.buttonImportIcon("/F/employee/pdfs/NoPicture.pdf");
```

The button field "myPicture" has been set to display both icon and caption. The employee's first and last names are concatenated to form the caption for the picture. Note that if there is an error in retrieving the icon, a substitute icon could be imported.

## buttonSetCaption

5.0	ⓓ		
-----	---	--	--

Sets the caption associated with a button.

Use [buttonGetCaption](#) to get the current caption.

See [buttonAlignX](#), [buttonAlignY](#), [buttonFitBounds](#), [buttonPosition](#), [buttonScaleHow](#), [buttonScaleWhen](#) on for details on how the icon and caption are placed on the button face.

### Parameter

<b>cCaption</b>	The caption associated with the button.
<b>nFace</b>	(optional) If specified, sets a caption of the given type: 0: (default) normal caption 1: down caption 2: rollover caption

### Returns

Nothing

### Example

```
var f = this.getField("myButton");
f.buttonSetCaption("Hello");
```

## buttonSetIcon

5.0	ⓓ		
-----	---	--	--

Sets the icon associated with a button.

See [buttonAlignX](#), [buttonAlignY](#), [buttonFitBounds](#), [buttonPosition](#), [buttonScaleHow](#), [buttonScaleWhen](#) on for details on how the icon and caption are placed on the button face.

Use either `field.buttonGetIcon` or `doc.getIcon` to get an [Icon Generic Object](#) that can be use for the `oIcon` parameter of this method.

### Parameter

<b>oIcon</b>	The <a href="#">Icon Generic Object</a> associated with the button.
--------------	---

---

<b>nFace</b>	(optional) If specified, sets an icon of the given type: 0: (default) normal icon 1: down icon 2: rollover icon
--------------	--

---

**Returns**

Nothing

**Example**

This example takes every named icon in the document and creates a listbox using the names. Selecting an item in the listbox sets the icon with that name as the button face of the field "myPictures". What follows is the mouse up action of the button field "myButton".

```
var f = this.getField("myButton")
var aRect = f.rect;
aRect[0] = f.rect[2];           // place listbox relative to the
aRect[2] = f.rect[2] + 144;    // position of "myButton"
var myIcons = new Array();
var l = addField("myIconList", "combobox", 0, aRect);
l.textSize = 14;
l.strokeColor = color.black;
for (var i = 0; i < this.icons.length; i++)
    myIcons[i] = this.icons[i].name;
l.setItems(myIcons);
l.setAction("Keystroke",
    'if (!event.willCommit) {\r\t'
    + 'var f = this.getField("myPictures");\r\t'
    + 'var i = this.getIcon(event.change);\r\t'
    + 'f.buttonSetIcon(i);\r\t'
    + '}'');
```

The named icons themselves can be imported into the document through an interactive scheme, such as the example given in [addIcon](#) or through a batch sequence.

See also [buttonGetCaption](#) for a more extensive example.

**checkThisBox**

5.0			
-----	--	--	--

Checks or unchecks the specified widget. Only **checkboxes** can be unchecked. A **radiobutton** cannot be unchecked using this method, but if its default state is unchecked (see [defaultIsChecked](#)) it can be reset to the unchecked state using [doc.resetForm](#).

**NOTE:** For a set of **radiobuttons** that do not have duplicate export values, you can set the **value** to the export value of the individual widget that should be checked (or pass an empty string if none should be).

**Parameters**

<b>nWidget</b>	The 0-based index of an individual <b>checkbox</b> or <b>radiobutton</b> widget for this field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). Every entry in the <b>Fields</b> panel has a suffix giving this index; for example, MyField #0.
<b>bCheckIt</b>	(optional) Whether the widget should be checked. The default is <b>true</b> .

**Returns**

Nothing

**Example**Check a **checkbox**.

```
// check the box "ChkBox"
var f = this.getField("ChkBox");
f.checkThisBox(0,true);
```

**clearItems**

4.0	ⓓ		
-----	---	--	--

Clears all the values in a **listbox** or **combobox**.Related methods and properties include [numItems](#), [getItemAt](#), [deleteItemAt](#), [currentValueIndices](#), [insertItemAt](#) and [setItems](#).**Parameters**

None

**Returns**

Nothing

**Example**

Clear the field "myList" of all items.

```
this.getField("myList").clearItems();
```

## defaultIsChecked

5.0			
-----	--	--	--

Sets the specified widget to be checked or unchecked by default.

**NOTE:** For a set of radio buttons that do not have duplicate export values, you can set the [defaultValue](#) to the export value of the individual widget that should be checked by default (or pass an empty string if none should be).

### Parameters

<b>nWidget</b>	The 0-based index of an individual <b>radiobutton</b> or <b>checkbox</b> widget for this field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). Every entry in the <b>Fields</b> panel has a suffix giving this index (for example, MyField #0).
<b>bIsDefaultChecked</b>	(optional) When <b>true</b> (the default) the widget should be checked by default (for example, when the field gets reset). When <b>false</b> , it should be unchecked by default.

### Returns

**true** on success.

### Example

Change the default of "ChkBox" to checked, then reset the field to reflect the default value.

```
var f = this.getField("ChkBox");
f.defaultIsChecked(0,true);
this.resetForm(["ChkBox"]);
```

## deleteItemAt

4.0	ⓓ		
-----	---	--	--

Deletes an item in a **combobox** or a **listbox**.

For a **listbox**, if the current selection is deleted the field no longer has a current selection. Having no current selection can lead to unexpected behavior by this method if it is again invoked without parameters on this same field; there is no current selection to delete. It is important, therefore, to make a new selection so that this method will behave as documented. A new selection can be made by using the [currentValueIndices](#).

**Parameters**


---

<b>nIdx</b>	(optional) The 0-based index of the item in the list to delete. If not specified, the currently selected item is deleted.
-------------	---

---

**Returns**

Nothing

**Example**

Delete the current item in the list, then select the top item in the list.

```
var a = this.getField("MyListBox");
a.deleteItemAt();           // delete current item, and...
a.currentValueIndices = 0;  // select top item in list
```

**getArray**

3.01			
------	--	--	--

Gets the array of terminal child fields (that is, fields that can have a value) for this [Field Object](#), the parent field.

**Parameters**

None

**Returns**An array of [Field Objects](#)**Example**

This example makes a calculation of the values of the child fields of the parent field.

```
// f has 3 children: f.v1, f.v2, f.v3
var f = this.getField("f");
var a = f.getArray();
var v = 0.0;
for (j =0; j < a.length; j++) v += a[j].value;
// v contains the sum of all the children of field "f"
```

**getItemAt**

3.01			
------	--	--	--

Gets the internal value of an item in a **combobox** or a **listbox**.

The number of items in a list can be obtained from **field.numItems**. See also [insertItemAt](#), [deleteItemAt](#), [clearItems](#), [currentValueIndices](#) and [setItems](#).

**Parameters**

<b>nIdx</b>	The 0-based index of the item in the list to obtain, or -1 for the last item in the list.
<b>bExportValue</b>	(optional, version 5.0) Whether to return an export value. <ul style="list-style-type: none"> <li>• When <b>true</b>, (the default) if the requested item has an export value, returns the export value. If there is no export value, returns the item name.</li> <li>• When <b>false</b>, the method returns the item name.</li> </ul>

**Returns**

The export value or name of the specified item.

**Example**

In the two examples that follow, assume there are three items on "myList": "First", with an export value of 1; "Second", with an export value of 2; and "Third" with no export value.

```
// returns value of first item in list, which is 1
var f = this.getField("myList");
var v = f.getItemAt(0);
```

The following example illustrates the use of the second optional parameter.

```
for (var i=0; i < f.numItems; i++)
    console.println(f.getItemAt(i,true) + ": " +
        f.getItemAt(i,false));
```

The output to the console reads:

```
1:      First
2:      Second
Third:  Third
```

Thus, by putting the second parameter to **false** the item name (face value) can be obtained, even when there is an export value.

**getLock**

6.0				
-----	--	--	--	--

Gets a [Lock Object](#), a generic object that contains the lock properties of a **signature** field.

See also [setLock](#) of the [Field Object](#).

**Parameters**

None

**Returns**

The [Lock Object](#) for the field.

## insertItemAt

3.01	Ⓓ		
------	---	--	--

Inserts a new item into a **combobox** or a **listbox**.

Related methods and properties include [numItems](#), [getItemAt](#), [deleteItemAt](#), [clearItems](#), [currentValueIndices](#) and [setItems](#).

### Parameters

<b>cName</b>	The item name that will appear in the form.
<b>cExport</b>	(optional) The export value of the field when this item is selected. If not provided, the <b>cName</b> is used as the export value.
<b>nIdx</b>	(optional) The index in the list at which to insert the item. If 0 (the default), the new item is inserted at the top of the list. If -1, the new item is inserted at the end of the list.

### Returns

Nothing

### Example

```
var l = this.getField("myList");
l.insertItemAt("sam", "s", 0); /* inserts sam to top of list l */
```

## isBoxChecked

5.0			
-----	--	--	--

Determines whether the specified widget is checked.

**NOTE:** For a set of **radiobuttons** that do not have duplicate export values, you can get the **value**, which is equal to the export value of the individual widget that is currently checked (or returns an empty string, if none is).

### Parameters

<b>nWidget</b>	The 0-based index of an individual <b>radiobutton</b> or <b>checkbox</b> widget for this field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order).  Every entry in the <b>Fields</b> panel has a suffix giving this index, for example MyField #0.
----------------	---



**Returns**

**true** if the specified widget is currently checked, **false** otherwise.

**Example**

```
var f = this.getField("ChkBox");
if (f.isBoxChecked(0))
    app.alert("The Box is Checked");
else
    app.alert("The Box is not Checked");
```

**isDefaultChecked**

5.0			
-----	--	--	--

Determines whether the specified widget is checked by default (for example, when the field gets reset).

**NOTE:** For a set of radio buttons that do not have duplicate export values, you can get the **defaultValue**, which is equal to the export value of the individual widget that is checked by default (or returns an empty string, if none is).

**Parameters**

<b>nWidget</b>	The 0-based index of an individual <b>radiobutton</b> or <b>checkbox</b> widget for this field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). Every entry in the <b>Fields</b> panel has a suffix giving this index, for example MyField #0.
----------------	---

**Returns**

**true** if the specified widget is checked by default, **false** otherwise.

**Example**

```
var f = this.getField("ChkBox");
if (f.isDefaultChecked(0))
    app.alert("The Default: Checked");
else
    app.alert("The Default: Unchecked");
```

**setAction**

5.0	Ⓓ		ⓧ
-----	---	--	---

Sets the JavaScript action of the field for a given trigger.

Related methods are [bookmark.setAction](#), [doc.setAction](#), [doc.addScript](#), [doc.setPageAction](#).

**NOTE:** This method will overwrite any action already defined for the chosen trigger.

**Parameters**

<b>cTrigger</b>	A string that sets the trigger for the action. Values are: MouseUp MouseDown MouseEnter MouseExit OnFocus OnBlur Keystroke Validate Calculate Format For a <b>listbox</b> , use the <b>Keystroke</b> trigger for the Selection Change event.
<b>cScript</b>	The JavaScript code to be executed when the trigger is activated.

**Returns**

Nothing

**Example**

This example sets up a button field with a mouse up action.

```
var f = this.addField("actionField", "button", 0 , [20, 100, 100, 20]);
f.setAction("MouseUp", "app.beep(0);");
f.delay = true;
  f.fillColor = color.ltGray;
  f.buttonSetCaption("Beep");
  f.borderStyle = border.b;
  f.lineWidth = 3;
  f.strokeColor = color.red;
  f.highlight = highlight.p;
f.delay = false;
```

**setFocus**

4.05			
------	--	--	--

Sets the keyboard focus to this field. This can involve changing the page that the user is currently on or causing the view to scroll to a new position in the document. This method automatically brings the document that the field resides in to the front, if it is not already there.

See also the [bringToFront](#).

**Parameters**

None

**Returns**

Nothing

**Example**

Search for a certain open doc, then focus in on the field of interest. This script uses `app.activeDocs`, which requires the documents to be **disclosed** to be `true`, or the script to be run during console, batch or menu events.

```
var d = app.activeDocs;
for (var i = 0; i < d.length; i++) {
  if (d[i].info.Title == "Response Document") {
    d[i].getField("name").value="Enter your name here: "
    // also brings the doc to front.
    d[i].getField("name").setFocus();
    break;
  }
}
```

**setItems**

4.0	Ⓓ		
-----	---	--	--

Sets the list of items for a **combobox** or a **listbox**.

Related methods and properties include [numItems](#), [getItemAt](#), [deleteItemAt](#), [currentValueIndices](#) and [clearItems](#).

**Parameters**


---

<b>oArray</b>	<p>An array in which each element is either an object convertible to a string or another array.</p> <ul style="list-style-type: none"> <li>● For an element that can be converted to a string, the user and export values for the list item are equal to the string.</li> <li>● For an element that is an array, the array must have two sub-elements convertible to strings, where the first is the user value, and the second is the export value.</li> </ul>
---------------	---

---

**Returns**

Nothing




**Examples**

```
var l = this.getField("ListBox");
l.setItems(["One", "Two", "Three"]);
```

```
var c = this.getField("StateBox");
c.setItems([["California", "CA"], ["Massachusetts", "MA"],
            ["Arizona", "AZ"]]);
```

```
var c = this.getField("NumberBox");
c.setItems(["1", 2, 3, ["PI", Math.PI]]);
```

## setLock

6.0				
-----	---	---	---	--

Controls which fields are to be locked when a signature is applied to this **signature** field. Once the fields are locked no modifications can be done to the fields. When the signature is cleared, all the fields that were locked down are unlocked. The property settings can be obtained using [getLock](#).

**NOTES:** (Security ): The method can be executed during a batch, application initialization, console, or menu events. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

This method cannot be applied to a field that is in a document that is already signed. Not allowed in the Adobe Reader.

### Parameters

<b>oLock</b>	A <a href="#">Lock Object</a> containing the lock properties.
--------------	---

### Returns

**true** if succesful, **false** otherwise, or can throw an exception.

### Lock Object

A generic JS object containing lock properties. This object is passed to **field.setLock** and returned by **field.getLock** for a **signature** field. It contains the following properties.

Property	Type	Access	Description
<b>action</b>	String	R/W	The language independent name of the action. Values are: <b>All:</b> All fields in the document are to be locked. <b>Include:</b> Only the fields specified in fields are to be locked. <b>Exclude:</b> All fields except those specified in fields are to be locked.
<b>fields</b>	Array of Strings	R/W	An array of strings containing the field names. Required if the value of action is <b>Include</b> or <b>Exclude</b> .

## signatureGetModifications

7.0				
-----	--	--	--	--

This method returns an object containing information on the modifications that have been made to the document after the signature field was signed. The modifications reports only the difference between the current and signed state of the document. Transient objects, for example objects added after the signature but then subsequently deleted, will not be reported.

### Parameters

None

### Returns

An object containing modification information. The object has the following properties:

Property	Type	Description
<b>formFieldsCreated</b>	Array of Field Objects	Array of form fields created after signing.
<b>formFieldsDeleted</b>	Array of Generic Objects	Array of form fields deleted after signing. Each generic object in the array is a string of the form <b>name : type</b> .
<b>formFieldsFilledIn</b>	Array of Field Objects	Array of form fields filled in after signing.
<b>formFieldsModified</b>	Array of Field Objects	Array of form fields modified after signing. In this context, form field fill-in does not constitute modification.
<b>annotsCreated</b>	Array of Annot Objects	Array of annots created after signing. If the annot is transient (e.g., a dynamically created Popup) then the corresponding element of the array is a string of the form <b>author : name : type</b> .
<b>annotsDeleted</b>	Array of Generic Objects	Array of annots deleted after signing. Each generic object in the array is of the form <b>author : name : type</b> .
<b>annotsModified</b>	Array of Annot Objects	Array of annots modified after signing. If the annot is transient (e.g., a dynamically created Popup) then the corresponding element of the array is a string of the form <b>author : name : type</b> .
<b>numPagesCreated</b>	Integer	Number of pages added after signing.
<b>numPagesDeleted</b>	Integer	Number of pages deleted after signing.

Property	Type	Description
<b>numPagesModified</b>	Integer	Number of pages whose content has been modified after signing (add/delete/modify of annots/form fields are not considered as page modification for this purpose).
<b>spawnedPagesCreated</b>	Array of Strings	List of pages spawned after signing. For each spawned page, the name of the source template is provided.
<b>spawnedPagesDeleted</b>	Array of Strings	List of spawned pages deleted after signing. For each spawned page, the name of the source template is provided.
<b>spawnedPagesModified</b>	Array of Strings	List of spawned pages modified after signing. For each spawned page, the name of the source template is provided.

### Example

This example writes modification information back to the console.

```

var sigField = this.getField( "mySignature" );
var sigMods = sigField.signatureGetModifications();

var formFieldsCreated = sigMods.formFieldsCreated;
for( var i = 0; i < formFieldsCreated.length; i++ )
    console.println( formFieldsCreated[i].name );

var formFieldsDeleted = sigMods.formFieldsDeleted;
for( var i = 0; i < formFieldsDeleted.length; i++ )
    console.println( formFieldsDeleted[i].name );

var formFieldsFilledIn = sigMods.formFieldsFilledIn;
for( var i = 0; i < formFieldsFilledIn.length; i++ )
    console.println( formFieldsFilledIn[i].name );

var formFieldsModified = sigMods.formFieldsModified;
for( var i = 0; i < formFieldsModified.length; i++ )
    console.println( formFieldsModified[i].name );

var spawnedPages = sigMods.spawnedPagesCreated;
for( var i = 0; i < spawnedPages.length; i++ )
    console.println( spawnedPages[i] );

console.println( sigMods.numPagesDeleted );

```

## signatureGetSeedValue

6.0				
-----	--	--	--	--

Returns a [SeedValue Generic Object](#) that contains the seed value properties of a signature field. Seed values are used to control properties of the signature, including the signature appearance, reasons for signing, and the person.

See [signatureSetSeedValue](#).

### Parameters

None

### Returns

A [SeedValue Generic Object](#).

### Example

The following illustrates accessing the seed value for a signature field.

```
var f = this.getField( "sig0" );
var seedValue = f.signatureGetSeedValue();
// displays the seed value filter and flags
console.println( "Filter name:" + seedValue.filter);
console.println( "Flags:" + seedValue.flags);
// displays the certificate seed value constraints
var certSpec = seedValue.certspec;
console.println( "Issuer:" + certspec.issuer);
```

## signatureInfo

5.0		Ⓢ		
-----	--	---	--	--

Returns a [SignatureInfo Object](#) that contains the properties of the signature. The object is a snapshot of the signature that is taken at the time that this method is called. A security handler may specify additional properties that are specific to the security handler.

**NOTES:** (Security Ⓢ): There are no restrictions on when this method can be called, however, the specified security handler, `oSig`, may not always be available; see the method [security.getHandler](#) for details.

Some properties of a signature handler, for example, `certificates` (a property of the [SignatureInfo Object](#)), may return a `null` value until the signature is validated. Therefore, `signatureInfo` should be called again after [signatureValidate](#).

## Parameters

---

<b>oSig</b>	(optional) The <a href="#">SecurityHandler Object</a> to use to retrieve the signature properties. If not specified, the security handler is determined by user preferences: it is usually the handler that was used to create the signature.
-------------	---

---

## Returns

A [SignatureInfo Object](#) that contains the properties of the signature. This type of object is also used when signing signature fields, signing FDF objects, or with the **PDF.signatureValidate** method.

## Example

The following illustrates how to access signature info properties.

```
// get all info
var f = getField( "Signature1" );
f.signatureValidate();
var s = f.signatureInfo();
console.println( "Signature Attributes:" );
for(i in s) console.println( i + " = " + s[i] );

// get particular info
var f = this.getField("Signature1"); // uses the ppklite sig handler
var Info = f.signatureInfo();
// some standard signatureInfo properties
console.println("name = " + Info.name);
console.println("reason = " + Info.reason);
console.println("date = " + Info.date);

// additional signatureInfo properties from PPkLite
console.println("contact info = " + Info.contactInfo);

// get the certificate; first (and only) one
var certificate = Info.certificates[0];

// common name of the signer
console.println("subjectCN = " + certificate.subjectCN);
console.println("serialNumber = " + certificate.serialNumber);

// Display some information about this the distinguished name of signer
console.println("subjectDN.cn = " + certificate.subjectDN.cn);
console.println("subjectDN.o = " + certificate.subjectDN.o);
```



## signatureSetSeedValue

6.0	ⓓ	Ⓢ	ⓧ	
-----	---	---	---	--

Sets properties that are used when signing signature fields. The properties are stored in the signature field and are not altered when the field is signed, the signature is cleared, or when `resetForm` is called. Use `signatureGetSeedValue` to obtain the property settings.

**NOTES:** (Security Ⓢ): The method can be executed during a batch, application initialization, console, or menu events. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

Seed values cannot be set for author signatures. Author signatures are signatures with a `SignatureInfo Object` `mdp` property value of `allowNone`, `default`, or `defaultAndComments`.

Not allowed in the Adobe Reader.

### Parameters

---

<code>oSigSeedValue</code>	A <a href="#">SeedValue Generic Object</a> containing the signature seed value properties.
----------------------------	--

---

### Returns

Nothing

### SeedValue Generic Object

A generic JS object, passed to `field.signatureSetSeedValue` and returned by `field.signatureGetSeedValue`, which represents a signature seed value. It has the following properties:

Property	Type	Access	Description
<code>certspec</code>	object	R/W	A seed value <a href="#">CertificateSpecifier Generic Object</a> .
<code>filter</code>	String	R/W	The language independent name of the security handler to be used when signing.

Property	Type	Access	Description
<b>flags</b>	Number	R/W	Flags controlling which properties in this object are critical (1, required) and not critical (0, optional). The value should be set to the logical or of the following values: 1: if filter is critical, 2: if subFilter is critical, 4: if version is critical 8: if reasons field is critical. If this field is not present, interpretation of all attributes is optional.
<b>legalAttestations</b>	Array of Strings	R/W	(version 7.0) A list of legal attestations that the user is allowed to use when creating a MDP signature.
<b>mdp</b>	String	R/W	(version 7.0) The Modification Detection and Prevention (MDP) setting to use when signing the field. Values are unique identifiers, described in the table titled <a href="#">Modification Detection and Prevention (MDP) Values</a> . Note that <b>allowAll</b> results in MDP not being used for the signature, resulting in this not being an author signature, but rather a user signature.
<b>reasons</b>	Array of Strings	R/W	A list of reasons that the user is allowed to use when signing.
<b>subFilter</b>	Array of Strings	R/W	An array of acceptable formats to use for the signature. Refer to the signature info object's subFilter property for a list of known formats.
<b>timeStampSpec</b>	Object	R/W	(version 7.0) A <a href="#">Seed Value timeStamp Specifier Object</a> .
<b>version</b>	Number	R/W	The minimum version of the signature format dictionary that is required when signing.

### CertificateSpecifier Generic Object

This generic JS object contains the certificate specifier properties of a signature seed value. Used in the **certSpec** property of the [SeedValue Generic Object](#). This objects contains the following properties:

Property	Type	Access	Description
<b>subject</b>	Array of <a href="#">Certificate Object</a>	R/W	Array of <a href="#">Certificate Objects</a> that are acceptable for signing. <b>NOTE:</b> If specified, the signing certificate must be an exact match with one of the certificates in this array.

Property	Type	Access	Description
<b>issuer</b>	Array of <a href="#">Certificate Object</a>	R/W	Array of <a href="#">Certificate Objects</a> that are acceptable for signing. <b>NOTE:</b> If specified, the signing certificate must be issued by a certificate that is an exact match with one of the certificates in this array
<b>oid</b>	Array of Strings	R/W	Array of strings that contain Policy OIDs that must be present in the signing certificate. This property is only applicable of the <b>issuer</b> property is present.
<b>url</b>	String	R/W	A URL that can be used to enroll for a new credential if a matching credential is not found.
<b>flags</b>	Number	R/W	Bit flags controlling which properties in this object are critical (1, required) and not critical (0, optional). The value should be set to the logical or of the following values: 1 if subject is critical, 2 if issuer is critical, 4 if oid is critical. If this field is not present, interpretation of all attributes is optional.

### Seed Value timeStamp Specifier Object

The properties of the seed value timeStamp specifier object are as follows:

Property	Type	Access	Description
<b>url</b>	String	R/W	URL of the timeStamp server providing RFC 3161 compliant timeStamp.
<b>flags</b>	Number	R/W	A bit flag controlling whether this property is critical (required) or not critical (optional). 0 if not critical 1 if critical The default is 0.

### Example 1

Sets the signing handler as PPKMS and the format as "adbe.pkcs7.sha1".

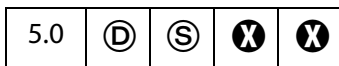
```
var f = this.getField( "sig0" );

f.signatureSetSeedValue( {
  filter: "Adobe.PPKMS",
  subFilter: ["adbe.pkcs7.sha1"],
  flags: 0x03 } );
```

**Example 2**

Sets the signing handler as PPKLite and the issuer of the signer's certificate as caCert. Both are mandatory seed values and signing will fail if either of constraint is not met.

```
var caCert = security.importFromFile("Certificate", "/C/CA.cer");
f.signatureSetSeedValue({
  filter: "Adobe.PPKLite",
  certspec: {
    issuer: [caCert],
    url: "http://www.ca.com/enroll.html",
    flags : 0x02
  },
  flags: 0x01
});
```

**signatureSign**

Signs the field with the specified security handler. See also [security.getHandler](#) and [securityHandler.login](#).

**NOTES:** (SecurityⓈ) This method can only be executed during batch, console, menu, or application initialization events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

Signature fields cannot be signed if they are already signed. Use [resetForm](#) to clear signature fields.

Not available in the Adobe Reader.

**Parameters**

<b>oSig</b>	Specifies the <a href="#">SecurityHandler Object</a> to be used to sign. Throws an exception if the specified handler does not support signing operations. Some security handlers require that the user be logged in before signing can occur. operations. Some security handlers require that the user be logged in before signing can occur. If <b>oSig</b> is not specified then this method will select a handler based on user preferences or by prompting the user if <b>bUI</b> is <b>true</b> .
<b>oInfo</b>	(optional) A <a href="#">SignatureInfo Object</a> specifying the writable properties of the signature. See also <a href="#">signatureInfo</a> .
<b>cDIPath</b>	(optional) The device-independent path to the file to save to following the application of the signature. If not specified, the file is saved back to its original location.

---

<b>bUI</b>	(optional, version 6.0) Whether the security handler should show user interface when signing. If <b>true</b> , <b>oInfo</b> and <b>cDIPath</b> are used as default values in the signing dialogs. If <b>false</b> (the default), the signing occurs without any user interface.
<b>cLegalAttest</b>	(optional, version 6.0) A string that can be provided when creating an author signature.  Author signatures are signatures where the <b>mdp</b> property of the <a href="#">SignatureInfo Object</a> has a value other than <b>allowAll</b> . When creating an author signature, the document is scanned for legal warnings and these warnings are embedded in the document. A caller can determine what legal warnings are found by first calling <b>doc.getLegalWarnings</b> . If warnings are to be embedded an author may wish to provide an attestation as to why these warnings are being applied to a document.

---

**Returns**

**true** if the signature was applied successfully, **false** otherwise.

**Example 1**

The following example signs the "Signature" field with the PPKLite signature handler:

```
var myEngine = security.getHandler( "Adobe.PPKLite" );
myEngine.login( "dps017", "/c/profile/dps.pfx" );
var f = this.getField("Signature");

// Sign the field
f.signatureSign( myEngine,
  { password: "dps017",           // provide password
    location: "San Jose, CA",     // ... see note below
    reason: "I am approving this document",
    contactInfo: "dpsmith@adobe.com",
    appearance: "Fancy"});
```

**NOTE:** In the above example, a password was provided. This may or may not have been necessary depending whether the **Password Timeout** had expired. The **Password Timeout** can be set programmatically by **securityHandler.setPasswordTimeout**.

**Example 2**

The following example illustrates signing an author signature field

```
var myEngine = security.getHandler( "Adobe.PPKLite" );
myEngine.login( "dps017", "/c/profile/dps.pfx" );

var f = this.getField( "AuthorSigFieldName" );
var s = { reason: "I am the author of this document",
         mdp: "allowNone" };
f.signatureSign({
```

```

    oSig: myEngine,
    oInfo: s,
    bUI: false,
    cLegalAttest: "Fonts are not embedded to reduce file size"
  });

```

## signatureValidate

5.0	Ⓓ			
-----	---	--	--	--

Validates and returns the validity status of the signature in a **signature** field. This routine can be computationally expensive and take a significant amount of time depending on the signature handler used to sign the signature.

**NOTE:** There are no restrictions on when this method can be called, however, the parameter **oSig** will not always be available; see [security.getHandler](#) for details.

### Parameters

<b>oSig</b>	(optional) The security handler to be used to validate the signature. The value can be either a <a href="#">SecurityHandler Object</a> or a <a href="#">SignatureParameters Generic Object</a> . If this handler is not specified, the method uses the security handler returned by the signature's <b>handlerName</b> property.
<b>bUI</b>	(optional, version 6.0) When <b>true</b> , allows UI to be shown, if necessary, when validating the data file. UI may be used to select a validation handler if none is specified. The default is <b>false</b> .

### Returns

Returns the validity status of the signature. Validity values are:

- 1: Not a signature field
- 0: Signature is blank
- 1: Unknown status
- 2: Signature is invalid
- 3: Signature of document is valid, identity of signer could not be verified
- 4: Signature of document is valid and identity of signer is valid.

See the **status** and **statusText** properties of the [SignatureInfo Object](#).

## SignatureParameters Generic Object

A generic object with the following properties that specify security handlers to be used for validation by `field.signatureValidate`:

Property	Description
<code>oSecHdlr</code>	The security handler object to use to validate this signature
<code>bAltSecHdlr</code>	If <b>true</b> , an alternate security handler, selected based on user preference settings, may be used to validate the signature. The default is <b>false</b> , which means that the security handler returned by the signature's <code>handlerName</code> property is used to validate the signature. This parameter is not used if <code>oSecHdlr</code> is provided.

### Example

```
var f = this.getField("Signature1") // get signature field
var status = f.signatureValidate();
var sigInfo = f.signatureInfo();
if ( status < 3 )
    var msg = "Signature not valid! " + sigInfo.statusText;
else
    var msg = "Signature valid! " + sigInfo.statusText;
app.alert(msg);
```

## FullScreen Object

5.0	Ⓟ		
-----	---	--	--

The interface to fullscreen (presentation mode) preferences and properties. To acquire a `fullScreen` object, use `app.fs`.

## FullScreen Properties

### backgroundColor

The background color of the screen in full screen mode. See [Color Arrays](#) for details.

Type: Color Array

Access: R/W.

### Example

```
app.fs.backgroundColor = color.ltGray;
```

## clickAdvances

Whether a mouse click anywhere on the page will cause the viewer to advance one page.

Type: *Boolean*

Access: *R/W*.

## cursor

Determines the behavior of the mouse pointer in full screen mode. The convenience **cursor** object defines all the valid cursor behaviors:

Cursor Behavior	Keyword
Always hidden	<code>cursor.hidden</code>
Hidden after delay	<code>cursor.delay</code>
Visible	<code>cursor.visible</code>

Type: *Number*

Access: *R/W*.

### Example

```
app.fs.cursor = cursor.visible;
```

## defaultTransition

The default transition to use when advancing pages in full screen mode. Use [transitions](#) to obtain list of valid transition names supported by the viewer.

**No Transition** is equivalent to `app.fs.defaultTransition = ""`;

Type: *Number*

Access: *R/W*.

### Example

Put document into presentation mode

```
app.fs.defaultTransition = "WipeDown";
app.fs.isFullScreen = true;
```

## escapeExits

Whether the escape key can be used to exit full screen mode.

Type: *Boolean*

Access: *R/W*.



## isFullScreen

Puts the Acrobat viewer in fullscreen mode rather than regular viewing mode. This only works if there are documents open in the Acrobat viewer window.

**NOTE:** A PDF document being viewed from within a web browser cannot be put into fullscreen mode.

*Type: Boolean*                      *Access: R/W.*

### Example

```
app.fs.isFullScreen = true;
```

In the above example, the Adobe Acrobat viewer is set to fullscreen mode. If **isFullScreen** was previously **false**, the default viewing mode would be set. The default viewing mode is defined as the original mode the Acrobat application was in before full screen mode was initiated.

## loop

Whether the document will loop around to the beginning of the document in response to a page advance (mouse click, keyboard, and/or timer generated) in full screen mode.

*Type: Boolean*                      *Access: R/W.*

## timeDelay

The default number of seconds before the page automatically advances in full screen mode. See [useTimer](#) to activate/deactivate automatic page turning.

*Type: Number*                      *Access: R/W.*

### Example

```
app.fs.timeDelay = 5;           // delay 5 seconds
app.fs.useTimer = true;        // activate automatic page turning
app.fs.usePageTiming = true;   // allow page override
app.fs.isFullScreen = true;    // go into fullscreen
```

## transitions

An array of strings representing valid transition names implemented in the viewer. **No Transition** is equivalent to setting [defaultTransition](#) to the empty string:

```
app.fs.defaultTransition = "";
```

*Type: Array*                      *Access: R.*

### Example

This script produces a listing of the currently supported transition names.

```
console.println("[ " + app.fs.transitions + " ]");
```

## usePageTiming

Whether automatic page turning will respect the values specified for individual pages in full screen mode. Set transition properties of individual pages using [setPageTransitions](#).

Type: Boolean

Access: R/W.

## useTimer

Whether automatic page turning is enabled in full screen mode. Use [timeDelay](#) to set the default time interval before proceeding to the next page.

Type: Boolean

Access: R/W.

---

## Global Object

This is a static JavaScript object that allows you to share data between documents and to have data be persistent across sessions. Such data is called *persistent global data*. Global data-sharing and notification across documents is done through a *subscription* mechanism, which allows you to monitor global data variables and report their value changes across documents.

### Creating Global Properties

You can specify global data by adding properties to the global object. The property type can be a String, a Boolean, or a Number.

For example, to add a variable called "radius" and to allow all document scripts to have access to this variable, the script simply defines the property:

```
global.radius = 8;
```

The global variable "radius" is now known across documents throughout the current viewer session. Suppose two files, A.pdf and B.pdf, are open in the viewer, and the global declaration is made in A.pdf. From within either file (A.pdf or B.pdf) you can calculate the volume of a sphere using **global.radius**:

```
var V = (4/3) * Math.PI * Math.pow(global.radius, 3);
```

In either file, you obtain the same result, 2144.66058. If the value of **global.radius** changes and the script is executed again, the value of **V** changes accordingly.

## Deleting Global Properties

To delete a variable or a property from the `global` object, use the `delete` operator to remove the defined property. For information on the reserved JavaScript keyword `delete`, see [Core JavaScript 1.5 Documentation](#).

For example, to remove the `global.radius` property, call the following script:

```
delete global.radius
```

## Global Methods

### setPersistent

3.01	Ⓟ		
------	---	--	--

Controls whether a specified variable is persistent across invocations of Acrobat.

Persistent global data only applies to variables of type Boolean, Number, or String. Acrobat 6.0 places a 2-4k limit for the maximum size of the global persistent variables. Any data added to the string after this limit is dropped.

The global variables that are persistent are stored upon application exit in the `glob.js` file located in the user's folder for `Folder Level JavaScripts`, and re-loaded at application start. There is a 2-4k limit on the size of this file, for Acrobat 6.0 or later.

It is recommended that JavaScript developers building scripts for Acrobat, use a naming convention when specifying persistent global variables. For example, you could name all your variables `"myCompany_name"`. This will prevent collisions with other persistent global variable names throughout the documents.

#### Parameters

<b>cVariable</b>	The variable (global property) for which to set persistence.
<b>bPersist</b>	When <b>true</b> , the property will exist across Acrobat Viewer sessions. When <b>false</b> (the default) the property will be accessible across documents but not across the Acrobat Viewer sessions.

#### Returns

Nothing

#### Example

For example, to make the "radius" property persistent and accessible for other documents you could use:

```
global.radius = 8; // declare radius to be global
global.setPersistent("radius", true); // now say it's persistent
```

The volume calculation, defined above, will now yield the same result across viewer sessions, or until the value of `global.radius` is changed.

## subscribe

5.0			
-----	--	--	--

Allows you to automatically update one or more fields when the value of the subscribed global variable changes. If the specified property is changed, even in another document, the specified function is called. Multiple subscribers are allowed for a published property.

### Parameters

<b>cVariable</b>	The global property.
<b>fCallback</b>	The function to call when the property is changed.

### Returns

Nothing

### Example

Suppose there are two files, `setRadius.pdf` and `calcVolume.pdf`, open in Acrobat or Reader.

- In `setRadius.pdf` there is a single button with the code:  
`global.radius = 2;`
- In `calcVolume.pdf` there is a Document Level JavaScript named **subscribe**:  

```
// In the Advanced > JavaScripts > Document JavaScripts
global.subscribe("radius", RadiusChanged);
function RadiusChanged(x)           // callback function
{
    var V = (4/3) * Math.PI * Math.pow(x,3);
    this.getField("MyVolume").value = V; // put value in text field
}
```
- Open both files in the Viewer, now, clicking on the button in `setRadius.pdf` file immediately gives an update in the text field "MyVolume" in `calcVolume.pdf` of 33.51032 (as determined by `global.radius = 2`).

The syntax of the callback function is as follows:

```
function fCallback(newval) {
// newval is the new value of the global variable you
// have subscribed to.
    < code to process the new value of the global variable >
}
```

## Icon Generic Object

This generic JS object is an opaque representation of a Form XObject appearance stored in `doc.icons`. It is used with [Field Objects](#) of type `button`. The `icon` object contains the following property:

Property	Type	Access	Description
<code>name</code>	string	R	The name of the icon. An icon may or may not have a name depending on whether it exists in the document level named icons tree.

## Icon Stream Generic Object

This generic JS object represents an icon stream. It is used by `app.addToolButton` and `collab.addStateModel`. It has the following properties:

Property	Description
<code>read(nBytes)</code>	A function which takes the number of bytes to read and returns a Hex encoded string. The data should be the icon representation as a 32 bit per pixel with 4 channels (ARGB) 8 bits per channel with the channels interleaved. If the icon has multiple layers, then the function may return the pixels for the topmost layer, followed by the next layer behind it, and so on.
<code>width</code>	The icon width in pixels.
<code>height</code>	The icon height in pixels.

The `util.iconStreamFromIcon` method can be used to convert an [Icon Generic Object](#) to an Icon Stream Generic Object.

## Identity Object

5.0		©	
-----	--	---	--

This is a static object that identifies the current user of the application.

**NOTES:** (Security<sup>Ⓢ</sup>): **Identity** object properties are only accessible during batch, console, menu, and application initialization events in order to protect the privacy of the user. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

---

## Identity Properties

### corporation

The corporation name that the user has entered in the identity preferences panel.

*Type: String*                      *Access: R/W.*

### email

The email address that the user has entered in the identity preferences panel.

*Type: String*                      *Access: R/W.*

### loginName

The login name as registered by the operating system.

*Type: String*                      *Access: R.*

### name

The user name that the user entered in the identity preferences panel.

*Type: String*                      *Access: R/W.*

### Example

The following can be executed in the console, or, perhaps, a folder level JavaScript.

```
console.println("Your name is " + identity.name);  
console.println("Your e-mail is " + identity.email);
```

---

## Index Object

5.0			
-----	--	--	--

This is a non-creatable object returned by various methods of the [Search Object](#) and [Catalog Object](#). The **index** object represents a Catalog-generated index. You use this object to perform various indexing operations using Catalog. You can find the status of the index with a search.

---

## Index Properties

### available

Whether the index is available for selection and searching. An index may be unavailable if a network connection is down or a CD-ROM is not inserted, or if the index administrator has brought the index down for maintenance purposes.

*Type: Boolean*

*Access: R.*

### name

The name of the index as specified by the index administrator at indexing time.

See [search.indexes](#), which returns an array of the index objects currently accessed by the search engine.

*Type: String*

*Access: R.*

### Example

This example enumerates all of the indexes and writes their names to the console.

```
for (var i = 0; i < search.indexes.length; i++) {  
    console.println("Index[" + i + "] = " + search.indexes[i].name);  
}
```

### path

The device-dependent path where the index resides. See Section 3.10.1, "File Specification Strings", in the [PDF Reference](#) for exact syntax of the path.

*Type: String*

*Access: R.*

## selected

Whether the index is to participate in the search. If **true**, the index will be searched as part of the query, if **false** it will not be. Setting or unsetting this property is equivalent to checking the selection status in the index list dialog.

Type: Boolean

Access: R/W.

## Index Methods

### build

6.0		Ⓢ	ⓧ	ⓧ	Ⓟ
-----	--	---	---	---	---

Builds the index associated with the **index** object using the Catalog plug-in. This method does not build a new index.

The index is built at the same location as the index file. If the index already exists, the included directories are re-scanned for changes and the index is updated. If the index does not exist, the UI pops up, and new index can be defined and build through the UI.

The index build is started immediately if Catalog is idle. Otherwise, it gets queued with Catalog.

**NOTE:** (SecurityⓈ, version 7.0) This method can only be executed during batch or console events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

### Parameters

<b>cExpr</b>	(optional) An expression to be evaluated once the build operation on the index is complete. Default is no expression. See the <a href="#">PDF Reference</a> , "JavaScript Action" for more details.
<b>bRebuildAll</b>	(optional) If <b>true</b> , a clean build is performed. The index is first deleted and then built. The default is <b>false</b> .

### Returns

A [CatalogJob Generic Object](#). The **CatalogJob** object can be used to check the job parameters and status.

### Example

```
/* Building an index */
if (typeof catalog != "undefined") {
    var idx = catalog.getIndex("/c/mydocuments/index.pdx");
    var job = idx.build("Done()", true);
    console.println("Status : ", job.status);
}
```



}

## Link Object

This object is used to set and get the properties and to set the JavaScript action of a link.

A **link** object is obtained from `doc.addLink` or `doc.getLinks`. See also, `doc.removeLinks`.

## Link Properties

### borderColor

6.0	Ⓧ		Ⓧ	
-----	---	--	---	--

The border color of a **link** object. See [Color Arrays](#) for information on defining color arrays and how colors are used with this property.

*Type: Array*

*Access: R/W.*

### borderWidth

6.0	Ⓧ		Ⓧ	
-----	---	--	---	--

The border width of the **link** object.

*Type: Integer*

*Access: R/W.*

### highlightMode

6.0	Ⓧ		Ⓧ	
-----	---	--	---	--

The visual effect to be used when the mouse button is pressed or held down inside an active area of a link. The valid values are:

None  
 Invert (the default)  
 Outline  
 Push

*Type: String*

*Access: R/W.*

**rect**

6.0	ⓓ		ⓧ	
-----	---	--	---	--

The rectangle in which the link is located on the page. Contains an array of four numbers, the coordinates in *rotated user space* of the bounding rectangle, listed in the following order: upper-left *x*, upper-left *y*, lower-right *x* and lower-right *y*.

*Type:* Array

*Access:* R/W.

**Link Methods****setAction**

6.0			ⓧ	
-----	--	--	---	--

Sets the specified JavaScript action for the MouseUp trigger for the **link** object.

**NOTE:** This method will overwrite any action already defined for this link.

**Parameters**

<b>cScript</b>	The JavaScript action to use.
----------------	-------------------------------

**Returns**

Nothing

**Marker Object**

A Marker object represents a named location in a media clip that identifies a particular time or frame number, similar to a track on an audio CD or a chapter on a DVD. Markers are defined by the media clip itself.

A Marker object can be obtained from the **Markers.get ()** method.

## Marker Object Properties

### frame

6.0				
-----	--	--	--	--

A frame number, where 0 represents the beginning of media. For most players, markers have either a frame or a time value, but not both.

*Type: Number*

*Access: R.*

### index

6.0				
-----	--	--	--	--

An arbitrary index number assigned to this marker. Markers have sequential index numbers beginning with 0, but these index numbers may not be in the same order that the markers appear in the media.

*Type: Number*

*Access: R.*

### name

6.0				
-----	--	--	--	--

The name of this marker. Each marker in a media clip has a unique name.

*Type: String*

*Access: R.*

### Example

Get a marker by its index, then print the name of the marker to the console.

```
// assume player is a MediaPlayer object
var markers = player.markers;
// get marker with index of 2
var marker = g.get( { index: 2 } );
console.println( "The marker with index of " + marker.index
    + ", has a name of " + marker.name );
```

### time

6.0				
-----	--	--	--	--

A time in seconds, where 0 represents the beginning of media. For most players, markers have either a frame or a time value, but not both.

*Type: Number**Access: R.***Example**

Get a named marker, then print the time in seconds from the beginning of the media, of that marker.

```
// assume player is a MediaPlayer object
var markers = player.markers;
// get marker with name of "Chapter 1"
var marker = g.get( { name: "Chapter 1" } );
console.println( "The named marker \"Chapter 1\", occurs at time "
    + marker.time);
```

---

**Markers Object**

The [markers](#) property of a MediaPlayer is a Markers object which represents all of the markers found in the media clip currently loaded into the player. A marker is a named location in a media clip that identifies a particular time or frame number, similar to a track on an audio CD or a chapter on a DVD. Markers are defined by the media clip itself.

The constructor is `app.media.Markers`.

---

**Markers Object Properties****player**

6.0				
-----	--	--	--	--

The [MediaPlayer Object](#) that this Markers object belongs to.

*Type: [MediaPlayer Object](#) Access: R.*

---

**Markers Object Methods****get**

6.0				
-----	--	--	--	--

The `Markers.get()` method looks up a marker by name, index number, time in seconds, or frame number, and returns the [Marker Object](#) representing the requested marker. The

object parameter should contain either a name, index, time, or frame property. A marker name can also be passed in directly as a string.

If a time or frame is passed in, the nearest marker at or before that time or frame is returned. If the time or frame is before any markers in the media, then null is returned.

### Parameters

An object or string representing the name, index number, time in seconds, or the frame number of the marker. The object parameter should contain either a name, index, time, or frame property. A marker name can also be passed in directly as a string.

### Returns

Marker Object or **null**

Marker index numbers are assigned sequentially starting with 0, and they are not necessarily in order by time or frame. In particular, note that these are not the same values that Windows Media Player uses for marker numbers. To find all of the available markers in a media clip, call **MediaPlayer.markers.get()** in a loop starting with **{index: 0}** and incrementing the number until **get()** returns **null**.

### Example:

This example counts the number of markers on the media clip.

```
var index, i = 0;
// assume player is a MediaPlayer object.
var m = player.markers;
while ( (index = m.get( { index: i } ) ) != null ) i++;
console.println("There are " + i + " markers.");
```

### Example:

```
// Get a marker by name, two different ways
var marker = player.markers.get( "My Marker" );
var marker = player.markers.get( { name: "My Marker" } );
// Get a marker by index
var marker = player.markers.get( { index: 1 } );
// Get a marker by time
var marker = player.markers.get( { time: 17.5 } );
// Get a marker by frame
var marker = player.markers.get( { frame: 43 } );
```

---

## MediaOffset Object

A MediaOffset represents a position in a MediaClip, either in terms of time or a frame count.

This position can either be relative to a named marker, or it can be an absolute position (i.e. relative to the beginning of the media).

The MediaOffset can be specified either as an object with the properties named below, or it can simply be a number, which is interpreted as **{time: number}**.

Some media formats (e.g. QuickTime) are time-based and others (e.g. Flash) are frame-based. A MediaOffset that specifies a time or frame must match the media format in use. If both time and frame are specified, the results are undefined: the incorrect one may be ignored, or a JavaScript exception may be thrown.

The MediaOffset object is used by `MediaPlayer.seek()`, `MediaPlayer.where()`, `MediaSettings.endAt` and `MediaSettings.startAt`.

## MediaOffset Object Properties

### frame

6.0				
-----	--	--	--	--

A frame number. If the **marker** property is also present, this frame number is relative to the specified marker and may be positive, negative, or zero. Otherwise, it is relative to the beginning of media and may not be negative. Note that `{ frame: 0 }` represents the beginning of media.

*Type: Number*

*Access: R/W.*

### marker

6.0				
-----	--	--	--	--

The name of a specific marker in the media.

*Type: String*

*Access: R/W.*

### time

6.0				
-----	--	--	--	--

A time in seconds, or **Infinity**. If the **marker** property is also present, this time is relative to the specified marker and is a nonnegative value, but not **Infinity**. Otherwise, the time is relative to the beginning of media and must not be negative. Note that the offset `{ time: 0 }` represents the beginning of media.

*Type: Number*

*Access: R/W.*

### Example

Below are examples of absolute and relative offsets

```
{ time: 5.4 } // offset 5.4 seconds from the beginning of media
{ marker: "Chapter 1", time: 17 } // 17 seconds after "Chapter 1"
```

These offsets can be used by the **MediaPlayer.seek()** method:

```
// assume player is a MediaPlayer object
player.seek({ time: 5.4 });
player.seek({ marker: "Chapter 1", time: 17 });
```

---

## MediaPlayer Object

A MediaPlayer object represents an instance of a multimedia player such as QuickTime, Windows Media Player, or others. Its **settings** and **events** properties let you manipulate the player from JavaScript code and handle events that the player fires. MediaPlayer is not part of a PDF file; it is a transient object created in memory when needed.

---

## MediaPlayer Object Properties

### annot

6.0				
-----	--	--	--	--

**MediaPlayer.annot** is a reference to the ScreenAnnot associated with a MediaPlayer. This property exists only for a MediaPlayer object that is connected to a ScreenAnnot. The property is set by **app.media.addStockEvents()** or by methods that call **addStockEvents()** indirectly, such as **app.media.openPlayer()**.

Type: *ScreenAnnot Object* Access: R/W.

### defaultSize

6.0				
-----	--	--	--	--

The **MediaPlayer.defaultSize** property is a read-only object containing the width and height of the MediaPlayer's MediaClip:

```
{ width: number, height: number }
```

If the media player is unable to provide this value, then defaultSize is **undefined**.

Type: *Object*

Access: *R*.

## doc

6.0				
-----	--	--	--	--

**MediaPlayer.doc** is a reference to the [Doc Object](#) that owns the MediaPlayer.

Type: *Object*

Access: *R*.

## events

6.0				
-----	--	--	--	--

The **MediaPlayer.events** property is a [Events Object](#) containing the event listeners that are attached to a MediaPlayer. See [Events Object](#) for details.

Type: *Events Object*

Access: *R/W*.

### Example

Create a media player, then modify the events of that player. The script is executed as a Rendition action with an associated rendition.

```
var events = new app.media.Events;
var player = app.media.createPlayer();
player.events.add({
    onReady: function() { console.println("The player is ready"); }
});
player.open();
```

## hasFocus

6.0				
-----	--	--	--	--

The **MediaPlayer.hasFocus** property is **true** if the media player is open and has the keyboard focus.

Type: *Boolean*

Access: *R*.

## id

6.0				
-----	--	--	--	--

The **MediaPlayer.id** property contains the player ID for the player software that this player is using. It is **undefined** if the player has not been opened. This player ID is the same value that is found in **PlayerInfo.id** for the media player software that implements this player.



*Type: Boolean**Access: R.***Example**

Print player id to the console

```
// assume args has been defined
var player = app.media.openPlayer( args )
console.println("player.id = " + player.id);
// in the console, this script could possibly print...
player.id = vnd.adobe.swname:ADBE_MCI
```

**innerRect**

6.0				
-----	--	--	--	--

The **MediaPlayer.innerRect** property is a rectangle array representing the player's inner rectangle on the screen. As with other such arrays in Acrobat JavaScript, the coordinates are in the order [left, top, right, bottom]. The rectangle does *not* include any window title, or other such gadgets around the edges of the player, but it does include the player controller if a controller is present. It is undefined if the player is not open.

For a docked media player, this rectangle is in device space and is read-only: It will throw an exception if you try to set it. Instead, use **MediaPlayer.triggerGetRect()** to cause a docked player to be resized. For a floating media player, the rectangle is in screen coordinates and is writable, but the user's security settings may override a value you set here. For example, if you try to move a floating media player offscreen, it may be forced back on-screen. This will not throw an exception. You can read this property after writing it to see if your value was overridden.

*Type: Array**Access: R or R/W.*See also, [outerRect](#).**isOpen**

6.0				
-----	--	--	--	--

**MediaPlayer.isOpen**, a boolean, is **true** if the media player is currently open. Use **MediaPlayer.open()** and **MediaPlayer.close()** to open or close a player.

*Type: Boolean**Access: R.*

## isPlaying

6.0				
-----	--	--	--	--

The **MediaPlayer.isPlaying** property is **true** if the media is currently playing. It is **false** if the player is not open, or if the media is paused, stopped, fast forwarding or rewinding, or in any other state.

Type: *Boolean*

Access: *R*.

## markers

6.0				
-----	--	--	--	--

**MediaPlayer.markers** is a collection of all the markers available for the current media.

See [Markers Object](#) for details of this property.

Type: *Markers Object*

Access: *R*.

### Example

See [Example 2](#) following **MediaPlayer.seek()** for an illustration of usage.

## outerRect

6.0				
-----	--	--	--	--

**MediaPlayer.outerRect** is a rectangle Array representing the player's outer rectangle on the screen. As with other such arrays in Acrobat JavaScript, the coordinates are in the order [left, top, right, bottom]. This rectangle includes any player controller, window title, and other such gadgets around the edges of the player. It is **undefined** if the player is not open.

For a docked media player, this rectangle is in device space and is read-only: It will throw an exception if you try to set it. Instead, use **MediaPlayer.triggerGetRect()** to cause a docked player to be resized. For a floating media player, the rectangle is in screen coordinates and is writable, but the user's security settings may override a value you set here. For example, if you try to move a floating media player offscreen, it may be forced back on-screen. This will not throw an exception. You can read this property after writing it to see if your value was overridden.

Type: *Array*

Access: *R or R/W*.

See also [innerRect](#).

## page

6.0				
-----	--	--	--	--

**MediaPlayer.page** is the page number in which a docked media player appears. It is **undefined** for players that are not docked. A docked media player can be moved to another page by changing its **page** property, and this triggers a **GetRect** (see [onGetRect](#)) event.

Type: *Number*

Access: *R/W*.

### Example

Play a media clip on page 1 (base zero). The placement of the media player on page 1 is the same the **ScreenAnnot** on page 0.

```
var player = app.media.openPlayer({
    rendition: this.media.getRendition( "myClip" ),
    annot: this.media.getAnnot({ nPage:0, cAnnotTitle:"myScreen" }),
    settings: { windowType: app.media.windowType.docked }
});
player.page = 1;
```

See [onGetRect](#) and [triggerGetRect](#) for variations on this same example.

## settings

6.0				
-----	--	--	--	--

**MediaPlayer.settings** includes all of the settings that are used to create a **MediaPlayer**. See [MediaSettings Object](#) for a complete list.

**NOTE:** In Acrobat 6.0, changing a property in **MediaPlayer.settings** after the player has been created has no effect. This may be changed in a future release to make these settings live. For compatibility with both current and future releases, avoid changing any settings properties while a player is open.

Type: *MediaSettings Object* Access: *R/W*.

## uiSize

6.0				
-----	--	--	--	--

**MediaPlayer.uiSize** is an array containing the size of the controller of the player for each edge of the player, in the same order as a window rectangle: [ left, top, right, bottom ]. Each of these values is normally a positive value or zero. These values do not include window gadgets such as title bars, only the controller.

This property is not available until the Ready event is fired (see [onReady](#) and [afterReady](#)). Unlike most MediaPlayer properties, it is permissible to read it during an “on” event method such as [onReady](#).

Type: Array

Access: R.

### Example

Get the **uiSize** of the player. This code is executed as a Rendition action event.

```
var args = {
  events: {
    onReady: function () {
      console.println("uiSize = " + player.uiSize );
    }
  }
};
var player = app.media.openPlayer(args);
```

## visible

6.0				
-----	--	--	--	--

The **MediaPlayer.visible** property controls whether the player is visible. Unlike **MediaPlayer.settings.visible**, this property takes effect immediately. If the player is not open, reading this property returns **undefined** and setting it throws an exception.

Setting this property may fire events. For example, if the player is visible and has the focus, making it invisible fires a Blur event.

Type: Boolean

Access: R/W.

### Example

Play the audio *only* of a video clip

```
// assume a definition of args
var player = app.media.openPlayer(args);
player.visible = false;
```

---

## MediaPlayer Object Methods

### close

6.0				
-----	--	--	--	--

Closes the media player if it is open. Does nothing (and is not an error) if the player is closed.

The `eReason` parameter should be a value from the `app.media.closeReason` enumeration. This value is passed through to the `event.media.closeReason` property for the Close event (see `onClose` and `afterClose`) that the `close()` method fires.

If the player has the keyboard focus, a Blur event (`onBlur/afterBlur`) is fired before the Close event. Other events, such as Status (`onStatus/afterStatus`) and Stop (`onStop/afterStop`), may also be fired depending on the particular media player.

**Parameters**

---

<code>eReason</code>	<code>eReason</code> is a value from the <code>app.media.closeReason</code> enumeration.
----------------------	--

---

**Returns**

Nothing

**open**

6.0				
-----	--	--	--	--

The `MediaPlayer.open()` method attempts to open the media player as specified by `MediaPlayer.settings`. If the player is already open, an exception is thrown. If the player was previously opened and then closed, `open()` may be called to open the player again. This uses the same JavaScript object as before but opens a new instance of the actual media player (e.g. the new player does not remember the playback position from the old player).

For a docked player, a `GetRect` event (`onGetRect`) is fired when the player is opened.

If `MediaPlayer.settings.autoPlay` is `true` (which it is by default), then playback begins and a `Play` event (`onPlay/afterPlay`) is fired.

The `open()` method may result in a security prompt dialog depending on the user's settings. This may also result in events being fired to other media players, screen annots, or other objects. For example, if another media player has the keyboard focus, it will receive a `Blur` event (`onBlur/afterBlur`).

If `bAllowSecurityUI` is `false`, then `open()` never displays a security prompt, but returns a failure code instead.

For a media player in a floating window, additional security checks are made against the user's settings. For example, the user may specify that title bars are required on all floating player windows. If `MediaPlayer.settings.floating` contains options that the user does not allow, then `bAllowFloatOptionsFallback` controls what happens. If it is `false`, playback is disallowed and an error code is returned. If it is `true`, then the options in `MediaPlayer.settings.floating` are changed as needed to conform to the user's security settings, and then `open()` proceeds with those changed settings.

The return value is an object which currently contains one property, **code**, which is a result code from the **app.media.openCode** enumeration. If your PDF is opened in a future version of Acrobat, there may be additional properties in this object, or a code value added in that future version. Be sure to handle any such values gracefully.

**Parameters**

<b>bAllowSecurityUI</b>	(optional) The default is <b>true</b> . See the description of this parameter given above.
<b>bAllowFloatOptionsFallback</b>	(optional) The default is <b>true</b> . See the description of this parameter given above.

**Returns**

An object with a **code** property

**Example**

See ["Example 1" on page 152](#) for an example of usage.

**pause**

6.0				
-----	--	--	--	--

Pauses playback of the current media and fires a Pause event ([onPause/afterPause](#)). The Pause event may occur during the **pause ()** call or afterward, depending on the player.

The **pause ()** method has no effect if the media is already paused or stopped, or if playback has not yet started or has completed. Not every media player supports **pause ()**, and not every media format supports it; in particular, most streaming formats do not support **pause ()**. Players may either throw an exception or silently ignore **pause ()** in these cases.

**Parameters**

None

**Returns**

Nothing

**Example**

See [Example 2](#) following the **seek ()** method below for an example of usage.

## play

6.0				
-----	--	--	--	--

Starts playback of the current media and fires a Play event ([onPlay/afterPlay](#)). The Play event may occur during the **play ()** call or afterward, depending on the player.

If the media is already playing, it continues playing and no event is fired. If it is paused, rewinding, or fast forwarding, it resumes playback at the current position. If it is stopped, either at the beginning or end of media, playback starts from the beginning.

### Parameters

None

### Returns

Nothing

### Example

See [Example 2](#) following the **seek ()** method below for an example of usage.

## seek

6.0				
-----	--	--	--	--

Sets the current media's playback location to the position described by the [MediaOffset Object](#) contained in **oMediaOffset**.

If the media is playing, it continues playing at the new location. If the media is paused, it moves to the new location and remains paused there. If the media is stopped, the result will vary depending on the player.

Different media players handle seek errors in different ways: Some ignore the error and others throw a JavaScript exception.

Most, but not all, media players fire a Seek event ([onSeek/afterSeek](#)) when a seek is completed.

The seek operation may take place during the execution of the **seek ()** method or later, depending on the player. If **seek ()** returns before the seek operation is completed and you call another player method before the seek is completed, the results will vary depending on the player.

### Parameters

---

<b>oMediaOffset</b>	A <a href="#">MediaOffset Object</a> , the properties of which indicate the playback location to be set.
---------------------	--

---

### Returns

Nothing

**Example 1**

```
// Rewind the media clip
player.seek({ time: 0 });

// Play starting from marker "First"
player.seek({ marker: "First" });

// Play starting five seconds after marker "One"
player.seek({ marker: "One", time: 5 });
```

**Example 2**

The following script randomly plays (famous) quotations. The media is an audio clip (.wma), which does support markers and scripts, of (famous) quotations. The **afterReady** listener counts the number of markers, one at the beginning of each quotation. At the end of each quotation, there is also an embedded command script, the **afterScript** listener watches for these commands, and if it is a "pause" command, it pauses the player.

```
var nMarkers=0;
var events = new app.media.Events;
events.add({
  // count the number of quotes in this audio clip, save as nMarkers
  afterReady: function()
  {
    var g = player.markers;
    while ( (index = g.get( { index: nMarkers } ) ) != null )
      nMarkers++;
  },
  // Each quote should be followed by a script, if the command is to
  // pause, then pause the player.
  afterScript: function( e ) {
    if ( e.media.command == "pause" ) player.pause();
  }
});
var player = app.media.openPlayer({
  rendition: this.media.getRendition( "myQuotes" ),
  settings: { autoPlay: false },
  events: events
});
// randomly choose a quotation
function randomQuote() {
  var randomMarker, randomMarkerName;
  console.println("nMarkers = " + nMarkers);
  // randomly choose an integer between 1 and nMarkers, inclusive
  randomMarker = Math.floor(Math.random() * 100) % ( nMarkers ) + 1;
  // indicate what quotation we are playing
  this.getField("Quote").value = "Playing quote " + randomMarker;
  // The marker names are "quote 1", "quote 2", "quote 3", etc.
  randomMarkerName = "quote " + randomMarker;
  // see the marker with the name randomMarkerName
```



```

        player.seek( { marker: randomMarkerName } );
        player.play();
    }

```

Action is initiated by the mouse up button action such as

```
try { randomQuote() } catch(e) {}
```

## setFocus

6.0				
-----	--	--	--	--

Sets the keyboard focus to the media player and fires a Focus event ([onFocus/afterFocus](#)). If another player or PDF object has the focus, that object receives a Blur event ([onBlur/afterBlur](#)). If the media player already has the focus, nothing happens. If the player is not open or not visible, an exception is thrown.

### Parameters

None

### Returns

Nothing

### Example

See ["Example 1" on page 152](#) for an example of usage.

## stop

6.0				
-----	--	--	--	--

Stops playback of the current media, if it is playing or paused, and fires a Stop event ([onStop/afterStop](#)). The Stop event may occur during execution of the `stop()` method or afterward, depending on the player. Does nothing if the media is not playing or paused.

Throws an exception if the player is not open.

After playback stops, the player sets the media position to either the beginning or end of media, depending on the player. If `MediaPlayer.play()` is called after this, playback starts at the beginning of media.

### Parameters

None

### Returns

Nothing

## triggerGetRect

6.0				
-----	--	--	--	--

Fires a GetRect event (see [onGetRect](#)) to cause a docked media player to be resized.

### Parameters

None

### Returns

Nothing

### Example

This example is similar to the one that follows [onGetRect](#). Page 0 has a series of (thumbnail-size) ScreenAnnots. Below is a typical Rendition action or mouse up button JavaScript action, when the action is executed, the media clip is resized and played.

```
var rendition = this.media.getRendition("Clip1");
var annot = this.media.getAnnot({ nPage:0,cAnnotTitle:"ScreenClip1" });
var player = app.media.openPlayer({
    rendition: rendition,
    annot: annot,
    settings: { windowType: app.media.windowType.docked },
    events: {
        onGetRect: function (e) {
            var width = e.media.rect[2] - e.media.rect[0];
            var height = e.media.rect[3] - e.media.rect[1];
            width *= 3; // triple width and height
            height *= 3;
            e.media.rect[0] = 36; // move left, upper to
            e.media.rect[1] = 36; // upper left-hand corner
            e.media.rect[2] = e.media.rect[0]+width;
            e.media.rect[3] = e.media.rect[1]+height;
            return e.media.rect; // return this
        }
    }
});
player.triggerGetRect(); // trigger the onGetRec event
```

## where

6.0				
-----	--	--	--	--

Reports the current media's playback location in a [MediaOffset Object](#). This object contains either a time or frame property, depending on the media player and media type.

Throws an exception if the player is not open or if the player does not support **where** ().

**Parameters**

None

**Returns**[MediaOffset Object](#)**Example:**

```
// What is the playback location in seconds?
// This code assumes that the player supports where() using time.
var where = player.where();
var seconds = where.time;
// What chapter (marker) are we in?
var marker = player.markers.get({ time: seconds });
var name = marker ? marker.name : "no marker";
```

---

**MediaReject Object**

A **MediaReject** provides information about a Rendition that was rejected by a **Rendition.select ()** call. It includes a reference to the original Rendition along with the reason why it was rejected. In a **MediaSelection Object** returned by **select ()**, **MediaSelection.rejects** is an array of **MediaReject** objects.

---

**MediaReject Object Properties****rendition**

6.0				
-----	--	--	--	--

**MediaSelection.rendition** is a reference to the Rendition that was rejected in a **select ()** call.

Type: [Rendition Object](#)    Access: R.

**Example**

Get a list of rejected renditions. The script is executed as a Rendition action.

```
selection = event.action.rendition.select(true);
for ( var i=0; i<selection.rejects.length; i++)
    console.println("Rejected Renditions: "
        + selection.rejects[i].rendition.uiName);

// now play the first available rendition.
console.println( "Preparing to play " + selection.rendition.uiName);
var settings = selection.rendition.getPlaySettings();
var args = {
```

```

        rendition: selection.rendition,
        annot: this.media.getAnnot({ nPage: 0, cAnnotTitle: "myScreen" }),
        settings: settings
    };
    app.media.openPlayer( args );

```

---

## MediaSelection Object

**Rendition.select()** returns a MediaSelection, an object which can then be used to create a [MediaSettings Object](#) for playback.

---

## MediaSelection Object Properties

### selectContext

6.0				
-----	--	--	--	--

**MediaSelection.selectContext** is a value that can be used to write a loop that calls **Rendition.select()** repeatedly to do a customized selection based on any criteria that you can test in JavaScript code.

*Type: Object*

*Access: R.*

#### Example:

```

function MyTestSelection( selection )
{
    // This function should test the selection as you wish and return
    // true to use it or false to reject it and try another one.
}
function MyGetSelection( rendition )
{
    var selection;
    for( selection = rendition.select(); selection;
        selection = rendition.select
            ( { oContext: selection.selectContext } ))
    {
        if( MyTestSelection( selection ) )
            break;
    }
    return selection;
}

```

## players

6.0				
-----	--	--	--	--

**MediaSelection.players** is an array of strings identifying the media players that may be used to play **MediaSelection.rendition**. Both the players and rendition properties are **null** if no playable rendition is found.

Type: *Array of String*      Access: *R*.

### Example

Get a list of the players that will play the selected rendition. The code below assumes execution as a Rendition action.

```
var selection = event.action.rendition.select();
for ( var o in selection.players )
    console.println( selection.players[o].id );
```

## rejects

6.0				
-----	--	--	--	--

**MediaSelection.rejects** is an array of [MediaReject Objects](#). These are the Renditions that were rejected by the **Rendition.select()** call that returned this MediaSelection. See [MediaReject Object](#) for details.

Type: *Array of MediaReject Objects*      Access: *R*.

### Example

See the [Example](#) following **MediaReject.rendition** for an example.

## rendition

6.0				
-----	--	--	--	--

**MediaSelection.rendition** is the selected rendition, or **null** if none was playable.

Type: *Rendition Object*      Access: *R*.

### Example

Get the name of the selected rendition. This script is executed from a Rendition action event.

```
var selection = event.action.rendition.select();
console.println( "Preparing to play " + selection.rendition.uiName );
```

## MediaSettings Object

A `MediaSettings` object, which appears in a `MediaPlayer.settings` property, contains settings required to create and open a `MediaPlayer`. Many of these settings have default values, but some are required depending on the type of player being opened and depending on other settings. See the notes for each `MediaSettings` property for details.

Acrobat and the various media players will attempt to use these settings, but there is no guarantee that they will all be honored. For example, very few players honor `MediaSettings.palindrome`.

## MediaSettings Object Properties

### autoplay

6.0				
-----	--	--	--	--

The `MediaSettings.autoplay` property specifies whether the media clip should begin playing automatically after the player is opened. If you set `autoplay` to `false`, use `MediaPlayer.play()` to begin playback. The default value is `true`.

*Type: Boolean*

*Access: R/W.*

#### Example

See the examples following [afterReady](#) and [players](#).

### baseURL

6.0				
-----	--	--	--	--

`MediaSettings.baseURL` is the base URL to be used to resolve any relative URLs used in the media clip, e.g. if the media opens a web page. There is no default value; if `baseURL` is not specified, the interpretation of a relative URL will vary depending the media player, but in most cases will not work.

*Type: String*

*Access: R/W.*

### bgColor

6.0				
-----	--	--	--	--

`MediaSettings.bgColor` specifies the background color for the media player window. The array may be in any of the color array formats supported by Acrobat JavaScript.

If `bgColor` is not specified, the default value depends on the window type:

- Docked: White
- Floating: The window background color specified in the operating system control panel
- Full Screen: The full screen background color specified in the user's Acrobat preferences

Type: Color Array      Access: R/W.

### Example

```
// Red background
settings.bgColor = [ "RGB", 1, 0, 0 ];
```

## bgOpacity

6.0				
-----	--	--	--	--

**MediaSettings.bgOpacity** specifies the background opacity for the media player window. The value may range from 0.0 (fully transparent) to 1.0 (fully opaque). The default value is 1.0.

Type: Number      Access: R/W.

## endAt

6.0				
-----	--	--	--	--

**MediaSettings.endAt** defines the ending time or frame for playback. This may be an absolute time or frame value, or a marker name, or a marker plus a time or frame, as described under [MediaOffset Object](#). Playback ends at the specified time or frame, or as close to that point as the media player is able to stop. If **endAt** is not specified, the default value is the end of media.

See also [startAt](#).

Type: [MediaOffset Object](#)      Access: R/W.

### Example

The following script plays an audio clip beginning 3 seconds into the media to 8 seconds into the media.

```
var player = app.media.openPlayer({
  rendition: this.media.getRendition( "myAudio" ),
  doc: this,
  settings: {
    startAt: 3,
    endAt: 8
  }
});
```

## data

6.0				
-----	--	--	--	--

**MediaSettings.data**, often referred to as the **MediaData object**, is an object that a media player can use to read its media clip data, whether from an external file or embedded in the PDF. The contents of this object are not directly usable from JavaScript.

This data object is obtained in several ways, from `app.media.getAltTextData()`, `app.media.getURLData()`, or indirectly via `Rendition.getPlaySettings()`. The data object may be bound to the rendition's document, so it may become unavailable if the document is closed.

Type: Object

Access: R.

### Example

See the examples that follow `app.media.getURLData()`

## duration

6.0				
-----	--	--	--	--

**MediaSettings.duration** is the amount of time in seconds that playback will take. If not specified, the default is to play the entire media, or the amount of time between the `startAt` and `endAt` points if either of those is specified.

Note that the duration may be longer than the entire media length or the difference between the `startAt` and `endAt` points. In that case, playback continues to the end of media or to the `endAt` point, and then playback pauses at that location until the duration elapses.

Type: Number

Access: R/W.

### Example

Play a floating window with infinite duration. The "Playback Location" (from the UI) of the rendition is a floating window. The code below is executed from a form button. The floating window will remain open after the player has reached the end of the media. We close the player before opening it again to avoid stacked floating windows.

If this script is executed from a Rendition action, the rendition could be specified through the UI, and closing the player would not be necessary.

```
var rendition = this.media.getRendition("Clip");
if ( player && player.isOpen )
    try { player.close(app.media.closeReason.done); } catch(e) {};
var player = app.media.openPlayer({
    rendition: rendition,
    settings: { duration: Infinity }
});
```



## floating

6.0				
-----	--	--	--	--

**MediaSettings.floating** is an object containing properties that define the location and style of a floating window.

This object is ignored unless **MediaSettings.windowType** has a value of **app.media.windowType.floating**.

Type: Object

Access: R.

### Properties of floating

Defaults are used for all the floating settings if they are not specified.

Property	Type	Description
<b>align</b>	Number	Specifies how the floating window is to be positioned relative to the window specified by the <b>over</b> property. The value of <b>align</b> is one of the values of <b>app.media.align</b> .
<b>over</b>	Number	Specifies what window the floating window is to be aligned relative to. The value of <b>over</b> is one of the values of <b>app.media.over</b> .
<b>canResize</b>	Number	Specifies whether the floating window may be resized by the user. The value of <b>canResize</b> is one of the values of <b>app.media.canResize</b> .
<b>hasClose</b>	Boolean	If <b>true</b> , the floating window should have a close window control button.
<b>hasTitle</b>	Boolean	If <b>true</b> , a title should be displayed in the title bar.
<b>title</b>	String	This title to be displayed if <b>hasTitle</b> is <b>true</b> .
<b>ifOffScreen</b>	Number	Specifies what action should be taken if the floating window is positioned totally or partially offscreen. The value of <b>ifOffScreen</b> is one of the values of <b>app.media.ifOffScreen</b> .
<b>rect</b>	Array of four Numbers	An array of screen coordinates specifying the location and size of the floating window. Required if <b>width</b> and <b>height</b> are not given.

Property	Type	Description
<b>width</b>	Number	The width of the floating window. Required if <b>rect</b> is not given.
<b>height</b>	Number	The height of the floating window. Required if <b>rect</b> is not given.

**Example**

```

var rendition = this.media.getRendition( "myClip" );
var floating = {
  align: app.media.align.topCenter,
  over: app.media.over.appWindow,
  canResize: app.media.canResize.no,
  hasClose: true,
  hasTitle: true,
  title: rendition.altText,
  ifOffScreen: app.media.ifOffScreen.forceOnScreen,
  width: 400,
  height: 300
};
var player = app.media.openPlayer({
  rendition: rendition,
  settings: {
    windowType: app.media.windowType.floating,
    floating: floating
  }
});
    
```

**layout**

6.0				
-----	--	--	--	--

**MediaSettings.layout** is a value chosen from the **app.media.layout** enumeration, which defines whether and how the content should be resized to fit the window. The default value varies with different media players.

Type: Number                      Access: R/W.

**monitor**

6.0				
-----	--	--	--	--

For a full screen media player, **MediaSettings.monitor** determines which display monitor will be used for playback. This may be either a **Monitor Object** or a **Monitors Object**. If it is an array, the first element (which is a Monitor object) is used.

Type: Monitor or Monitors object

Access: R/W.

**NOTE:** Only the `rect` property `MediaSettings.monitor.rect` (in the case of a Monitor object) or `MediaSettings.monitor[0].rect` (for a Monitors object) is used for playback.

See `monitorType` (below) for a discussion of the relationship between the `monitor` and `monitorType` properties.

### Example

Play a media clip in full screen from a form button.

```
var player = app.media.openPlayer({
  rendition: this.media.getRendition("Clip"),
  settings: {
    monitor: app.monitors.primary(),
    windowType: app.media.windowType.fullScreen,
  }
});
```

**NOTE:** The user trust manager settings must allow fullscreen play back.

## monitorType

6.0				
-----	--	--	--	--

`MediaSettings.monitorType` is an `app.media.monitorType` value that represents the type of monitor to be selected for playback for a floating or full screen window.

Type: Number

Access: R/W.

What is the difference between the `monitor` and `monitorType` properties? The `monitor` property specifies a specific monitor on the current running system by defining its rectangle. The `monitorType` specifies a general category of monitor such as primary, secondary, best color depth, and so forth. A PDF file that does not use JavaScript cannot specify a particular monitor, but it can specify a `monitorType`. When a `monitorType` is specified in a call to `app.media.createPlayer()` or `app.media.openPlayer()`, JavaScript code in `media.js` fetches the list of actual monitors available on the running system and then uses the `monitorType` to select one of those monitors for playback. This monitor rectangle is then used when `MediaPlayer.open()` is called to select the actual monitor.

### Example

Play a media clip in full screen on a monitor with the best color depth from a form button.

```
var player = app.media.openPlayer({
  rendition: this.media.getRendition("Clip"),
  settings: {
    monitorType: app.media.monitorType.bestColor,
  }
});
```

```

        windowType: app.media.windowType.fullScreen,
    }
});

```

## page

6.0				
-----	--	--	--	--

For a docked media player, **MediaSettings.page** is the document page number in which the player should be docked. For other types of media players, this property is ignored.

Type: Number                      Access: R/W.

See also **MediaPlayer.page**.

## palindrome

6.0				
-----	--	--	--	--

If **MediaSettings.palindrome** is **true**, the media plays once normally and then plays in reverse back to the beginning. If **repeat** is specified, then this forward-and-reverse playback will repeat that many times. Each complete forward and reverse playback counts as one repeat.

The default value is **false**.

Type: Boolean                      Access: R/W.

**NOTE:** Most media players do not support palindrome and ignore this setting.

### Example

Use QuickTime, which supports palindrome, to view the media clip.

```

var playerList = app.media.getPlayers().select({ id: /quicktime/i });
var settings = { players: playerList, palindrome: true };
var player = app.media.openPlayer({ settings: settings });

```

The above code should be run within a Rendition action event with an associated rendition.

## players

6.0				
-----	--	--	--	--

**MediaSettings.players** is an array of objects that represent the media players that may be used to play this rendition. JavaScript code does not usually access this array directly, but passes it through from **Rendition.select()** to the **settings** object for **app.media.createPlayer()**.

*Type: Players or Array of String**Access: R/W.***Example**

List the available players that can play this rendition. This script is run as a Rendition action with associated rendition.

```
var player = app.media.openPlayer({ settings: {autoPlay: false} });
console.println("players: " + player.settings.players.toSource() );

// Sample output to the console:
players: [{id:"vnd.adobe.swname:ADBE_MCI", rank:0},
{id:"vnd.adobe.swname:AAPL_QuickTime", rank:0},
{id:"vnd.adobe.swname:RNWK_RealPlayer", rank:0},
{id:"vnd.adobe.swname:MSFT_WindowsMediaPlayer", rank:0}]
```

**rate**

6.0				
-----	--	--	--	--

**MediaSettings.rate** defines the playback rate, where 1 is normal playback, .5 is half-speed, 2 is doublespeed, -1 is normal speed in reverse, and so on.

Many players and media types are limited in the values they support for rate and will choose the closest playback rate that they support.

The default value is 1.

*Type: Number**Access: R/W.***Example**

Play a media clip at doublespeed. This script is executed as a Rendition action.

```
var player = app.media.createPlayer();
player.settings.rate = 2;
player.open();
```

**repeat**

6.0				
-----	--	--	--	--

**MediaSettings.repeat** is the number of times the media playback should automatically repeat. The default of value of 1 causes the media to be played once.

Many players support only integer values for repeat, but some allow non-integer values such as 1.5. A value of **Infinity** plays the media clip continuously.

The default value is 1.

*Type: Number**Access: R/W.*

**Example**

Play a media clip from a Rendition action continuously.

```
var player = app.media.openPlayer({settings: { repeat: Infinity } });
```

**showUI**

6.0				
-----	--	--	--	--

**MediaSettings.showUI**, a boolean, specifies whether the controls of the media player should be visible or not.

The default value is **false**.

Type: *Boolean*

Access: *R/W*.

**Example**

Show the controls of the media player. This script is executed as a Rendition action.

```
var player = app.media.createPlayer();
player.settings.showUI = true;
player.open();
```

or

```
app.media.openPlayer( {settings: {showUI: true} } );
```

**startAt**

6.0				
-----	--	--	--	--

**MediaSettings.startAt** defines the starting time or frame for playback. This may be an absolute time or frame value, or a marker name, or a marker plus a time or frame, as described under [MediaOffset](#). Playback starts at the specified time or frame, or as close to that point as the media player is able to stop. If **startAt** is not specified, the default value is the beginning of media.

Type: *MediaOffset Object* Access: *R/W*.

See also [endAt](#).

**Example**

See the example that follows [endAt](#).

**visible**

6.0				
-----	--	--	--	--

**MediaSettings.visible**, a boolean, specifies whether the player should be visible.

The default value is **true**.

Type: *Boolean*

Access: *R/W*.

### Example

Set a docked media clip to play audio only. Script is executed as a Rendition action.

```
var args = {
  settings: {
    visible: false,
    windowType: app.media.windowType.docked
  }
};
app.media.openPlayer( args );
```

See also **MediaPlayer.visible**.

## volume

6.0				
-----	--	--	--	--

**MediaSettings.volume** specifies the playback volume. A value of 0 is muted, a value of 100 is normal (full) volume; values in between are intermediate volumes. Future media players may allow values greater than 100 to indicate louder than normal volume, but none currently do.

The default value is 100.

Type: *Number*

Access: *R/W*.

## windowType

6.0				
-----	--	--	--	--

**MediaSettings.windowType** is a value chosen from the **app.media.windowType** enumeration, which defines what type of window the MediaPlayer should be created in.

If you use the low-level function **doc.media.newPlayer()**, the default value for **windowType** is **app.media.windowType.docked**.

If you use the higher-level **createPlayer()** or **openPlayer()** functions of the **App.media Object**, the default value is determined as follows:

- If an **annot** is provided (see the description of the **PlayerArgs Object**), the default is **app.media.windowType.docked**.
- If a settings.floating object is provided (see the description of the **PlayerArgs Object**), the default is **app.media.windowType.floating**.
- Otherwise, the default is undefined.

*Type: Number**Access: R/W.***Example**

The script below creates media players with different window types. Script is executed as a Rendition action, so the selection of the specification of the rendition is not needed.

```
// Docked player that will be played in the associated ScreenAnnot
app.media.openPlayer({
  settings: { windowType: app.media.windowType.docked }
});
// Play in full screen mode, see also monitor and monitorType
app.media.openPlayer({
  settings: { windowType: app.media.windowType.fullScreen }
});
// Show media clip in a floating window, also, see the floating property
var args = {
  settings: {
    windowType: app.media.windowType.floating,
    floating: {
      title: "A. C. Robot",
      width: 352,
      height: 240,
    }
  }
};
app.media.openPlayer( args );
```

---

**Monitor Object**

A Monitor object represents an individual display monitor. A Monitor object can be obtained from `app.media.monitors`, which returns an array of all monitors connected to the system. `app.media.monitors` is a [Monitors Object](#) so the methods of the Monitors object can be used to select or filter out monitors from a multi-monitor system based on different criteria. See the Monitors object for details.

The Monitor object and the Monitors object are used in `MediaSettings.monitor`.

---

**Monitor Object Properties****colorDepth**

6.0				
-----	--	--	--	--

`Monitor.colorDepth` is the color depth of the monitor, i.e. the number of bits per pixel.



*Type: Number**Access: R.***Example**

Get the primary monitor, and check its color depth. The **Monitors.primary()** method is use to select the primary monitor.

```
var monitors = app.monitors.primary();
console.println( "Color depth of primary monitor is "
    + monitors[0].colorDepth );
```

**isPrimary**

6.0				
-----	--	--	--	--

**Monitor.primary**, a boolean, is **true** for the primary monitor, **false** for all other monitors.

*Type: Boolean**Access: R.***Example**

Get the widest monitor, and see if its the primary monitor.

```
var monitors = app.monitors.widest();
var isIsNot = (monitors[0].isPrimary) ? "is" : "is not";
console.println("The widest monitor "+isIsNot+" the primary monitor.");
```

**rect**

6.0				
-----	--	--	--	--

**Monitor.rect** is a rectangle representing a boundaries of the monitor in virtual desktop coordinates.

The origin of the virtual desktop origin is the top left corner of the primary monitor, so the primary monitor's bounds are always in the form [ 0, 0, right, bottom ]. Secondary monitors may have positive or negative values in their bounds arrays, depending on where they are positioned relative to the primary monitor.

*Type: Rectangle**Access: R.***workRect**

6.0				
-----	--	--	--	--

Monitor.workRect is a rectangle representing a monitor s workspace boundaries in virtual desktop coordinates. See **Monitor.rect** for information about these coordinates.

The workspace is the area of a monitor that is normally used for applications, omitting any docked toolbars, taskbars, or the like. For example, running Windows on a single 800x600 display, **Monitor.rect** is [ 0, 0, 800, 600 ]. With a standard Windows taskbar 30 pixels high and always visible at the bottom of the screen, **Monitor.workRect** is [ 0, 0, 800, 570 ].

Type: *Rectangle*

Access: *R.*

---

## Monitors Object

A Monitors object is a read-only array of [Monitor Object](#), each one representing a display monitor.

The **app.monitors** property returns a Monitors object that includes every monitor connected to the user's system. JavaScript code can loop through this array to get information about the available monitors and select one for a full screen or popup media player.

Monitors also has a number of filter methods that select one or more monitors based on various criteria.

All of the monitor selection options provided in the PDF file format are implemented as calls to these filter methods, which are written in JavaScript code in `media.js`.

None of the Monitors filter methods modify the original Monitors object. They each return a new Monitors object which normally contains one or more Monitor objects. If a single monitor matches the filtering criterion better than any other, the result Monitors object contains that one monitor. If more than one monitor satisfies the filtering criterion equally (e.g. for the [bestColor\(\)](#) method, if more than one monitor has the same, greatest color depth), then the result contains all of those monitors.

Several of the filter methods have an optional minimum or require parameter. If this parameter is specified and no monitor meets that minimum requirement, then the result Monitors object is empty. Otherwise, the result will always contain at least one monitor, if the original Monitors object was not empty.

Wherever a filter method refers to height, width, or area, these are dimensions in pixels, not physical size.

A Monitors object is not actually an Array type, but it can be used as if it were a read-only array with numbered elements and a length property.

---

## Monitors Object Properties

A Monitors object works like an Array, where each array element is a [Monitor Object](#) that represents a single monitor. The Monitors object returned by **app.monitors** is unsorted the monitors are not listed in any particular order. Elements of the Monitors object can be accessed using the usual array notation.

**Example.**

```
var monitors = app.monitors;
for ( var i = 0; i < monitors.length; i++)
  console.println("monitors["+i+"] .colorDepth = "+monitors[i].colorDepth);
```

**Monitors.length** contains the number of elements in the Monitors object. For the Monitors object returned by **app.monitors**, this is the number of monitors in the user's system. For a Monitors object returned by one of the filter methods, this number may be smaller.

---

## Monitors Object Methods

### bestColor

6.0				
-----	--	--	--	--

The **Monitors.bestColor()** method returns a copy of the [Monitors Object](#), filtered to include the monitor(s) with the greatest color depth.

If **nMinColor** is specified, returns an empty Monitors array if the best color depth is less than **nMinColor**.

**Parameters**

<b>nMinColor</b>	(optional) The minimal color depth required of the monitor.
------------------	---

**Returns**

A [Monitors Object](#)

**Example**

```
var monitors = app.monitors.bestColor(32);
if (monitors.length == 0 )
  console.println("Cannot find the required monitor.");
else
  console.println("Found at least one monitor.");
```

### bestFit

6.0				
-----	--	--	--	--

The **Monitors.bestFit()** method returns a copy of the [Monitors Object](#), filtered to include only the smallest monitor(s) with at least the specified **nWidth** and **nHeight** in pixels.

**Parameters**

<b>nWidth</b>	Minimum width of the best fit monitor
<b>nHeight</b>	Minimum height of the best fit monitor.
<b>bRequire</b>	(optional) If no monitors have at least the specified width and height, then returns an empty Monitors array if <b>bRequire</b> is <b>true</b> , or a Monitors array containing the largest monitor(s) if <b>bRequire</b> is <b>false</b> or omitted.

**Returns**

A [Monitors Object](#)

**desktop**

**Monitors.desktop()** creates a new [Monitors Object](#) containing one Monitor which represents the entire virtual desktop. In this Monitor object, the **rect** property is the union of every **rect** in the original Monitors object, the **workRect** property is the union of every **workRect** in the original Monitors object, and **colorDepth** is the minimum **colorDepth** value found in the original Monitors object.

**Parameters**

None

**Returns**

A [Monitors Object](#)

**NOTE:** The **desktop()** method is normally called directly on a Monitors object returned by **app.monitors**. If that Monitors object is first filtered by any of its other methods, then the **desktop()** method does the same calculations listed above with that subset of the monitors.

**document**

6.0				
-----	--	--	--	--

The **Monitors.document()** method returns a copy of the [Monitors Object](#), filtered to include the monitor(s) that display the greatest amount of the document, as specified by the document object parameter **doc**.

If the document does not appear on any of the monitors in the original Monitors object, then returns an empty Monitors array if **bRequire** is **true** or a Monitors array containing at least one arbitrarily chosen monitor from the original array if **bRequire** is **false** or omitted.

**Parameters**

<b>doc</b>	The document object of the document
<b>bRequire</b>	(optional) A boolean. See the description above.

**Returns**

A [Monitors Object](#)

**filter**

6.0				
-----	--	--	--	--

**Monitors.filter()** returns a copy of the [Monitors Object](#), filtered by calling a ranker function for each monitor in the list. The ranker function takes a Monitor parameter and returns a numeric rank. The return value from **filter()** is a Monitors array containing the monitors which had the highest rank (either a single monitor, or more than one if there was a tie).

**Parameters**

<b>fnRanker</b>	A (ranker) function that takes a Monitor parameter and and returns a numeric rank
<b>nMinRank</b>	(optional) If <b>nMinRank</b> is undefined, <b>filter()</b> always includes at least one monitor from the original list (unless the original list was empty). If <b>nMinRank</b> is specified, then <b>filter()</b> returns an empty Monitors array if no monitors had at least that rank according to the ranker function.

**Returns**

A [Monitors Object](#)

**NOTE:** Most of the other Monitors filtering functions are implemented as filter() calls.

**Example:**

This is the implementation of **Monitors.bestColor( minColor )** from `media.js`: Returns a Monitors object containing the monitor(s) that have the greatest color depth. If `minColor` is specified, returns an empty Monitors array if the best color depth is less than `minColor`.

```
bestColor: function( minColor )
{
    return this.filter(
        function( m ) { return m.colorDepth; }, minColor );
}
```

## largest

6.0				
-----	--	--	--	--

**Monitors.largest ()** returns a copy of the [Monitors Object](#), filtered to include the monitor(s) with the greatest area in pixels.

### Parameters

<b>nMinArea</b>	(optional) If the optional parameter <b>nMinArea</b> , a number, is specified, <b>largest ()</b> returns an empty <a href="#">Monitors</a> array if that greatest area is less than that value.
-----------------	---

### Returns

A [Monitors Object](#)

## leastOverlap

6.0				
-----	--	--	--	--

The **Monitors.leastOverlap ()** method returns a copy of the [Monitors Object](#), filtered to include the monitor(s) that contain the smallest amount of the rectangle, as specified by the **rect** parameter.

### Parameters

<b>rect</b>	A rectangle, an array of four numbers in screen coordinates.
<b>maxOverlapArea</b>	(optional) If <b>maxOverlapArea</b> is specified, the result <a href="#">Monitors</a> array contains only those monitors which contain at least that much area of the rectangle, or an empty <a href="#">Monitors</a> array if no monitors contain that much area of the rectangle.

### Returns

A [Monitors Object](#)

## mostOverlap

6.0				
-----	--	--	--	--

The **Monitors**.**mostOverlap** () method returns a copy of the [Monitors Object](#), filtered to include the monitor(s) that contain the largest amount of the rectangle, as specified by the **rect** parameter.

### Parameters

<b>rect</b>	A rectangle, an array of four numbers in screen coordinates.
<b>minOverlapArea</b>	(optional) If there is no monitor with at least that much overlapping area, then returns an empty <a href="#">Monitors</a> array if <b>minOverlapArea</b> is specified, or a <a href="#">Monitors</a> array containing at least one arbitrarily chosen monitor from the original array if <b>minOverlapArea</b> is omitted.

### Returns

A [Monitors Object](#)

## nonDocument

6.0				
-----	--	--	--	--

The **Monitors**.**nonDocument** () method returns a copy of the [Monitors Object](#), filtered to include the monitor(s) that display none of, or the least amount of the document.

### Parameters

<b>doc</b>	The document object of the target document
<b>bRequire</b>	(optional) <b>bRequire</b> is a boolean which determines the return value when there is no monitor that is completely clear of the document. If <b>true</b> , <b>nonDocument</b> () returns an empty, or if false or omitted, <b>nonDocument</b> () returns a <a href="#">Monitors</a> array containing at least one arbitrarily chosen monitor from the original <a href="#">Monitors</a> array.

### Returns

A [Monitors Object](#)

## primary

6.0				
-----	--	--	--	--

**Monitors.primary()** returns a copy of the [Monitors Object](#), filtered by removing all secondary monitors, leaving only the primary monitor if it was present in the original list.

If the primary monitor was not present in the original list, returns a Monitors array containing at least one arbitrarily chosen monitor from the original list.

### Parameters

None

### Returns

A [Monitors Object](#)

### Example

Get the primary monitor, and check its color depth.

```
var monitors = app.monitors.primary();
// recall that each element in a monitors object is a monitor object,
// this code uses monitor.colorDepth
console.println( "Color depth of primary monitor is "
    + monitors[0].colorDepth );
```

## secondary

6.0				
-----	--	--	--	--

**Monitors.secondary()** returns a copy of the [Monitors Object](#), filtered by removing the primary monitor, returning only secondary monitors.

If the original Monitors object contained only the primary monitor and no secondary monitors, returns the original list.

### Parameters

None

### Returns

A [Monitors Object](#)

## select

6.0				
-----	--	--	--	--

**Monitors.select()** returns a copy of the [Monitors Object](#), filtered according to **nMonitor**, a monitor selection value as used in PDF and enumerated in [app.media.monitorType](#).



The `doc` is required when `nMonitor` is `app.media.monitorType.document` or `app.media.monitorType.nonDocument`, and ignored for all other `nMonitor` values.

These selection values correspond directly to the various Monitors filter methods. `select()` calls the corresponding filter method, and then, in most cases, also filters with `primary()` as a tie-breaker in case more than one monitor matches the main filter. See the code in `media.js` for details.

**Parameters**

<code>nMonitor</code>	The monitor type, a number from <code>app.media.monitorType</code> .
<code>doc</code>	A document object. The parameter is required if <code>nMonitor</code> is either <code>app.media.monitorType.document</code> or <code>app.media.monitorType.nonDocument</code> , ignored otherwise.

**Returns**

A [Monitors Object](#)

**Example:**

```
// These two calls are equivalent:
settings.monitor =
app.monitors().select( app.media.monitorType.document, doc );
settings.monitor = app.monitors().document(doc).primary();
```

**tallest**

6.0				
-----	--	--	--	--

`Monitors.tallest()` returns a copy of the [Monitors Object](#), filtered to include only the monitor(s) with the greatest height in pixels.

**Parameters**

<code>nMinHeight</code>	(optional) If <code>nMinHeight</code> is specified and no monitor has at least that height, the return value is an empty Monitors array.
-------------------------	--

**Returns**

A [Monitors Object](#)

## widest

6.0				
-----	--	--	--	--

**Monitors.widest** () returns a copy of the [Monitors Object](#), filtered to include only the monitor(s) with the greatest width in pixels.

### Parameters

---

<b>nMinWidth</b>	(optional) If nMinWidth is specified and no monitor has at least that width, the return value is an empty Monitors array.
------------------	---

---

### Returns

A [Monitors Object](#)

---

## OCG Object

An **OCG** object represents an *Optional Content Group* in a PDF file. Content in the file can “belong” to one or more Optional Content Groups. Content belonging to one or more OCGs is referred to as “Optional Content” and its visibility is determined by the on/off states of the OCGs to which it belongs. In the simplest case, optional content will belong to a single OCG with the content being visible when the OCG is on and hidden when the OCG is off. More advanced visibility behavior can be achieved by using multiple OCGs and different visibility mappings.

Use `doc.getOCGs` to get an array of **OCG** objects for a PDF document.

The methods `doc.addWatermarkFromFile` and `doc.addWatermarkFromText` add watermarks in an OCG.

See the [PDF Reference](#), Section 4.10, for additional details on Optional Content Groups.

---

## OCG Properties

### constants

7.0				
-----	--	--	--	--

Each instance of an OCG object inherits this property, which is a wrapper object for holding various constant values.

### intents Object

An OCG's intent array can contain arbitrary strings, but those contained in this object are the only ones recognized by Acrobat.

Property	Description
<b>design</b>	Designates a "Design" intent in an OCG object.
<b>view</b>	Designates a "View" intent in an OCG object.

### states Object

The **states** object is used to set the initial state of the OCG, see [initState](#).

Property	Description
<b>on</b>	Designates an OCG state of "On".
<b>off</b>	Designates an OCG state of "Off".

### initState

7.0	Ⓓ		ⓧ	ⓧ
-----	---	--	---	---

This property is used to determine whether this OCG is on or off by default. See the [states Object](#) for possible values.

Type: Boolean

Access: R/W.

#### Example

Set an initial state of an OCG to off.

```
var ocgs = this.getOCGs();
ocgs[0].initState.constants.states.off;
```

### locked

7.0	Ⓓ		ⓧ	ⓧ
-----	---	--	---	---

This property is used to determine whether this OCG is locked. If an OCG is locked then its on/off state cannot be toggled through the UI.

Type: Boolean

Access: R/W.

**name**

6.0	ⓓ		✕	✕
-----	---	--	---	---

The text string seen in the UI for this OCG. It can be used to identify OCGs, although it is not necessarily unique.

**NOTE:** In Acrobat 6.0, the **name** is read-only; for Acrobat 7.0, it is read/write.

*Type: String*

*Access: R/W.*

**Example**

```
/* Toggle the Watermark OCG */
function ToggleWatermark(doc)
{
    var ocgArray = doc.getOCGs();
    for (var i=0; i < ocgArray.length; i++) {
        if (ocgArray[i].name == "Watermark") {
            ocgArray[i].state = !ocgArray[i].state;
        }
    }
}
```

**state**

6.0				
-----	--	--	--	--

Represents the current on/off state of this OCG.

*Type: Boolean*

*Access: R/W.*

**Example**

Turn on all the OCGs in the given document.

```
function TurnOnOCGsForDoc(doc)
{
    var ocgArray = doc.getOCGs();
    for (var i=0; i < ocgArray.length; i++)
        ocgArray[i].state = true;
}
```

## OCG Methods

### getIntent

7.0				
-----	--	--	--	--

Returns this OCG's intent array.

An OCG will affect the visibility of content only if it has `onstants.intents.view` as an intent.

See also [setIntent](#) and the [intents Object](#).

#### Parameters

None

#### Returns

An array of strings. See `constants.intents` for possible values.

### setAction

6.0				
-----	--	--	--	--

Registers a JavaScript expression to be evaluated after every state change for this OCG.

**NOTE:** This method will overwrite any action already defined for this OCG.

#### Parameters

<b>cExpr</b>	The expression to be evaluated after the OCG state changes.
--------------	---

#### Returns

Nothing

#### Example

```
/* Beep when the given ocg is changed */
function BeepOnChange(ocg)
{
    ocg.setAction("app.beep()");
}
```

## setIntent

7.0	Ⓓ		ⓧ	ⓧ
-----	---	--	---	---

Sets this OCG's intent array. An OCG should only affect the visibility of content if this array contains constants.intents.view. See the [intents Object](#) for possible values.

See also [getIntent](#) and the [intents Object](#).

### Parameters

---

**aIntentArray** An array of strings to be used as this OCG's intent array.

---

### Returns

Nothing

### Example

Set the intent of all OCGs in the document to both View and Design

```
var ocgs = this.getOCGs();
for (i=0; i < ocgs.length; i++) {
    ocgs[i].setIntent( [ocgs[i].constants.intents.view,
    ocgs[i].constants.intents.design]);
}
```

## PlayerInfo Object

A PlayerInfo object represents a media player that is available for media playback. The `app.media.getPlayers()` function returns a [PlayerInfoList Object](#), which is a collection of PlayerInfo objects.

## PlayerInfo Object Properties

### id

6.0				
-----	--	--	--	--

**PlayerInfo.id** represents a media player plug-in and associated media player. This string is not localized and is not intended for display to the user. This string may be used in the `MediaPlayer.settings.players` array when creating a MediaPlayer, and it is also found in the `MediaPlayer.id` property after opening a player.

Type: String

Access: R.

**Example**

List player info for all media players that play "video/mpeg".

```
var playerInfoList = app.media.getPlayers("video/mpeg");

for ( var i=0; i < playerInfoList.length; i++) {
  console.println("id: " + playerInfoList[i].id)
  console.println("name: " + playerInfoList[i].name)
  console.println("version: " + playerInfoList[i].version)
}
```

**mimeTypes**

6.0				
-----	--	--	--	--

The **PlayerInfo.mimeTypes** property returns an array of strings listing the MIME types that this media player supports.

*Type: Array of String*      *Access: R.*

**Example:**

```
var qtinfo = app.media.getPlayers().select({id: /quicktime/i })[0];
console.println( qtinfo.mimeTypes );
```

**name**

6.0				
-----	--	--	--	--

**PlayerInfo.name** is the name of the media player. This string is localized according to the current language as found in `app.language`. It is suitable for display in list boxes and the like, but not for direct comparisons in JavaScript code.

*Type: String*      *Access: R.*

**version**

6.0				
-----	--	--	--	--

**PlayerInfo.version** is a string containing the version number of the media player. For most players, it is the version number of the underlying media player that is installed on the user's system. This string is in dotted decimal format, e.g. 7.4.030.1170 .

*Type: String*      *Access: R.*

## PlayerInfo Object Methods

### canPlay

6.0				
-----	--	--	--	--

**PlayerInfo.canPlay()** checks to see if the media player can be used for playback, taking the user's security settings into account.

If the parameter **bRejectPlayerPrompt** is **true**, then the method returns **false** if using this player would result in a security prompt. Otherwise the method returns **true** if playback is allowed either with or without a security prompt. (This method itself never triggers a security prompt, but a later attempt to create a media player may.)

#### Parameters

<b>oDoc</b>	A document object
<b>bRejectPlayerPrompt</b>	A boolean, which if <b>true</b> , the method returns <b>false</b> if using this player would result in a security prompt, and if <b>false</b> , returns <b>true</b> if playback is allowed either with or without a security prompt. The default is <b>false</b> .

#### Returns

Boolean

### honors

7.0				
-----	--	--	--	--

This method asks a player plugin if it can honor all of the settings, methods, and events listed in the **args** parameter. The answer is not guaranteed to be correct, but is a best guess of the player plugin without actually trying to open a media player. For example, if **args.URL** is provided, the scheme (such as "http://") is checked, but **PlayerInfo.honors()** does not try to actually open the URL.

**NOTE:** Acrobat 6.0 compatibility: **PlayerInfo.honors()** is supported only on Acrobat 7 and above. The Acrobat SDK provides JavaScript source code that can be copied into a PDF to provide compatibility with both Acrobat 6 and Acrobat 7. This code uses hard coded tests for Acrobat 6 and calls **PlayerInfo.honors()** on newer versions of Acrobat. See the [playerHonors Function](#) for details.

**PlayerInfo.honors** and the HonorsArgs (see [HonorsArgs Object](#)) are similar to the **MH** ("must honor") entries in the PDF format, some of which can be set in the **Playback Requirements** panel of the **Rendition Settings** for a multimedia rendition. The **honors**



method provides a way to choose a player that meets playback requirements dynamically in JavaScript code instead of statically in the PDF file.

## Parameters

---

<b>args</b>	The <a href="#">HonorsArgs Object</a> to be tested. The <a href="#">HonorsArgs Object</a> is very similar to the parameter, <a href="#">PlayerArgs Object</a> , used by the <code>app.media.openPlayer ()</code> method. In fact, any <code>PlayerArgs</code> object can be used as an <code>HonorsArgs</code> . <code>HonorsArgs</code> also allows a few other options that are used only with <code>honors ()</code> .
-------------	---

---

## Returns

Boolean, **true** if the player plugin can honor everything in the **args** object.

## Example

Play a media clip using a player that supports specific features.

```
function playWithRequirements( args )
{
    var plugins = app.media.getPlayers( args.mimeType )
    if( plugins )
    {
        for (var plugin in plugins)
        {
            if( plugin.honors( args ) )
            {
                args.players = [plugin];
                return app.media.openPlayer( args );
            }
        }
    }
}
```

Play using a media player that has these capabilities for an AVI file on an http URL: It can turn off autoplay, supports the pause method, the seek method and startAt setting using a marker+time offset, and supports the Ready and Close events.

```
playWithRequirements({
    mimeType: 'video/avi',
    URL: 'http://www.foo.com/bar.avi',
    settings:
    {
        autoPlay: false,
        startAt: { marker: 'test', time: 1 } },
    },
    methods:
    {
        pause: [],
        seek[ { marker: 'test', time: 1 } ],
    },
}
```

```

    events:
    {
        afterReady: doAfterReady( e ),
        onClose: doOnClose( e ),
    },
});

```

## HonorsArgs Object

The HonorsArgs object lists settings, methods, and events that are used in a call to **PlayerInfo.honors** (or `playerHonors`--in this discussion we will use "PlayerInfo.honors" to refer to both).

Any **PlayerArgs Object** (as used in a call to `app.media.openPlayer`) may be used as an HonorsArgs object, or an HonorsArgs object can be built from scratch to be used in a **PlayerInfo.honors** call.

If the same object is used in `app.media.openPlayer` and **PlayerInfo.honors**, be aware that the two functions interpret unknown **args** differently: **app.media.openPlayer** ignores settings or events that it does not know about, but **PlayerInfo.honors** returns **false** if there are any settings, methods, or events it does not recognize.

For example, `{ settings: { upsideDown: true } }` would be allowed in an `app.media.openPlayer` call. There is no such setting as "upsideDown", so the setting is ignored. But in a **PlayerInfo.honors** call, this unknown setting would cause **PlayerInfo.honors** to return **false**.

Below is a complete list of the properties allowed in the HonorsArgs object. This illustration is loosely in the form of a JavaScript object literal, but it shows the type or description of each property instead of an actual property value:

```

args =
{
    mimeType: string,
    URL: string,
    settings:
    {
        autoPlay: boolean,
        baseURL: string,
        bgColor: Acrobat color array,
        duration: number,
        endAt: MediaOffset,
        layout: number,
        palindrome: boolean,
        rate: number,
        repeat: number,
        showUI: boolean,
        startAt: MediaOffset,
        visible: boolean,
        volume: number,
    },
    methods:

```

```

    {
      pause: [],
      play: [],
      seek: [ MediaOffset ],
      stop: [],
      where: [],
    },
    events:
    {
      Done: anything, onDone: anything, afterDone: anything,
      Error: anything, onError: anything, afterError: anything,
      Escape: anything, onEscape: anything, afterEscape: anything,
      Pause: anything, onPause: anything, afterPause: anything,
      Play: anything, onPlay: anything, afterPlay: anything,
      Ready: anything, onReady: anything, afterReady: anything,
      Script: anything, onScript: anything, afterScript: anything,
      Seek: anything, onSeek: anything, afterSeek: anything,
      Status: anything, onStatus: anything, afterStatus: anything,
      Stop: anything, onStop: anything, afterStop: anything,
    },
  }

```

Any PlayerArgs object (as used in a call to **app.media.openPlayer**) may be used as an HonorsArgs object, or you can build an HonorsArgs object from scratch.

Additional comments on the above listing.

- The **mimeType**, **URL**, and **settings** properties are identical to the corresponding properties in PlayerArgs. The **mimeType** property is required; **playerInfo.honors** does not try to determine the MIME type from the URL's file extension. **URL** can be a real URL or a fictitious one as long as it's in the correct URL format. See [MediaSettings Object](#) for a description of these properties.
- The **methods** property lists the MediaPlayer methods that the player must support for the given MIME type. The value of each **methods** property is an array containing the arguments that would be passed into a call to that method. In Acrobat 7, the only player method that has any arguments is **seek**, which takes a single MediaOffset argument. See [MediaPlayer Object](#) for a description of these properties.

If you use the same object as a PlayerArgs and an HonorsArgs, it can have a **methods** property, even though a PlayerArgs normally doesn't have that property. Anywhere a PlayerArgs is used, the unknown property is ignored.

- The **events** property lists the events that the player must support. As shown above, each event can be named with the on or after prefix, or no prefix. All three mean the same thing; if a player supports a particular "on" event, then it always supports the corresponding "after" event (because the "after" events are generated in the same way for all players). See [EventListener Object](#) for a description of these properties.

The notation **anything** means literally that: in the HonorsArgs, these values are just placeholders. So, the events object from a PlayerArgs works in an HonorsArgs:

```

events:
{

```

```

        afterReady: doAfterReady( e ),
        onClose: doOnClose( e ),
    },

```

Or, if you are coding an HonorsArgs from scratch, you can simplify the notation if you wish:

```

events: { Ready: true, Close: true },

```

### playerHonors Function

This function is provided as JavaScript source code which can be copied into a PDF file as a document script. It performs the same tests as the PlayerInfo.honors method in Acrobat 7, but it works on Acrobat 6 as well.

When running on Acrobat 6, **playerHonors** uses hard coded tests that match the capabilities of the media players shipped with Acrobat 6.

When running on Acrobat 7 and greater, **playerHonors** calls **PlayerInfo.honors**.

### Parameters

<b>doc</b>	A <a href="#">Doc Object</a> .
<b>info</b>	A <a href="#">PlayerInfo Object</a> .
<b>args</b>	The <a href="#">HonorsArgs Object</a> to be tested.

### Returns

Boolean, **true** if the player plugin can honor everything in the **args** object.

### Example

This is the same [Example](#) as shown for **PlayerInfo.honors**, but using the **playerHonors()** JavaScript function. This works on both Acrobat 6 and 7, provided a copy of the **playerHonors** source code is placed into the target PDF.

```

function playWithRequirements( args ) {
    var plugins = app.media.getPlayers( 'video/avi' )
    if( plugins ) {
        for (var plugin in plugins) {
            if( playerHonors( doc, plugin, args ) ) {
                args.players = [plugin];
                return app.media.openPlayer( args );
            }
        }
    }
}

```

## canUseData

6.0				
-----	--	--	--	--

Tells whether the player can use the specified data, as passed by its parameter **oData**, for playback. Returns **true** if the data can be used for playback, and **false**, otherwise.

### Parameters

<b>oData</b>	<b>oData</b> is a <b>MediaData</b> object (see <b>MediaSettings.data</b> for a description of this object). This data object is obtained in several ways, from <b>app.media.getAltTextData()</b> , <b>app.media.getURLData()</b> , or indirectly via <b>Rendition.getPlaySettings()</b> .
--------------	---

### Returns

Boolean

## PlayerInfoList Object

The **app.media.getPlayers()** method returns a **PlayerInfoList**, which is an array of **PlayerInfo Objects**.

The **PlayerInfoList** has one method, **select()**, which can be used to filter the list using any of the properties in a **PlayerInfo**.

When a media player is created using **app.media.createPlayer()**, the **settings.players** property (see the **PlayerArgs Object**) may contain a **PlayerInfoList**, to restrict the player creation to choose from only those players specified in the list.

## PlayerInfoList Object Properties

The **PlayerInfoList** works like an array, the elements, which are **PlayerInfo Objects**, can be accessed using the usual array notation. The number of elements in the **PlayerInfoList** array can be obtain from the **length** property.

## PlayerInfoList Object Methods

### select

6.0				
-----	--	--	--	--

**PlayerInfoList.select()** returns a copy of the PlayerInfoList, filtered to include only the players that meet the selection criteria. This will be an empty array if no players match.

The object parameter may contain any of the following properties. Any properties that are specified are required to match; properties that are omitted match any player.

```
id: string or regular expression
name: string or regular expression
version: string or regular expression
```

The id, name, and version properties may be either strings for an exact match, or regular expressions.

#### Parameters

<b>object</b>	(optional) An object which contains any of the properties <b>id</b> , <b>name</b> , or <b>version</b> . The values of these properties may be a string or a regular expression.
---------------	---

#### Returns

[PlayerInfoList Object](#)

#### Example 1

Use QuickTime to view the media clip.

```
var playerList = app.media.getPlayers().select({ id: /quicktime/i });
// QuickTime supports palindrome, so let's try it.
var settings = { players: playerList, palindrome: true };
var player = app.media.openPlayer({ settings: settings });
```

#### Example 2

Choose the Flash player by using a pattern match on its player ID.

```
var player = app.media.createPlayer();
player.settings.players = app.media.getPlayers().select({ id:/flash/i});
player.open();
```

---

## PlugIn Object

5.0			
-----	--	--	--

This object gives access to information about the plug-in it represents. A `plugIn` object is obtained using `app.pluginIn`.

---

## PlugIn Properties

### certified

If `true`, the plug-in is certified by Adobe. Certified plug-ins have undergone extensive testing to ensure that breaches in application and document security do not occur. The user can configure the viewer to only load certified plug-ins.

*Type: Boolean*

*Access: R.*

#### Example

Get the number of uncertified plugins.

```
var j=0; aPlugins = app.pluginIn;
for (var i=0; i < aPlugins.length; i++)
    if (!aPlugins[i].certified) j++;
console.println("Report: There are "+j+" uncertified plugins loaded.");
```

### loaded

If `true`, the plug-in was loaded.

*Type: Boolean*

*Access: R.*

### name

The name of the plug-in.

*Type: String*

*Access: R.*

#### Example

```
// get array of PlugIn Objects
var aPlugins = app.pluginIn;
// get number of plugins
var nPlugins = aPlugins.length;
// enumerate names of all plugins
for (var i = 0; i < nPlugins; i++)
    console.println("Plugin \#" + i + " is " + aPlugins[i].name);
```

## path

The device-independent path to the plug-in. See *File Specification Strings*, Section 3.10.1, of the [PDF Reference](#) for the exact syntax of the path.

Type: *String*

Access: *R*.

## version

The version number of the plug-in. The integral part of the version number indicates the major version, the decimal part indicates the minor and update versions. For example, 5.11 would indicate that the plug-in is major version 5, minor version 1, and update version 1.

Type: *Number*

Access: *R*.

---

## printParams Object

This object controls printing parameters that affect any document printed via JavaScript. Changing this object does not change the user preferences or make any permanent changes to the document.

In Acrobat version 6.0, `doc.print` takes a **printParams** object as its argument. You can obtain **printParams** object from `doc.getPrintParams`. The returned object can then be modified.

Many of the **printParams** properties take integer constants as values, which you can access using **constants**. For example:

```
// get the printParams object of the default printer
var pp = this.getPrintParams();
// set some properties
pp.interactive = pp.constants.interactionLevel.automatic;
pp.colorOverride = pp.colorOverrides.mono;
// print
this.print(pp);
```

The **constants** object properties are all Integers, and are all Read access.

---

## PrintParams Properties

### binaryOK

6.0			
-----	--	--	--

**true** if a binary channel to the printer is supported. The default is **true**.



*Type: Boolean**Access: R/W.*

## bitmapDPI

6.0			
-----	--	--	---

The dots per inch (DPI) to use when producing bitmaps or rasterizing transparency. Valid range is 1 to 9600. If the document protections specify a maximum printing resolution, the lower of the two values is used. The default is 300. Illegal values are treated as 300. See also [gradientDPI](#).

*Type: Integer**Access: R/W.*

## colorOverride

6.0			
-----	--	--	--

Whether to use color override. Values are the properties of the [constants colorOverrides Object](#). Illegal values are treated as **auto**, the default value.

**NOTE:** This property is supported on Windows platforms only.

### colorOverrides Object


Property	Description
<b>auto</b>	Let Acrobat decide color overrides. This is the default.
<b>gray</b>	Force color to grayscale.
<b>mono</b>	Force color to monochrome.

*Type: Integer constant*    *Access: R/W.*

### Example

```
var pp = this.getPrintParams();
pp.colorOverride = pp.constants.colorOverrides.mono;
this.print(pp);
```

## colorProfile

6.0			
-----	--	--	---

The color profile to use. A list of available color spaces can be obtained from the [printColorProfiles](#). The default is "Printer/PostScript Color Management"

Type: String

Access: R/W.

### constants

6.0			
-----	--	--	--

Each instance of a **printParams** object inherits this property, which is a wrapper object for holding various constant values. The **constants** object property values are all Integers, and are all Read access. The values are listed with the **printParams** properties to which they apply.

The **constants** objects are used to specify option values of some of the other properties of the **printParams** object, as shown in the following table:

constant object	contains constant values for printParams property
<b>colorOverrides</b>	<a href="#">colorOverride</a>
<b>fontPolicies</b>	<a href="#">fontPolicy</a>
<b>handling</b>	<a href="#">pageHandling</a>
<b>interactionLevel</b>	<a href="#">interactive</a>
<b>nUpPageOrders</b>	<a href="#">nUpPageOrder</a>
<b>printContents</b>	<a href="#">printContent</a>
<b>flagValues</b>	<a href="#">flags</a>
<b>rasterFlagValues</b>	<a href="#">rasterFlags</a>
<b>subsets</b>	<a href="#">pageSubset</a>
<b>tileMarks</b>	<a href="#">tileMark</a>
<b>usages</b>	<a href="#">usePrinterCRD</a> <a href="#">useTlConversion.</a>

Type: object

Access: R.

### downloadFarEastFonts

6.0			
-----	--	--	--

When **true**, send Far East fonts to the printer if needed. Set to **false** if printer has Far East fonts but incorrectly reports it needs them. The default is **true**.

Type: Boolean

Access: R/W.

## fileName

6.0			
-----	--	--	--

If not empty, the device-independent pathname for a filename to be used instead of sending the print job to the printer (Print to File). The pathname may be relative to the location of the current document. When printing to a file, if the interaction level (See [interactive](#)) is set to **full**, it is lowered to **automatic**. The default value is the empty string.

**NOTE:** Printing to a file produces output suitable for the printer, for example, Postscript or GDI commands.

**NOTE:** When [printerName](#) is an empty string and [fileName](#) is nonempty the current document is saved to disk as a PostScript file.

*Type: String*

*Access: R/W.*

### Example

```
var pp = this.getPrintParams();
pp.fileName = "/c/print/myDoc.prn";
this.print(pp);
```

### Example 2

Save the current document as a PostScript file.

```
var pp = this.getPrintParams();
pp.fileName = "/c/temp/myDoc.ps";
pp.printerName = "";
this.print(pp);
```

## firstPage

6.0			
-----	--	--	--

The first 0-based page number of the document to print. The first page of any document is 0, regardless of page number labels. Values out of the document page range are treated as 0. The default value is 0.

See also [lastPage](#).

*Type: Integer*

*Access: R/W.*

### Example

```
var pp = this.getPrintParams();
pp.firstPage = 0;
pp.lastPage = 9;
this.print(pp);
```

## flags

6.0			
-----	--	--	--

A bit field of flags to control printing. These flags can be set or cleared using bitwise operations through the [constants flagValues Object](#).

Zero or more flags can be set; unsupported flags are ignored. The flags default to those set by user preferences.

### flagValues Object

Where **X** appears in the **Reader** column, the property is not available for any version of the Adobe Reader.

Property	Reader	Description
<code>applyOverPrint</code>	<b>X</b>	Do overprint preview when printing, turn off if print natively supports overprinting
<code>applySoftProofSettings</code>	<b>X</b>	Use the softProofing settings before doing color management
<code>applyWorkingColorSpaces</code>	<b>X</b>	Apply working color spaces when printing
<code>emitHalftones</code>	<b>X</b>	Emit the halftones specified in the document
<code>emitPostScriptXObject</code>	<b>X</b>	PostScript only, do include PostScript XObjects' content in output
<code>emitFormsAsPSForms</code>	<b>X</b>	Converts Form XObjects to PS forms. The default is off.
<code>maxJP2KRes</code>	<b>X</b>	Use the maximum resolution of JPeg2000 images instead of the best matching resolution.
<code>setPageSize</code>		Enable setPageSize, choose paper tray by PDF page size
<code>suppressBG</code>	<b>X</b>	Do not emit the BlackGeneration in the document
<code>suppressCenter</code>		Do not center the page
<code>suppressCJKFontSubst</code>	<b>X</b>	Suppress CJK Font Substitution on Printer—does not apply when <code>kAVEmitFontAllFonts</code> is used
<code>suppressCropClip</code>		Do not emit the cropbox page clip
<code>suppressRotate</code>		Do not rotate the page
<code>suppressTransfer</code>	<b>X</b>	Do not emit the transfer functions in the document
<code>suppressUCR</code>	<b>X</b>	Do not emit the UnderColorRemovals in the document

Property	Reader	Description
<code>useTrapAnnots</code>	X	Print TrapNet and PrinterMark annotations, even if printing "document only".
<code>usePrintersMarks</code>	P	Print PrinterMark annotations, even if printing "document only".

*Type: Integer*

*Access: R/W.*

### Example 1

Check the "Apply Proof Settings" checkbox Output options in the Advanced Printing Setup dialog.

```
pp = getPrintParams();
fv = pp.constants.flagValues;
// or pp.flags |= fv.applySoftProofSettings;;
pp.flags = pp.flags | fv.applySoftProofSettings;
this.print(pp);
```

### Example 2

Uncheck "Auto-Rotate and Center" (checked by default) in the Print dialog.

```
pp = getPrintParams();
fv = pp.constants.flagValues;
pp.flags |= (fv.suppressCenter | fv.suppressRotate);
this.print(pp);
```

### Example 3

Check "Emit Undercolor Removal/Black Generation" checkbox of the PostScript Options in the Advanced Printing Setup dialog.

```
pp = getPrintParams();
fv = pp.constants.flagValues;
pp.flags &= ~(fv.suppressBG | fv.suppressUCR);
this.print(pp);
```

## fontPolicy

6.0			
-----	--	--	--

Sets the font policy. The value of the **fontPolicy** property is set through the [constants fontPolicies Object](#). The default is **pageRange**.

Type: Integer

Access: R/W.

### fontPolicies Object

Property	Description
<b>everyPage</b>	Emit needed fonts before every page, free all fonts after each page. This produces the largest, slowest print jobs, but requires the least amount of memory from the printer.
<b>jobStart</b>	Emit all fonts used at the beginning of the print job, free them at the end of the print job. This produces the smallest, fastest print jobs, but requires the most memory from the printer.
<b>pageRange</b>	(Default) Emit fonts before the first page that uses them, free them after the last page that uses them. This also produces the smallest, fastest print jobs, and can use less memory. However, the produced print job must be printed as produced due to page ordering.  <b>NOTE:</b> <b>pageRange</b> can be a good compromise between speed and memory, but do not use it if the postscript pages will be programmatically reordered afterwards.

## gradientDPI

6.0			X
-----	--	--	---

The dots per inch to use when rasterizing gradients. This value can generally be set lower than [bitmapDPI](#) because it affects areas to which the eye is less sensitive. It must be set from 1 to 9600. Illegal values are treated as 150. If the document protections specify a maximum printing resolution, the lower of the two values will be used. The default value is 150.


Type: Integer

Access: R/W.

## interactive

6.0			
-----	--	--	--

Sets the level of interaction between the user and the print job. The value of the **interactive** property is set through the [constants InteractionLevel Object](#). The default is **full**.

(Security , version 7.0) Non-interactive printing can only be executed during batch and console events. Printing is made non-interactive by setting **bUI** is to **false** or by setting the **interactive** property to silent, e.g.,

```
var pp = this.getPrintParams();
pp.interactive = pp.constants.interactionLevel.silent;
```

Outside of batch and console events, the values of **bUI** and of **interactive** are ignored, and a print dialog will always be presented.

**NOTE:**

See also [Privileged versus Non-privileged Context](#).

Type: Integer

Access: R/W.

### InteractionLevel Object

Property	Description
<b>automatic</b>	No print dialog is displayed. During printing a progress monitor and cancel dialog is displayed and removed automatically when printing is complete.
<b>full</b>	Displays the print dialog allowing the user to change print settings and requiring the user to press OK to continue. During printing a progress monitor and cancel dialog is displayed and removed automatically when printing is complete.
<b>silent</b>	No print dialog is displayed. No progress or cancel dialog is displayed. Even error messages are not displayed.

### Example

```
var pp = this.getPrintParams();
pp.interactive = pp.constants.interactionLevel.automatic;
pp.printerName = "Adobe PDF";
this.print(pp);
```

## lastPage

6.0			
-----	--	--	--

The last 0-based page number of the document to print. The term “0-based” means the first page of any document is 0, regardless of page number labels. If the value is less than

**firstPage** or outside the legal range of the document, this reverts to the default value. The default value is the number of pages in the document less one.

Type: *Integer*                      Access: *R/W*.

See **firstPage** for an example.

## nUpAutoRotate

7.0			
-----	--	--	--

The **nUpAutoRotate** property is a boolean, if set to **true**, automatically rotates each page to match the page orientation to the available paper area during Multiple Pages Per Sheet printing. The default is **false**, but **nUpAutoRotate** obeys the print settings.

Multiple Pages Per Sheet is obtained by setting **pageHandling** to **nUp**.

Type: *Boolean*                      Access: *R/W*.

## nUpNumPagesH

7.0			
-----	--	--	--

When printing Multiple Pages Per Sheet, **nUpNumPagesH** sets is the number of pages to be layed out in the horizontal direction. The default is 2, but **nUpNumPagesH** obeys the print settings.

Multiple Pages Per Sheet is obtained by setting **pageHandling** to **nUp**.

Type: *Integer*                      Access: *R/W*.

### Example

Perform Multiple Pages Per Sheet printing on this document, set up desired parameters and print.

```
pp = this.getPrintParams();
pp.pageHandling = pp.constants.handling.nUp;
pp.nUpPageOrders = pp.constants.nUpPageOrders.Vertical;
pp.nUpNumPagesH = 3;
pp.nUpNumPagesV = 3;
pp.nUpPageBorder=true;
pp.nUpAutoRotate=true;
this.print(pp);
```



## nUpNumPagesV

7.0			
-----	--	--	--

When printing Multiple Pages Per Sheet, **nUpNumPagesV** is the number of pages to be layed out in the vertical direction. The default is 2, but **nUpNumPagesV** obeys the print settings.

Multiple Pages Per Sheet is obtained by setting [pageHandling](#) to **nUp**.

*Type: Integer*                      *Access: R/W.*

See [nUpNumPagesH](#) for an example.

## nUpPageBorder

7.0			
-----	--	--	--

The **nUpPageBorder** property is a boolean, if set to **true**, draws and prints a page boundary around each of the page during Multiple Pages Per Sheet printing. The default is **false**, but **nUpPageBorder** obeys the print settings.

Multiple Pages Per Sheet is obtained by setting [pageHandling](#) to **nUp**.

*Type: Boolean*                      *Access: R/W.*

See [nUpNumPagesH](#) for an example.

## nUpPageOrder

7.0			
-----	--	--	--

When printing multiple pages per sheet, the **nUpPageOrder** property determines how the multiple pages are layed out on the sheet. The value of the **nUpPageOrder** property is set through the [constants nUpPageOrders Object](#). The default is **Horizontal**, but **nUpPageOrder** obeys the print settings.

Multiple Pages Per Sheet is obtained by setting [pageHandling](#) to **nUp**.

*Type: Integer*                      *Access: R/W.*

### nUpPageOrders Object

Property	Description
<b>Horizontal</b>	Pages are placed from left to right, from top to bottom.
<b>HorizontalReversed</b>	Pages are placed from right to left, from top to bottom.
<b>Vertical</b>	Pages are placed from top to bottom, from left to right.
<b>VerticalReversed</b>	Pages are placed from top to bottom, from right to left.

#### Example

Perform Multiple Pages Per Sheet printing on this document, set up desired parameters and print.

```
pp = this.getPrintParams();
pp.pageHandling = pp.constants.handling.nUp;
pp.nUpPageOrders = pp.constants.nUpPageOrders.Horizontal;
pp.nUpNumPagesH = 2;
pp.nUpNumPagesV = 2;
pp.nUpPageBorder=true;
this.print(pp);
```

### pageHandling

6.0			
-----	--	--	--

Takes one of four values. The value of the **pageHandling** property is set through the [constants handling Object](#). If set to an illegal value it is treated as **shrink**. The default is **shrink**.

Type: Integer

Access: R/W.

**handling Object**

Property	Reader	Description
<b>none</b>		No page scaling is applied.
<b>fit</b>		Pages are enlarged or shrunk to fit the printer's paper.
<b>shrink</b>		Small pages are printed small, large pages are shrunk to fit on the printer's paper.
<b>tileAll</b>	X	All pages are printed using tiling settings. One use of this is to turn a normal sized page into a poster by setting tile zoom greater than 1.
<b>tileLarge</b>	X	Small or normal pages are printed original size, large pages are printed on multiple sheets of paper.
<b>nUp</b>		(Version 7.0) Rescale pages to print multiple pages per printer page. Properties related to Multiple Pages Per Sheet printing are <a href="#">nUpAutoRotate</a> , <a href="#">nUpNumPagesH</a> , <a href="#">nUpNumPagesV</a> , <a href="#">nUpPageBorder</a> and <a href="#">nUpPageOrder</a> .

**Example 1**

```
var pp = this.getPrintParams();
pp.pageHandling = pp.constants.handling.shrink;
this.print(pp);
```

**Example 2**

Perform Multiple Pages Per Sheet printing on this document, set up desired parameters and print.

```
pp = this.getPrintParams();
pp.pageHandling = pp.constants.handling.nUp;
pp.nUpPageOrders = pp.constants.nUpPageOrders.Horizontal;
pp.nUpNumPagesH = 2;
pp.nUpNumPagesV = 2;
pp.nUpPageBorder=true;
this.print(pp);
```

**pageSubset**

6.0			
-----	--	--	--

Select even, odd, or all the pages to print. The value of **pageSubset** is set through the [constants subsets Object](#). The default is **a11**.

Type: Integer

Access: R/W.

**subsets Object**

Property	Description
<b>all</b>	Print all pages in page range.
<b>even</b>	Print only the even pages. Page labels are ignored for this. The document is treated as if it were numbered 1 through n, the number of pages.
<b>odd</b>	Print only the odd pages.

**Example**

```
var pp = this.getPrintParams();
pp.pageSubset = pp.constants.subsets.even;
this.print(pp);
```

**printAsImage**

6.0			
-----	--	--	--

Set to **true** to send pages as large bitmaps. This can be slow and more jagged looking but can work around problems with a printer’s PostScript interpreter. Set **bitmapDPI** to increase or decrease the resolution of the bitmap. If interaction (see **interactive**) is **full**, the user’s printer preferences for **printAsImage** will be used. The default is **false**.

Type: Boolean                      Access: R/W.

**printContent**

6.0			
-----	--	--	--

Sets the contents of the print job. The value of the **printContent** property is set through the **constants printContents Object**. The default is **doc**.

Type: Integer                      Access: R/W.

**printContents Object**

Property	Description
<b>doc</b>	Emit the document contents. Document comments are not printed
<b>docAndComments</b>	Emit the document contents and comments.

Property	Description
<b>formFieldsOnly</b>	Emit the contents of form fields only. Useful for printing onto pre-printed forms.

**Example**

```
var pp = this.getPrintParams();
pp.interactive = pp.constants.interactionLevel.silent;
pp.printContent = pp.constants.printContents.formFieldsOnly;
this.print(pp);
```

**printerName**

6.0			
-----	--	--	--

Set or get the name of destination printer. The **printerName** property is a Windows-only feature; currently, the destination printer cannot be set through this property on the Mac.

By default, **printerName** is set to the name of the default printer. If set **printerName** to an empty string the default printer will be used. When **printerName** is an empty string *and* **fileName** is a nonempty string, the current document is saved to disk as a PostScript file. See **Example 2** below.

See also [app.printerNames](#).

Type: *String*

Access: *R/W*.

**Example 1**

```
var pp = this.getPrintParams();
pp.printerName = "hp officejet d series";
this.print(pp);
```

**Example 2**

Save the current document as a PostScript file.

```
var pp = this.getPrintParams();
pp.fileName = "/c/temp/myDoc.ps";
pp.printerName = "";
this.print(pp);
```

**psLevel**

6.0			
-----	--	--	--

Level of PostScript that is emitted to PostScript printers. Level 0 indicates to use the PostScript level of the printer. Level 1 is not supported. In addition to 0, current legal values of **psLevel** are 2 and 3. If the printer only supports PostScript level 1, **printAsImage** is set to **true**. Illegal values are treated as 3. The default value for **psLevel** is 3.

Type: Integer

Access: R/W.

## rasterFlags

6.0			<input checked="" type="checkbox"/>
-----	--	--	-------------------------------------

A bit field of flags. These flags can be set or cleared using bitwise operations through the [constants rasterFlagValues Object](#). The default is set by user preferences.

Type: Integer

Access: R/W.

### rasterFlagValues Object

Property	Reader	Description
<code>textToOutline</code>	<input checked="" type="checkbox"/>	Text converted to outlines can become thicker (especially noticeable on small fonts). If text is mixed into artwork with transparency it may be converted to outline during flattening, resulting in inconsistency with text that is not mixed into artwork. In this case turning on this option will ensure all text looks consistent.
<code>strokesToOutline</code>	<input checked="" type="checkbox"/>	Strokes converted to outlines can become thicker (especially noticeable on thin strokes). If strokes are mixed into artwork with transparency they may be converted to outlines during flattening, resulting in inconsistency with strokes that are not mixed into artwork. In this case turning on this option will ensure all strokes looks consistent.
<code>allowComplexClip</code>	<input checked="" type="checkbox"/>	Select this to ensure that the boundaries between vector artwork and rasterized artwork fall closely along object paths. Selecting this option reduces stitching artifacts that result when part of an object is flattened while another part of the object remains in vector form. However, selecting this option may result in paths that are too complex for the printer to handle.
<code>preserveOverprint</code>	<input checked="" type="checkbox"/>	Select this if you are printing separations and the document contains overprinted objects. Selecting this option generally preserves overprint for objects that are not involved in transparency and therefore improves performance. This option has no effect when printing composite. Turning this off might result in more consistent output since all overprinting will be flattened whether it is involved in transparency or not.

**Example 1**

Check the “Convert All Text to Outlines” checkbox in the Transparency Flattening option of the Advanced Print Setup.

```
pp = getPrintParams();
rf = pp.constants.rasterFlagValues;
pp.rasterFlags |= rf.textToOutline;
this.print(pp);
```

**Example 2**

Uncheck “Complex Clip Regions” (checked by default) in the Transparency Flattening option of the Advanced Print Setup.

```
pp = getPrintParams();
rf = pp.constants.rasterFlagValues;
pp.rasterFlags = pp.rasterFlags & ~rf.allowComplexClip;
// or pp.rasterFlags &= ~rf.allowComplexClip;
this.print(pp);
```

**reversePages**


6.0			
-----	--	--	--

Set to **true** to print pages in reverse order (last to first). The default value is **false**.

Type: Boolean

Access: R/W.

**tileLabel**

6.0			
-----	--	--	---

Label each page of tiled output. Labeled pages indicate row and column, filename, and print date. The default is **false**.

Type: Boolean

Access: R/W.

**tileMark**

6.0			
-----	--	--	---

Tile marks indicate where to cut the page and where overlap occurs. The value is set through the [constants tileMarks Object](#). If set to an illegal value it is treated as **none**. The default is **none**.

Type: Integer

Access: R/W.

**tileMarks Object**

Property	Description
<b>none</b>	No tile marks
<b>west</b>	Western style tile marks
<b>east</b>	Eastern style tile marks.

**tileOverlap**

6.0			X
-----	--	--	---

The number of points that tiled pages have in common. Value must be between 0 and 144. Illegal values are treated as 0. The default value is 0.

Type: Integer

Access: R/W.

**tileScale**

6.0			X
-----	--	--	---

The amount that tiled pages are scaled. Pages that are not tiled are unaffected by this value. Default is unscaled (1.0). Larger values increase the size of the printout (for example, 2.0 is twice as large, a value of 0.5 is half as large). The value of **tileScale** must be between 0.01 and 99.99. Illegal values are treated as 1.0, which is the default value.

Type: Number

Access: R/W.

**transparencyLevel**

6.0			X
-----	--	--	---

An integer value from 1 to 100 indicates how hard Acrobat tries to preserve high level drawing operators. A value of 1 indicates complete rasterization of the image which results in poor image quality but high speeds. A value of 100 indicates as much should be preserved as possible, but can result in slow print speeds. If set to an illegal value, 75 is used. When rasterizing, the **bitmapDPI** and **gradientDPI** values are used. The default value is 75.

Type: Integer

Access: R/W.



## usePrinterCRD

6.0			
-----	--	--	--

Takes one of three values. The value is set through the [constants usages Object](#). See also [usePrinterCRD](#); the two properties use the same values, but the interpretations are different.

Type: Integer

Access: R/W.

### usages Object

Property	Description for usePrinterCRD
<b>auto</b>	Let Acrobat decide if printer Color Rendering Dictionary should be used. Acrobat maintains a list of a handful of printers that have incorrect CRDs. Illegal values are treated as <b>auto</b> . The default is <b>auto</b> .
<b>use</b>	Use printer's Color Rendering Dictionary.
<b>noUse</b>	Do not use printer's Color Rendering Dictionary.

## useT1Conversion

6.0			
-----	--	--	--

Takes one of three values. The value of the **useT1Conversion** property is set through the [constants usages Object](#). See also [usePrinterCRD](#); the two properties use the same values, but the interpretations are different.

**NOTE:** This property is supported on Windows platforms only.

Type: Integer

Access: R/W.

This property uses the [usages Object](#) values as follows.

Property	Description for useT1Conversion
<b>auto</b>	Let Acrobat decide whether to disable converting Type 1 fonts to more efficient printer representations (for example, TrueType). Acrobat maintains a list of a handful of printers that have problems with these fonts. Illegal values are treated as <b>auto</b> . The default is <b>auto</b> .
<b>use</b>	Allow conversion of Type 1 fonts even if printer is known to have problems with alternative font representations.
<b>noUse</b>	Never convert Type 1 fonts to more efficient representations..

## Rendition Object

A Rendition contains information needed to play a media clip, including embedded media data (or a URL) and playback settings. It corresponds to a Rendition in the Acrobat authoring user interface.

A Rendition is a base type for either a MediaRendition or a MediaSelector. A function that accepts a Rendition can take either of these two types. The properties and methods described in this section are available for both MediaRendition and MediaSelector. Use `Rendition.type` to distinguish between MediaRendition and MediaSelector.

## Rendition Object Properties

### altText

6.0				
-----	--	--	--	--

The `rendition.altText` property is the alternate text string for the rendition (an empty string if no alternate text was specified). This property is available only if the `type` of the `rendition` is `app.media.renditionType.media` (a MediaRendition).

*Type: String*

*Access: R.*

### Example

Get the `altText` of a rendition.

```
this.media.getRendition("myClip").altText;
```

See the examples that follow `app.media.getAltTextSettings()`

### doc

6.0				
-----	--	--	--	--

The `Rendition.doc` property is a reference to the document that contains the Rendition.

*Type: Doc*

*Access: R.*

## fileName

6.0				
-----	--	--	--	--

The **rendition.fileName** property returns an empty string if the media is embedded, and the filename or URL of the media if it's not embedded. This property is available only if the **type** of the **rendition** is **app.media.renditionType.media**.

Type: String

Access: R.

## type

6.0				
-----	--	--	--	--

The **Rendition.type** is an **app.media.renditionType** value indicating the type of rendition.

Currently, there are two types: **MediaRendition** and **RenditionList**:

- When **Rendition.type** is equal to **app.media.renditionType.media**, the Rendition is a **MediaRendition**. A **MediaRendition** is an individual Rendition, as it appears in the Settings tab of the Multimedia Properties dialog of the UI.
- When **Rendition.type** is equal to **app.media.renditionType.selector**, the Rendition is a **RenditionList**. A **RenditionList** is an array of **MediaRendition**. The list is the one that appears in the Settings tab of the Multimedia Properties dialog of the UI.

Future versions of Acrobat may add more **renditionType** values, so JavaScript code should not assume that only the existing **app.media.renditionType** values may be encountered.

Type: Number

Access: R.

## uiName

6.0				
-----	--	--	--	--

**Rendition.uiName** contains the name of the Rendition as found in the **N** entry in its dictionary in the PDF file.

Type: String

Access: R.

### Example

The following is executed as a Rendition action.

```
console.println("Preparing to play \""
+ event.action.rendition.uiName + "\"");
```

See the [Event Object](#) for a description of **event.action.rendition**.

---

## Rendition Object Methods

### getPlaySettings

6.0				
-----	--	--	--	--

Creates and returns a [MediaSettings Object](#) that can be used to create a `MediaPlayer` object.

This method is available only for a `MediaRendition`.

#### Parameters

---

<b>bGetData</b>	(optional) A boolean, which if <b>true</b> , the <code>MediaSettings</code> object returns the <code>MediaData</code> (See <a href="#">MediaSettings.data</a> ).
-----------------	--

---

#### Returns

A [MediaSettings Object](#)

**NOTE:** `app.media.getAltTextSettings()` calls `getPlaySettings(false)` to obtain the correct settings to display alternate text, see the `media.js`.

This `MediaSettings` object includes these properties:

```
autoPlay
baseURL (if specified in rendition)
bgColor
bgOpacity
data (if bGetData is true)
duration
endAt
layout
monitorType
palindrome
showUI
rate
repeat
startAt
visible
volume
windowType
```

In the current version of Acrobat, all of these properties are present in the settings object (except as noted above), and `null` is used when values such as [startAt](#) are unspecified. This may change in the future to return only those values which are actually specified, with defaults assumed for the rest.

#### Example:

```
// Get the MediaSettings for this Rendition
```

```

var settings = myRendition.getPlaySettings();
if( settings.startAt !== null ) // Do NOT use this strict comparison!
...
if( settings.startAt ) // This is OK
...

```

See `app.media.getAltTextSettings()` and `app.media.openPlayer()` for examples of usage.

## select

6.0				
-----	--	--	--	--

**Rendition.select()** selects a media player to play a `MediaRendition` or a `RenditionSelector`. If the `Rendition` is a `RenditionSelector`, **select()** examines every `MediaRendition` contained within and selects the most suitable one. (See [type](#) for a description of `RenditionSelector` and `MediaRendition`.)

The return value is a [MediaSelection Object](#) that can be used to create a [MediaSettings Object](#). This object can then be used to create a [MediaPlayer Object](#).

### Parameters

<b>bWantRejects</b>	(optional) If <b>bWantRejects</b> is <b>true</b> , the <b>rejects</b> property of the resulting <code>MediaSelection</code> will contain information about media players that were rejected during the selection process.
<b>oContext</b>	(optional) <b>oContext</b> is a <code>MediaSelection.selectContext</code> value from a previous <b>Rendition.select()</b> call. This parameter allows you to write a loop that calls <b>Rendition.select()</b> repeatedly until you find a media player that satisfies any selection criteria that you want to test in JavaScript code.

### Returns

A [MediaSelection Object](#)

### Example 1

Get a usable `MediaSelection` for this `Rendition`

```
var selection = rendition.select();
```

### Example 2

Get the name of the selected rendition. This script is executed from a `Rendition` action event.

```

var selection = event.action.rendition.select();
console.println( "Preparing to play " + selection.rendition.uiName);

```

## testCriteria

6.0				
-----	--	--	--	--

This method tests the Rendition against any criteria that are specified in the PDF file, such as minimum bandwidth, and returns a boolean indicating whether the Rendition satisfied all of those criteria.

### Parameters

None

### Returns

Boolean

---

## RDN Generic Object

This generic object represents a Relative Distinguished Name. It is used by `securityHandler.newUser` and the `certificate.issuerDN` and `subjectDN` properties.

It has the following properties.

Property	Type	Access	Description
<code>c</code>	String	R	Country or Region. Must be a two-character upper case ISO 3166 standard string (for example, 'US')
<code>cn</code>	String	R	Common name (for example, 'John Smith')
<code>o</code>	String	R	Organization name (for example, 'Adobe Systems Inc.')
<code>ou</code>	String	R	Organizational unit (for example, 'Acrobat Engineering')
<code>e</code>	String	R	Email address (for example, 'jsmith@adobe.com')

---

## Report Object

The Report object allows the user to programmatically generate PDF documents suitable for reporting with JavaScript. Use the `Report` constructor to create a `Report` object; for example,

```
var rep = new Report();
```

The properties and methods can then be used to write and format a report.

## Report Properties

### absIndent

5.0	Ⓓ		✗	✗
-----	---	--	---	---

Controls the absolute indentation level. It is desirable to use indent/outdent only whenever possible, as those calls correctly handle indentation overflows.

If a report is indented past the middle of the page, the effective indent is set to the middle. Note that [divide](#) does a little squiggly bit to indicate that it's been indented too far.

*Type: Number*

*Access: R/W.*

### color

5.0	Ⓓ		✗	✗
-----	---	--	---	---

Controls the color of any text and any divisions written into the report.

Text is written to the report with [writeText](#) and divisions (horizontal rules) are written using [divide](#).

*Type: Color*

*Access: R/W.*

#### Example

```
var rep = new Report();
rep.size = 1.2;
rep.color = color.blue;
rep.writeText("Hello World!");
```

### size

5.0	Ⓓ		✗	✗
-----	---	--	---	---

Controls the size of any text created by [writeText](#). It is a multiplier. Text size is determined by multiplying the **size** property by the default size for the given style.

*Type: Number*

*Access: R/W.*

#### Example

Write a "Hello World!" document.

```
var rep = new Report();
rep.size = 1.2;
rep.writeText("Hello World!");
```

## style

6.0	Ⓓ		✗	✗
-----	---	--	---	---

This property controls the style of the text font for the text created by `writeText`. Values of **style** are

DefaultNoteText  
NoteTitle

### Example

```
var rep = new Report();
rep.size = 1.2;
rep.style = "DefaultNoteText";
rep.writeText("Hello World!");
rep.open("My Report");
```

---

## Report Methods

### breakPage

5.0	Ⓓ		✗	✗
-----	---	--	---	---

Ends the current page and begins a new one.

#### Parameters

None

#### Returns

Nothing

### divide

5.0	Ⓓ		✗	✗
-----	---	--	---	---

Writes a horizontal rule across the page at the current location with the given width. The rule goes from the current indent level to the rightmost edge of the bounding box. If the indent level is past the middle of the bounding box, the rule has a squiggly bit to show this.

#### Parameters

---

<b>nWidth</b>	(optional) The horizontal rule width to use.
---------------	--

---



**Returns**

Nothing

**indent**

5.0	Ⓓ		✘	✘
-----	---	--	---	---

Increments the current indentation mark by **nPoints** or the default amount. If a report is indented past the middle of the page, the effective indent is set to the middle. Note that [divide](#) makes a squiggly bit to indicate that it has been indented too far.

See [writeText](#) for an example of usage.

**Parameters**

---

<b>nPoints</b>	(optional) The number of points to increment the indentation mark.
----------------	--

---

**Returns**

Nothing

**outdent**

5.0	Ⓓ		✘	✘
-----	---	--	---	---

The opposite of indent; that is, decrements the current indentation mark by **nPoints** or the default amount.

See [writeText](#) for an example of usage.

**Parameters**

---

<b>nPoints</b>	(optional) The number of points to decrement the indentation mark.
----------------	--

---

**Returns**

Nothing

**open**

5.0	Ⓓ		✘	✘
-----	---	--	---	---

Ends report generation, opens the report in Acrobat and returns a [Doc Object](#) that can be used to perform additional processing of the report.

**Parameters**


---

<b>cTitle</b>	The report title.
---------------	-------------------

---

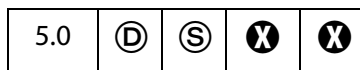
**Returns**

A [Doc Object](#).


**Example**

```
var docRep = rep.open("myreport.pdf");
docRep.info.Title = "End of the month report: August 2000";
docRep.info.Subject = "Summary of comments at the August meeting";
```

See [writeText](#) for a more complete example.

**save**

Ends report generation and saves the report to the specified path.

**NOTE:** (Security Privileged versus Non-privileged Context. The [Event Object](#) contains a discussion of Acrobat JavaScript events.

**Parameters**


---

<b>cDIPath</b>	The device-independent path.
<b>cFS</b>	(optional) The file system. The only value for <b>cFS</b> is "CHTTP"; in this case, the <b>cDIPath</b> parameter should be an URL. This parameter is only relevant if the web server supports WebDAV.

---

**Returns**

Nothing

**Example 1**

```
rep.save("/c/myReports/myreport.pdf");
```

**Example 2**

```
rep.save({
  cDIPath: "http://www.mycompany.com/reports/WebDAV/myreport.pdf",
  cFS: "CHTTP"
});
```

## mail

5.0	Ⓓ		✗	✗
-----	---	--	---	---

Ends report generation and mails the report.

See also [mailGetAdrs](#), [app.mailMsg](#), [doc.mailForm](#), and [fdf.mail](#).

### Parameters

<b>bUI</b>	(optional) Whether to display a user interface. If <b>true</b> (the default) the rest of the parameters are used to seed the compose-new-message window that is displayed to the user. If <b>false</b> , the <b>cTo</b> parameter is required and all others are optional.
<b>cTo</b>	(optional) A semicolon-separated list of recipients for the message.
<b>cCc</b>	(optional) A semicolon-separated list of CC recipients for the message.
<b>cBcc</b>	(optional) A semicolon-separated list of BCC recipients for the message.
<b>cSubject</b>	(optional) The subject of the message. The length limit is 64k bytes.
<b>cMsg</b>	(optional) The content of the message. The length limit is 64k bytes.

### Returns

Nothing

## Report

5.0	Ⓓ		✗	✗
-----	---	--	---	---

A constructor. Creates a new **Report** object with the given media and bounding boxes (values are defined in points or 1/72 of an inch). Defaults to a 8.5 x 11 inch media box and a bounding box that is indented .5 inches on all sides from the media box.

### Parameters

<b>aMedia</b>	(optional) The media type.
<b>aBBox</b>	(optional) The bounding box size.

### Returns

Nothing

## writeText

5.0	Ⓓ		✗	✗
-----	---	--	---	---

Writes out a block of text to the report. Every call is guaranteed to begin on a new line at the current indentation mark. Correctly wraps Roman, CJK, and WGL4 text.

### Parameters

<b>String</b>	The block of text to use.
---------------	---------------------------

### Example

```
// Get the comments in this document, and sort by author
this.syncAnnotScan();
annots = this.getAnnots({nSortBy: ANSB_Author});

// open a new report
var rep = new Report();

rep.size = 1.2;
rep.color = color.blue;
rep.writeText("Summary of Comments: By Author");
rep.color = color.black;
rep.writeText(" ");
rep.writeText("Number of Comments: " + annots.length);
rep.writeText(" ");

var msg = "\200 page %s: \"%s\"";
var theAuthor = annots[0].author;
rep.writeText(theAuthor);
rep.indent(20);
for (var i=0; i < annots.length; i++) {
    if (theAuthor != annots[i].author) {
        theAuthor = annots[i].author;
        rep.writeText(" ");
        rep.outdent(20);
        rep.writeText(theAuthor);
        rep.indent(20);
    }
}
rep.writeText(util.printf(msg, 1 + annots[i].page, annots[i].contents));
}

// now open the report
var docRep = rep.open("myreport.pdf");
docRep.info.Title = "End of the month report: August 2000";
docRep.info.Subject = "Summary of comments at the August meeting";
```

See the file `Annots.js` for additional examples of the Report object.

## Row Generic Object

This generic JS object contains the data from every column in a row. It is returned by `statement.getRow`. It contains the following properties:

Property	Type	Access	Description
<code>columnArray</code>	Array	R	An array of <a href="#">Column Generic Objects</a> . This is equivalent to what <code>statement.getColumnArray</code> would return if called on the same <code>statement</code> at the same time that this <code>row</code> object was created.
<code>column properties</code>	any	R	There is a property corresponding to each column selected by the query, containing the data for that row in that column.

## ScreenAnnot Object

A `ScreenAnnot` is a rectangular area within a PDF document viewed on the display screen. A `ScreenAnnot` may have Renditions and RenditionActions associated with it for multimedia playback.

## ScreenAnnot Object Properties

### altText

6.0				
-----	--	--	--	--

The `annot.altText` property is the alternate text string for `annot` (an empty string if no alternate text was specified).

Type: *String*

Access: *R*.

### Example

Get an `annot` and write its `altText` to the debug console.

```
var annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
console.println( "annot.altText = " + annot.altText );
```

## alwaysShowFocus

6.0				
-----	--	--	--	--

Normally, a ScreenAnnot shows and hides a focus rectangle to indicate whether it has the keyboard focus. If **ScreenAnnot.alwaysShowFocus** is **true**, the focus rectangle is displayed by the ScreenAnnot even if it does not have the focus. This is used for docked media playback, so that the focus rectangle of the annot can remain visible even though the media player actually has the keyboard focus.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: Boolean

Access: R/W.

## display

6.0				
-----	--	--	--	--

Same as **Field.display**, as documented in the *Acrobat JavaScript Scripting Reference*.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: Integer

Access: R/W.

### Example

Hide the annot.

```
var annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
annot.display = display.hidden;
```

## doc

6.0				
-----	--	--	--	--

**ScreenAnnot.doc** is a reference to the document that contains the ScreenAnnot.

Type: Doc object

Access: R.

## events

6.0				
-----	--	--	--	--

**ScreenAnnot.events** is an [Events Object](#) containing the event listeners that are attached to a ScreenAnnot.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: [Events Object](#) Access: *R/W*.

### Example

Create a simple focus event listener.

```
var annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
var myFocusEvent = {
  onFocus: function () {
    console.println("Focusing...");
  }
};
annot.events.add( myFocusEvent );
```

This event listener can be removed at a later time by executing the following code.

```
annot.events.remove( myFocusEvent );
```

## extFocusRect

6.0				
-----	--	--	--	--

When a ScreenAnnot draws a focus rectangle, the rectangle normally encloses only the ScreenAnnot itself. If **extFocusRect** is specified, then the ScreenAnnot takes the union of its normal rectangle and **extFocusRect**, and it uses the resulting rectangle to draw the focus rectangle.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: *Array of Number of length 4*

Access: *R/W*.

## innerDeviceRect

6.0				
-----	--	--	--	--

**ScreenAnnot.innerDeviceRect** and **ScreenAnnot.outerDeviceRect** define the interior and exterior rectangles of the ScreenAnnot as it appears in the current page view.

Type: *Array of Number of length 4*

Access: *R*.

### Example

Get the **innerDeviceRect**.

```
annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
console.println("annot.innerDeviceRect = "
  + annot.innerDeviceRect.toString() );
```

## noTrigger

6.0				
-----	--	--	--	--

If **ScreenAnnot.noTrigger** is **true**, then the screen annot cannot be triggered through the Acrobat user interface. Typically, clicking the mouse on a Screen Annot starts playback of a media player; **noTrigger** suppresses this.

This property is not saved in the PDF file; if you change it, the change affects the current session only.

Type: Boolean

Access: R/W.

### Example

Use form buttons to control the media clip, so turn off interaction with annot.

```
annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
annot.noTrigger = true;
```

## outerDeviceRect

6.0				
-----	--	--	--	--

**ScreenAnnot.innerDeviceRect** and **ScreenAnnot.outerDeviceRect** define the interior and exterior rectangles of the ScreenAnnot as it appears in the current page view.

Type: Array of Number of length 4

Access: R.

## page

6.0				
-----	--	--	--	--

**ScreenAnnot.page** is the page number of the PDF file in which the ScreenAnnot is located.

Type: Number

Access: R.

## player

6.0				
-----	--	--	--	--

**ScreenAnnot.player** is a reference to the MediaPlayer associated with a ScreenAnnot. This property exists only for a [ScreenAnnot Object](#) that is connected to a MediaPlayer. The property is set by **MediaPlayer.open()** or by methods that call **open()** indirectly, such as **app.media.openPlayer()**.



*Type: ScreenAnnot**Access: R/W.***rect**

6.0	ⓓ			
-----	---	--	--	--

**ScreenAnnot.rect** is the rectangle of the ScreenAnnot in default user coordinates. Changing this property dirties the PDF file, and the new setting will be saved if the PDF file is saved. The [innerDeviceRect](#) and [outerDeviceRect](#) properties are also updated to reflect the new rectangle.

*Type: Array of Number of length 4**Access: R/W.***Example**

Adjust the position of the annot slightly.

```
var annot = this.media.getAnnot({ nPage:0, cAnnotTitle: "myScreen" });
var aRect = annot.rect;
aRect[0] += 10;
aRect[2] += 10;
annot.rect = aRect;
```

---

**ScreenAnnot Object Methods****hasFocus**

6.0				
-----	--	--	--	--

**ScreenAnnot.hasFocus ()** tells whether the screen annot currently has the keyboard focus.

**Parameters**

None

**Returns**

Boolean

**setFocus**

6.0				
-----	--	--	--	--

**ScreenAnnot.setFocus ()** sets the keyboard focus to the screen annot. The focus is set synchronously (before setFocus returns) if it is safe to do so. If it is unsafe to set the focus synchronously (e.g. when the property is changed within an on event method), then

**bAllowAsync** determines what happens: If **true**, the focus will be set asynchronously during idle time; if **false** or omitted, the focus remains unchanged.

The return value is **true** if the operation was performed synchronously, or **false** if it was deferred to be performed asynchronously.

### Parameters

---

<b>bAllowAsync</b>	(optional) A boolean which determines the behavior of <b>setFocus ()</b> when it is not safe to set the focus synchronously. If <b>true</b> , the focus will be set asynchronously during idle time; if <b>false</b> or omitted, the focus remains unchanged. The default is <b>false</b> .
--------------------	---

---

### Returns

Boolean

---

## Search Object

5.0			
-----	--	--	--

The **search** object is a static object that accesses the functionality provided by the Acrobat Search plug-in. This plug-in must be installed in order to interface with the **search** object (see [available](#)).

See also the [Index Object](#), which is returned by some of the methods of the **search** object.

The results for [query](#) calls are displayed in the Find dialog of Acrobat.

**NOTES:** Acrobat 7.0 indexes are incompatible with the search engines of Acrobat 5.0 and prior versions.

In Acrobat 7.0, searching indexes created by versions of Acrobat 5.0 and prior is not possible on the Mac platform.

---

## Search Properties

### attachments

7.0			
-----	--	--	--

Determines whether any PDF file attachments should be searched along with the base document. The default is **false**.

This property is ignored on the Mac platform when searching a document from within the Safari web browser. As a result, attachments are not searched inside Safari.

*Type: Boolean*

*Access: R/W.*

## available

5.0			
-----	--	--	--

Returns **true** if the Search plug-in is loaded and query capabilities are possible. A script author should check this boolean before performing a query or other search object manipulation.

*Type: Boolean*

*Access: R.*

### Example

Make sure the search object exists and is available.

```
if (typeof search != "undefined" && search.available) {
    search.query("Cucumber");
}
```

## docInfo

6.0			
-----	--	--	--

Whether the document Information is searched for the query. The default is **false**.

*Type: Boolean*

*Access: R/W.*

## docText

6.0			
-----	--	--	--

Whether the document text is searched for the query. The default is **true**.

*Type: Boolean*

*Access: R/W.*

## docXMP

6.0			
-----	--	--	--

Whether document level XMP metadata is searched for the query. The default is **false**.

*Type: Boolean*

*Access: R/W.*

## bookmarks

6.0			
-----	--	--	--

Whether bookmarks are searched for the query. The default is **false**

Type: Boolean

Access: R/W.

## ignoreAccents

7.0			
-----	--	--	--

Whether accents and diacritics are ignored while searching the query term. The default is **false**.

Type: Boolean

Access: R/W.

## ignoreAsianCharacterWidth

6.0			
-----	--	--	--

Whether the Kana characters in the document exactly match the search query. The default is **false**.

Type: Boolean

Access: R/W.

## indexes

5.0		Ⓢ	
-----	--	---	--

Returns an array of all of the [Index Objects](#) currently accessible by the search engine.

**NOTE:** (SecurityⓈ, version 7.0) This property can only be accessed during batch or console events. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

Type: Array

Access: R.

### Example

Enumerate all of the indexes and dump their names.

```
for (var i = 0; i < search.indexes.length; i++) {
    console.println("Index[" + i + "]=", search.indexes[i].name);
}
```

## jpegExif

6.0			
-----	--	--	--

Whether EXIF data associated with JPEG images in the PDF is searched. The default is **false**.

*Type: Boolean*

*Access: R/W.*

## legacySearch

6.0			
-----	--	--	--

Returns **true** if the Search5.api plug-in is loaded. Search5.api plug-in provides the capability to search indexes generated by Acrobat Catalog in Acrobat 5.0 (or earlier version). See the sections in the Acrobat Online Guide pertaining to searching such indexes.

*Type: Boolean*

*Access: R.*

## markup

6.0			
-----	--	--	--

Whether markup (annotations) are searched for the query. The default is **false**.

*Type: Boolean*

*Access: R/W.*

## matchCase

Whether the search query is case sensitive. The default is **false**.

*Type: Boolean*

*Access: R/W.*

## matchWholeWord

6.0			
-----	--	--	--

Whether search finds only occurrences of complete words that are specified in the query. For example, when this option is set to **true**, if you search for the word "stick", the words "tick" and "sticky" will not be highlighted. The default is **false**.

*Type: Boolean*

*Access: R/W.*

## maxDocs

5.0			
-----	--	--	--

The maximum number of documents that will be returned as part of the search query. The default is 100 documents.

*Type: Integer*

*Access: R/W.*

## objectMetadata

7.0			
-----	--	--	--

This property determines whether object-level metadata should be searched. This is the same data which is visible by clicking **Tools** in the main menu of Acrobat 7.0 and selecting **Object Data -> Object Data Tool**.

The default is **false**.

*Type: Boolean*

*Access: R/W.*

## proximity

5.0			
-----	--	--	--

Whether the search query will reflect the proximity of words in the results ranking when performing the search that contains **AND** boolean clauses. The default is **false**. See the sections in the Acrobat Online Guide pertaining to Search capabilities for a more thorough discussion of proximity.

*Type: Boolean*

*Access: R/W.*

## proximityRange

7.0			
-----	--	--	--

The range of proximity search in number of words. This property will be used only if the property **proximity** is set to **true**. See the sections in the Acrobat Online Guide pertaining to Search capabilities for a more thorough discussion of proximity.

The default is 900 words. The valid values of this parameter are any non-zero positive integer.

*Type: Integer*

*Access: R/W.*

## refine

5.0			
-----	--	--	--

Whether the search query will take the results of the previous query and refine the results based on the next query. The default is **false**. See the sections in the Acrobat Online Guide pertaining to Search capabilities for a more thorough discussion of refining queries.

Type: *Boolean*

Access: *R/W*.

## soundex

ⓧ			
---	--	--	--

Whether the search query will take the sound of words (for example, MacMillan, McMillan, McMilon) into account when performing the search. The default is **false**. See the sections in the Acrobat Online Guide pertaining to Search capabilities for a more thorough discussion of soundex.

**NOTE:** Beginning with Acrobat 6.0, the use of this property is discouraged. This property has a value of **false** and access is restricted to read only.

Type: *Boolean*

Access: *R*.

## stem

5.0			
-----	--	--	--

Whether the search query will take the stemming of words (for example, run, runs, running) into account when performing the search. The default is **false**. See the sections in the Acrobat Online Guide pertaining to Search capabilities for a more thorough discussion of stemming.

Type: *Boolean*

Access: *R/W*.

## thesaurus

ⓧ			
---	--	--	--

Whether the search query will find *similar* words. For example, searching for "embellish" might yield "enhanced", "gracefully", or "beautiful". The default is **false**.

**NOTE:** Beginning with Acrobat 6.0, the use of this property is discouraged. This property has a value of **false** and access is restricted to read only.

*Type: Boolean**Access: R.*

## wordMatching

6.0			
-----	--	--	--

How individual words in the query will be matched to words in the document. Values are:

MatchPhrase  
 MatchAllWords  
 MatchAnyWord  
 BooleanQuery (*default*)

This property is relevant only when a query has more than one word. The **BooleanQuery** option is ignored when searching active document.

*Type: String**Access: R/W.*

## Search Methods

### addIndex

5.0	Ⓟ		
-----	---	--	--

Adds the specified index to the list of searchable indexes.

#### Parameters

<b>cDIPath</b>	A device-independent path to an index file on the user's hard drive. See "File Specification Strings", Section 3.10.1, in the <a href="#">PDF Reference</a> for the exact syntax of the path.
<b>bSelect</b>	(optional) Whether the index should be selected for searching.

#### Returns

An [Index Object](#).

#### Example

Adds the standard help index for Acrobat to the index list:

```
search.addIndex("/c/program files/adobe/acrobat 5.0/help/exchhelp.pdx",
true);
```



## getIndexForPath

5.0			
-----	--	--	--

Searches the index list and returns the **index** object whose path corresponds to the specified path.

### Parameters

<b>cDIPath</b>	A device-independent path to an index file on the user's hard drive. See "File Specification Strings", Section 3.10.1, in the <a href="#">PDF Reference</a> for the exact syntax of the path.
----------------	---

### Returns

The [Index Object](#) whose path corresponds to the specified path.

## query

5.0			
-----	--	--	--

Searches the specified document or index for the specified text. Properties associated with the **search** object (such as **matchCase**, **matchWholeWord**, **stem**) may affect the result.

### Parameters

<b>cQuery</b>	The text for which to search.
<b>cWhere</b>	(optional) Specifies where the text should be searched. Values are: ActiveDoc Folder Index ActiveIndexes ( <i>default</i> )
<b>cDPIPath</b>	(optional) A device-independent path to a folder or Catalog index on the user's computer. See "File Specification Strings", Section 3.10.1, in the <a href="#">PDF Reference</a> for the exact syntax of the path. When <b>cWhere</b> is <b>Folder</b> or <b>Index</b> , this parameter is required.

### Returns

Nothing

**Examples**

Search for the word "Acrobat".

cWhere	Query
<b>ActiveIndexes</b>	<code>search.query("Acrobat");</code> // "ActiveIndexes" is the default. <code>search.query("Acrobat", "ActiveIndexes");</code>
<b>ActiveDoc</b>	<code>search.query("Acrobat", "ActiveDoc");</code>
<b>Folder</b>	<code>search.query("Acrobat", "Folder", "/c/myDocuments");</code> <code>search.query("Acrobat", "Folder", app.getPath("user", "documents"));</code> <code>search.query("Acrobat", "Folder", "//myserver/myDocuments");</code>
<b>Index</b>	<code>search.query("Acrobat", "Index", "/c/Myfiles/public/index.pdx");</code>

**removeIndex**

5.0	Ⓟ		
-----	---	--	--

Removes the specified **index** object from the index list.

**Parameters**

<b>index</b>	The <a href="#">Index Object</a> to remove from the index list.
--------------	---

**Returns**

Nothing

**Security Object**

5.0		Ⓢ	ⓧ	
-----	--	---	---	--

The security object is a static JavaScript object that exposes security-related PDF functions such as encryption and digital signatures. Security functions are performed using a [SecurityHandler Object](#) which is obtained from the security object using the [getHandler](#) method.

**NOTE:** (SecurityⓈ): The Security Object is available without restriction, including in Adobe Reader. The methods and properties of the Security Object can only be executed during batch, console or application initialization events including in Adobe Reader, except where otherwise stated. See also [Privileged versus Non-privileged Context](#). The [Event Object](#) contains a discussion of Acrobat JavaScript events.

## Security Constants

Several convenience strings are defined the Security object, beginning with Acrobat 7.0. The constants are held as properties of the wrapper objects listed below.

### HandlerName

These are constants used when determining what handler to use.

Property	Type	Access	Description
<b>StandardHandler</b>	String	R	This value can be specified in the <b>handler</b> property for a <a href="#">SecurityPolicy Object</a> which is based on the Standard (password-based) security handler.
<b>PPKLiteHandler</b>	String	R	This value can be specified in the <b>handler</b> property for a <a href="#">SecurityPolicy Object</a> which is based on the PPKLite (certificate-based) security handler. This value can also be passed to <b>security.getHandler</b> to create a new security context.
<b>APSHandler</b>	String	R	This the value specified in the <b>handler</b> property for a <a href="#">SecurityPolicy Object</a> which is based on the Adobe Policy Server security handler. This value can also be passed to <b>security.getHandler</b> to create a new security context.

### Example

The constant (string) **security.StandardHandler** is used to specify the **handler** property of the [SecurityPolicy Object](#).

```
security.getHandler(security.PPKLiteHandler, true);
```

### EncryptTarget

These are constants used when determining what data a policy is encrypting.

Property	Type	Access	Description
<b>EncryptTargetDocument</b>	String	R	This is one of the values that can be used in the <b>target</b> property of the <a href="#">SecurityPolicy Object</a> . This is used for a Security Policy which encrypts the entire document when applied.

Property	Type	Access	Description
<b>EncryptTargetAttachments</b>	String	R	This is one of the values that can be used in the <b>target</b> property of the <a href="#">SecurityPolicy Object</a> . It is used for a Security Policy which encrypts only the file attachments embedded within the document. This means the document can be opened, however the attachments cannot be opened without providing the correct security authentication. This is used for the eEnvelope workflows.

**Example:**

```
var filterOptions = { target: security.EncryptTargetAttachments };
security.chooseSecurityPolicy( { oOptions: filterOptions } );
```

---

## Security Properties

### handlers

5.0				
-----	--	--	--	--

Returns an array containing the language-independent names of the available security handlers that can be used for encryption or signatures.

See also [getSecurityPolicies](#).

Beginning with Acrobat 6.0, access to this property is unrestricted, to allow querying to see what handlers are available.

**Backward Compatibility Note:** In Acrobat 6.0, this call returned three handlers, "Adobe.PPKLite", "Adobe.PPKMS" and "Adobe.AAB". Starting Acrobat 7.0, all the functionality provided by Adobe.PPKMS has been rolled into Adobe.PPKLite, hence Adobe.PPKMS is no longer available as a separate handler.

Beginning with Acrobat 7.0, a new handler is available, "Adobe.APS". This handler is only used for authentication prior to calling any of the methods [encryptUsingPolicy](#), [getSecurityPolicies](#), or [chooseSecurityPolicy](#). It has no other valid usage currently.

Beginning with Acrobat 7.0, there are [Security Constants](#) defined on the security object for each of the handlers. (Except "Adobe.AAB", this handler will probably be deprecated in the near future so no constant was added for it.) These constants should be used whenever creating a new handler instance via [getHandler](#) or comparing against the handlers list.

*Type: Array**Access: R.***Example**

Get the list of security handlers available on this system:

```
for ( var i=0; i < security.handlers.length; i++ )
  console.println(security.handlers[i])
```


The output to the console might be

```
Adobe.APS
Adobe.PPKLite
Adobe.PPKMS
Adobe.AAB
```

**validateSignaturesOnOpen**

5.0	P	S	X
-----	---	---	---

Gets or sets the user-level preference that causes signatures to be automatically validated when a document is opened.

**NOTE:** (Security 

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

*Type: Boolean**Access: R/W.***Security Methods****chooseRecipientsDialog**

6.0		S	X	
-----	--	---	---	--

Opens a dialog that allows a user to choose a list of recipients. Returns an array of generic Group objects that can be used when encrypting documents or data using either [encryptForRecipients](#) or [addRecipientListCryptFilter](#) methods of the Doc Object.

**NOTES:** Can be executed only during console, menu, or application initialization events.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

Not available in Reader.

**Parameters**


---

<b>oOptions</b>	A <a href="#">DisplayOptions Generic Object</a> containing the parameters for the display options.
-----------------	--

---

**Returns**

An array of generic [Group Objects](#).

See `doc.encryptForRecipients` for a description of the generic [Group Object](#).

**DisplayOptions Generic Object**

It contains the following properties:

Property	Description
<b>bAllowPermGroups</b>	Controls whether permissions can be set for entries in the recipient list. Default value is <b>true</b> .
<b>bPlaintextMetadata</b>	Controls whether the checkbox is displayed that allows a user to select whether meta data is plaintext or encrypted, and also the default value. If not specified, the checkbox is not shown. If specified, the checkbox is shown and the default value is the value of this property.
<b>cTitle</b>	The title to be displayed in the dialog. The default is "Choose Recipients".
<b>cNote</b>	A note to be displayed in the dialog. The default is to not show any note.
<b>bAllowImportFromFile</b>	Whether the option is displayed that allows a user to import recipients from a file. The default value is <b>true</b> .
<b>bRequireEncryptionCert</b>	If <b>true</b> , recipients will be required to include an encryption certificate. The default value is <b>true</b> .
<b>bRequireEmail</b>	If <b>true</b> , recipients will be required to include an email address. The default value is <b>false</b> .
<b>bUserCert</b>	If <b>true</b> , the user will be prompted to provide his or her own certificate so that he or she can be included in the list of recipients. Setting this flag to <b>true</b> results in a prompt but does not require that the user provide a certificate.

---

**Example 1**

Retrieve groups with permissions

```

var oOptions = {
    bAllowPermGroups: true,
    bPlaintextMetadata: false,
    cTitle: "Encrypt and Email",
    cNote: "Select recipients",
    bAllowImportFromFile: false,
    bRequireEncryptionCert: true,
    bRequireEmail: true
};
var groupArray = security.chooseRecipientsDialog( oOptions );
console.println("Full name = "+ groupArray[0].userEntities[0].fullName);

```

**Example 2**

Get a list of recipients for which to encrypt data and then possibly email the document once done.

```

var oOptions = { bAllowPermGroups: false,
    cNote: "Select the list of recipients. "
    + "Each person must have both an email address and a certificate.",
    bRequireEmail: true,
    bUserCert: true
};
var oGroups = security.chooseRecipientsDialog( oOptions );
// Display the list of recipients in an alert
// Build an email "to" mailList
var numCerts = oGroups[0].userEntities.length;
var cMsg = "The document will be encrypted for the following:\n";
var mailList = new Array;
for( var g=0; g<numCerts; ++g )
{
    var ue = oGroups[0].userEntities[g];
    var oCert = ue.defaultEncryptCert;
    if( oCert == null )
        oCert = ue.certificates[0];
    cMsg += oCert.subjectCN + ", " + ue.email + "\n";
    var oRDN = oCert.subjectDN;
    if( ue.email )
    {
        mailList[g] = ue.email;
    }
    else
        if ( oRDN.e )
        {
            mailList[g] = oRDN.e;
        }
}
var result = app.alert( cMsg );

```

**Example 3**

List all the entries in an array of groups

```

var groups = security.chooseRecipientsDialog( oOptions );
for( g in groups ) {
  console.println( "Group No. " + g );
  // Permissions
  var perms = groups[g].permissions;
  console.println( "Permissions:" );
  for(p in perms) console.println( p + " = " + eval("perms." +p));
  // User Entities
  for( u in groups[i].userEntities ) {
  var user = groups[g].userEntities[u];
  console.println( "User No. " + u );
  for(i in user) console.println( i + " = " + eval("user." +i));
  }
}

```

## chooseSecurityPolicy

7.0		Ⓢ	ⓧ	
-----	--	---	---	--

Opens a dialog that allows a user to choose from a list of security policies, filtered according to the options.

**NOTE:** (Security Ⓢ) Can be executed only during batch, console, menu, or application initialization events. Not available in Reader. This method will display UI.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

### Parameters

<b>oOptions</b>	(optional) A <a href="#">SecurityPolicyOptions Generic Object</a> containing the parameters used for filtering the list of security policies returned. If not specified, all policies found are displayed.
-----------------	--

### Returns

Returns a single [SecurityPolicy Object](#) or **null** if the user aborted selection.

### Example

In this example a policy is chosen and the name is displayed.

```

var options = { cHandler: security.APSHandler };
var policy = security.chooseSecurityPolicy( options );
console.println("The policy chosen was: " + policy.name);

```

**security.APSHandler** is one of the [Security Constants](#).



## exportToFile

6.0		Ⓢ	ⓧ	
-----	--	---	---	--

Exports a [Certificate Object](#) to a local disk as a raw certificate file.

**NOTE:** (Security Ⓢ): Data being written must be data for a valid certificate; arbitrary data types cannot be written. This method will not overwrite an existing file.

See also `security.importFromFile`.

### Parameters

<code>oObject</code>	The <a href="#">Certificate Object</a> that is to be exported to disk.
<code>cDIPath</code>	The device-independent save path. <b>NOTE:</b> (Security Ⓢ): The parameter <code>cDIPath</code> must be <a href="#">Safe Path</a> and must end with the extension <code>.cer</code> .

### Returns

The path of the file that was written, if successful.

### Example

```
var outPath = security.exportToFile(oCert, "/c/outCert.cer");
```

## getHandler

5.0		Ⓢ		
-----	--	---	--	--

Obtains a [SecurityHandler Object](#). The caller can create as many new engines as desired and each call to `getHandler` creates a new engine; however, there is only one UI engine.

**NOTE:** (Security Ⓢ): This method is available from batch, console, app initialization and menu events. It is also available in the Adobe Reader.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

**Backward Compatibility Note:** As *Adobe.PPKMS* is no longer available as a separate handle starting Acrobat 7.0, invoking `getHandler` with `cName` as "Adobe.PPKMS" returns the engine associated with *Adobe.PPKLite* handler.

**Parameters**

<b>cName</b>	The language independent name of the security handler, as returned by the <a href="#">handlers</a> property. (version 7.0) Beginning with Acrobat 7, there are constant strings defined on the security object for each of the valid handlers. See <a href="#">Security Constants</a> , in particular, see the <a href="#">HandlerName</a> object.
<b>bUIEngine</b>	(optional) If <b>true</b> , the method returns the existing security handler instance that is associated with the Acrobat user interface (so that, for example, a user can log in via the user interface). If <b>false</b> (the default), returns a new engine.

**Returns**

The [SecurityHandler Object](#) specified by **cName**. If the handler is not present, returns a **null** object.

**Example**

This code selects the *Adobe.PPKLite SecurityHandler*.

```
// validate signatures on open
security.validateSignaturesOnOpen = true;

// list all available signature handlers
var a = security.handlers;
for (var i = 0; i < a.length; i++)
    console.println("a["+i+"] = "+a[i]);

// use "Adobe.PPKLite" handler engine for the UI
var ppklite = security.getHandler(
    security.PPKLiteHandler, true);
// login
ppklite.login("dps017", "/C/profiles/DPSmith.pfx");
```


See also the example following [signatureSign](#) for a continuation of this example.

**getSecurityPolicies**

7.0		Ⓢ	ⓧ	
-----	--	---	---	--

Returns the list of security policies currently available, filtered according to the options specified. The master list of security policies will be updated prior to filtering. The default security handler objects are used to retrieve the latest policies. If no policies are available or none meet the filtering restrictions, **null** will be returned.

**NOTES:** You may be able to retrieve more policies by calling `login()` on the default security handler objects before calling this function.

(Security ) Can be executed only during console or application initialization events. Not available in Reader.

## Parameters

<b>bUI</b>	(optional) A flag controlling whether UI can be displayed. Default value is <b>false</b> .
<b>oOptions</b>	(optional) A <a href="#">SecurityPolicyOptions Generic Object</a> containing the parameters used for filtering the list of security policies returned. If not specified, all policies found are returned.

## Returns

An array of [SecurityPolicy Objects](#) or **null**.

## SecurityPolicyOptions Generic Object

The SecurityPolicyOptions object has the following properties:

Property	Description
<b>bFavorites</b>	If not passed, policies are not filtered based on whether a policy is a Favorite. If <b>true</b> , only policies which are Favorites are returned. If <b>false</b> , only policies which are <i>not</i> Favorites are returned.
<b>cHandler</b>	If not passed, policies are not filtered based on security filter. If defined, only policies which match the specified security filter are returned. The valid values are defined in <a href="#">Security Constants</a> , see the <a href="#">HandlerName</a> object. Only a single value can be passed.
<b>cTarget</b>	If not passed, policies are not filtered based on target. If defined, only policies which match the specified target are returned. The valid values are defined in <a href="#">Security Constants</a> , see the <a href="#">EncryptTarget</a> object. Only a single value can be passed.

## Example 1

In this example the list of favorite PPKLite policies is retrieved and the names are displayed. This example uses `security.PPKLiteHandler`, see [Security Constants](#).

```
var options = { bFavorites:true, cHandler:security.PPKLiteHandler };
var policyArray = security.getSecurityPolicies( { oOptions: options } );
for( var i = 0; i < policyArray.length; i++)
    console.println( policyArray[i].name );
```

## Example 2

In this example the login is forced, the list of APS policies is retrieved, and the names are displayed. This example uses `security.APSHandler`, see [Security Constants](#).

```

var aps = security.getHandler( security.APSHandler, true );
aps.login();
var options = { cHandler: security.APSHandler };
var policyArray = security.getSecurityPolicies({
    bUI: true,
    oOptions: options
});
for(var i = 0; i < policyArray.length; i++)
    console.println( policyArray[i].name );

```

## importFromFile

6.0		Ⓢ	ⓧ	
-----	--	---	---	--

Reads a raw data file and returns the data as an object with a type specified by **cType**. The file being imported must be a valid certificate.

Related method, [security.exportToFile](#)

### Parameters

<b>cType</b>	The type of object to be returned by this method. The only supported type is "Certificate".
<b>cDIPath</b>	(optional) When <b>bUI</b> is <b>false</b> , this parameter is a required and specifies the device-independent path to the file to be opened. If <b>bUI</b> is <b>true</b> , this is the seed path used in the open dialog.
<b>bUI</b>	(optional) <b>true</b> if the user should be prompted to select the file that is to be imported. The default is <b>false</b> .
<b>cMsg</b>	(optional) If <b>bUI</b> is <b>true</b> , the title to use in the open dialog. If <b>cMsg</b> is not specified, the default title is used for the dialog.

### Returns

A [Certificate Object](#).

### Example

```
var oMyCert = security.importFromFile("Certificate", "/c/myCert.cer");
```

## SecurityPolicy Object

7.0		Ⓢ	ⓧ	
-----	--	---	---	--

The Security Policy object represents a group of security settings used to apply encryption to a document. It can be acquired as the return value of both [getSecurityPolicies](#) and [chooseSecurityPolicy](#).

### SecurityPolicy Properties

Property	Type	Access	Description
<b>policyID</b>	String	R	This is a generated string used to uniquely identify the Security Policy. It is not intended to be human readable. This may be set to a known policyId on a newly created SecurityPolicy object to force any method using this policy to retrieve the correct security settings before applying the policy.
<b>name</b>	String	R	This is the policy name used for UI. It may be localized.
<b>description</b>	String	R	This is the policy description used for UI. It may be localized.
<b>handler</b>	String	R	This is an enumerated value representing the security handler implementing this Security Policy. The possible values are defined as <a href="#">Security Constants</a> in the <a href="#">Security Object</a> , in particular, see the <a href="#">HandlerName</a> object.
<b>target</b>	String	R	This is an enumeration value representing the target data to be encrypted. The possible values are defined as <a href="#">Security Constants</a> in the <a href="#">Security Object</a> with a prefix of "EncryptTarget". See the <a href="#">EncryptTarget</a> object.

### SecurityHandler Object

SecurityHandler objects are used to access security handler capabilities such as signatures, encryption and directories. Different security handlers will have different properties and methods. This section documents the full set of properties and methods that security

objects may have. Individual SecurityHandler objects may or may not implement these properties and methods.

SecurityHandler objects can be obtained using the `security.getHandler` method.

The JavaScript interface for *Adobe.PPKLite* signatures was introduced in Acrobat 5.0, with the remainder of the JavaScript interface being introduced in Acrobat 6.0. Prior to Acrobat 6.0 there was no support in Acrobat to enable JavaScript in third party security handlers.

Not all security handlers are JavaScript enabled. Not all JavaScript enabled handlers are enabled for all security operations. Third party public key security handlers may support JavaScript, but only if they use the new *PubSec* programming interface that was introduced in Acrobat 6.0.

JavaScript enabled handlers provided by Adobe include:

- the *Adobe.PPKLite* security handler, supporting signature and encryption, on Windows operating system providing directory access through the Microsoft Active Directory Scripting Interface (ADSI); and
- the *Adobe.AAB* security handler providing a local address book and support for directory operations.

Note that the Standard security handler, used for password encryption of documents, is not JavaScript enabled in general, however starting with Acrobat 7.0 encryption using Standard security is possible using predefined policies. See `encryptUsingPolicy` for more details.

Also starting with Acrobat 7.0, the *Adobe.APS* handler can be used for encryption via the `encryptUsingPolicy` method. This handler will also make a directory available via the directory services, but as no certificates will be returned from this directory it is of limited general use.

**NOTES:** (Security ©): **SecurityHandler** Objects can only be created using the Security Object `getHandler` method. This method is available only for batch, console, application initialization and menu events, and is available in the Adobe Reader.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

## SecurityHandler Properties

### appearances

5.0		©		
-----	--	---	--	--

An array containing the language-dependent names of the available user-configured appearances for the specified security handler. Appearances are used to create the on-page visual representation of a signature when signing a signature field. The name of an

appearance can be specified as a signature info object property when signing a signature field using `field.signatureSign`.

Acrobat provides a standard signature appearance module that is used by Adobe signature plug-ins and that can also be used by third party signature plug-ins. This standard signature appearance module is pre-configured with one appearance and can be configured by users to contain more appearances. The name of the one pre-configured appearance, called *Standard Text* in the user interface, is not returned by this property.

If a security handler does not support selection of appearances then this property will return null.

Type: Array

Access: R.

## digitalIDs

6.0		Ⓢ		
-----	--	---	--	--

This method returns the certificates that are associated with the currently selected Digital IDs for this security handler.

Type: Object

Access: R.

The return value is a generic object with the following properties:

Property	Type	Version	Description
<code>oEndUserSignCert</code>	Certificate Object	6.0	The certificate that is associated with the currently selected Digital IDs that is to be used by this security handler object when signing. The property is undefined if there is no current selection.
<code>oEndUserCryptCert</code>	Certificate Object	6.0	The certificate that is associated with the currently selected Digital IDs that is to be used when encrypting a document with this security handler object. The property is undefined if there is no current selection.
<code>certs</code>	Array of Certificate Objects	6.0	An array of certificates corresponding to the list of all Digital IDs that are available for this security handler object.
<code>stores</code>	Array of String identifying the Digital ID store	6.0	An array of strings, one for every Certificate Object, identifying the store where the Digital ID is stored. The string values are up to the security handler. For <i>Adobe.PPKLite</i> the valid values are 'PKCS12' and 'MSCAPI'.

The *Adobe.PPKLite* security handler returns all currently available Digital IDs, present in any of the following stores: Password-protected Digital ID files (both PKCS#12 and APF) and, on Windows OS, IDs present in the Windows (MCCAPI) store.

Both `oEndUserSignCert` and `oEndUserCryptCert` properties can be set using the user-interface. `oEndUserSignCert` can also be set using the `login` method. This means that `oEndUserCryptCert` will only be returned when using a Security Handler object that is obtained using the `getHandler` method with `bUIEngine` set to `true`.

### Example

```
var sh = security.getHandler( "Adobe.PPKMS", true );
var ids = sh.digitalIDs;
var oCert = ids.oEndUserSignCert;
security.exportToFile( oCert, "/c/MySigningCert.cer" );
```

## directories

6.0		Ⓒ		
-----	--	---	--	--

Returns an array of the available [Directory Objects](#) for this Security Handler. New [Directory Objects](#) can be created using the `newDirectory` method.

Type: Array

Access: R.

## directoryHandlers

6.0		Ⓒ		
-----	--	---	--	--

Returns an array containing the language independent names of the available directory handlers for the specified security handler. As an example, the *Adobe.PPKMS* security handler has a directory handler named *Adobe.PPKMS.ADSI* that supports queries using the *Microsoft Active Directory Script Interface* (ADSI). Valid directory handler names are required when activating a new [Directory Object](#) using its `info` property.

Type: Array

Access: R.

## isLoggedIn

5.0		Ⓒ		
-----	--	---	--	--

Returns `true` if currently logged into this [SecurityHandler Object](#). See the `login` method.

Different security handlers will have their own rules for determining the value of this property. The *Adobe.PPKLite* handler will return `true` if a user is logged in to a profile file



(also called credential file, implemented as a PKCS#12 file). *Adobe.PPKMS* will always return **true**.

*Type: Boolean*

*Access: R.*

### Example

```
var ppklite = security.getHandler("Adobe.PPKLite", true);
console.println( "Is logged in = " + ppklite.isLoggedIn ); // false
ppklite.login( "dps017", "/C/signatures/DPSmith.pfx" );
console.println( "Is logged in = " + ppklite.isLoggedIn ); // true
```

## loginName

5.0		Ⓢ		
-----	--	---	--	--

The name associated with the actively selected signing Digital ID for the security handler. This may require that the [login](#) method be called in order to select a signing credential. The return value is **null** if a signing credential is not selected or if the security handler does not support this property.

*Type: String*

*Access: R.*

## loginPath

5.0		Ⓢ		
-----	--	---	--	--

The device-independent path to the user's profile file used to login to the security handler. The return value is null if no one is logged in, if the security handler does not support this property, or if this property is irrelevant for the currently logged in user.

*Type: String*

*Access: R.*

## name

5.0		Ⓢ		
-----	--	---	--	--

The language-independent name of the security handler. Example values for the Default Certificate, Windows Certificate, and Entrust Security Handlers are *Adobe.PPKLite*, *Adobe.PPKMS*, and *Entrust.PPKEF*. All security handlers must support this property.

*Type: String*

*Access: R.*

**signAuthor**

6.0		Ⓢ		
-----	--	---	--	--

Whether the security handler is capable of generating certified documents. A certified document is a document that is signed with both a byte range signature and an object signature. Object signatures are generated by walking the object tree of the document and are used to detect and prevent modifications to a document. Refer to the `mdp` property of the [SignatureInfo Object](#) for details regarding modification detection and prevention (MDP) settings.

*Type: Boolean*

*Access: R.*

**signFDF**

6.0		Ⓢ		
-----	--	---	--	--

Indicates that the security handler is capable of signing FDF files.

*Type: Boolean*

*Access: R.*

**signInvisible**

5.0		Ⓢ		
-----	--	---	--	--

Whether the security handler is capable of generating invisible signatures.

*Type: Boolean*

*Access: R.*

**signValidate**

6.0		Ⓢ		
-----	--	---	--	--

Indicates whether the security handler is capable of validating signatures.

*Type: Boolean*

*Access: R.*

**signVisible**

5.0		Ⓢ		
-----	--	---	--	--

Whether the security handler is capable of generating visible signatures.

*Type: Boolean**Access: R.***uiName**

5.0		Ⓢ		
-----	--	---	--	--

The language-dependent string for the security handler. This string is suitable for user interfaces. All security handlers must support this property.

*Type: String**Access: R.*

---

**SecurityHandler Methods****login**

5.0		Ⓢ		
-----	--	---	--	--

This method provides a mechanism by which Digital IDs can be accessed and selected for a particular Security Handler. Through user-interface, a 'Default DigitalID' can be selected which is either eternally persistent or persistent for as long as the application is running. If such a selection has been made through the UI, then it might not be necessary to log into a Security Handler prior to using the DigitalID.

Parameters tend to be specific to a particular handler. The behaviour for *Adobe.PPKLite* and *Adobe.PPKMS* handlers is specified below.

The parameters **cPassword** and **cDIPath** are provided for backward compatibility, or they can be included as properties of the **oParams** object. This latter method is the preferred calling convention beginning in Acrobat 6.0.

See also [logout](#), [newUser](#), and [loginName](#).

**Parameters**

<b>cPassword</b>	(optional, version 5.0) The password necessary to access the password-protected Digital ID. This parameter is supported by <i>Adobe.PPKLite</i> for accessing Digital ID files and PKCS#11 devices.
<b>cDIPath</b>	(optional, version 5.0) A device independent path to the password-protected Digital ID file or a PKCS#11 library. This parameter is supported by <i>Adobe.PPKLite</i> .

---

<b>oParams</b>	(optional, version 6.0) A <a href="#">LoginParameters Generic Object</a> with parameters that are specific to a particular <a href="#">SecurityHandler Object</a> . The common fields in this object are described below. These fields include the <b>cDIPath</b> and <b>cPassword</b> values, thus allowing the parameter list to be expressed in different ways.
<b>bUI</b>	(optional, version 6.0) Set to <b>true</b> if it is desired that user interface be used to log the user in. This attribute should be supported by all security handlers that support this method.

---

**Returns**

Returns **true** if the login succeeded, **false** otherwise.

**LoginParameters Generic Object**

This generic JS object contains parameters for the [login](#) method. It has the following properties:

---

Property	Type	Version	Description
<b>cDIPath</b>	String	5.0	The path to a file that contains the Digital ID or a PKCS#11 library. Supported by <i>Adobe.PPKLite</i> security handler.
<b>cPassword</b>	String	6.0	A password that is used to authenticate the user. This password may be used to access a password-protected Digital ID file or a PKCS#11 device. Supported by <i>Adobe.PPKLite</i> security handler. Note that Acrobat does not guarantee that this password is obfuscated in memory.
<b>cPFX</b>	String	7.0	(optional) The entire PFX (hex encoded) to log into. If this parameter is specified, it takes precedence over <b>cDIPath</b> . Note, currently there's no way to retrieve a hex encoded PFX file through JS. This is only used internally.

---

Property	Type	Version	Description
<b>oEndUserSignCert</b>	generic object	6.0	Selects a Digital ID for the purpose of performing end user signing. The value of this property is a <a href="#">Certificate Object</a> , or generic object with the same property names as a <a href="#">Certificate Object</a> , defining the certificate that is being selected. It may or may not be necessary to call this method for a particular handler. For example, if logged in to a PKCS#12 file containing one signing Digital ID with <i>Adobe.PPKLite</i> , a signing credential will not need to be selected. All security handlers must be able to process the binary and SHA1Hash properties of this object. This object can be empty if <b>bUI</b> is <b>true</b> .
<b>cMsg</b>	String	6.0	A message to display in the login dialog, if <b>bUI</b> is <b>true</b> .
<b>cURI</b>	String	7.0	URI used to connect to a server. Only supported by the <i>Adobe.APS</i> handler.
<b>cUserId</b>	String	7.0	User name used when connecting to a server. Only supported by the <i>Adobe.APS</i> handler.
<b>cDomain</b>	String	7.0	Domain name used when connecting to a server. Only supported by the <i>Adobe.APS</i> handler.
<b>iSlotID</b>	Integer	7.0	Specifies the slot ID of a PKCS#11 device to log into. This parameter is supported by the <i>Adobe.PPKLite</i> handler only.
<b>cTokenLabel</b>	String	7.0	Specifies the token label of a PKCS#11 device to log into. This parameter is supported by the <i>Adobe.PPKLite</i> handler only.

**Example 1**

```
// Use "Adobe.PPKLite" Security Handler Object for the UI
var ppklite = security.getHandler( security.PPKLiteHandler, true );
var oParams = { cPassword: "dps017", cDIPath: "/C/DPSmith.pfx" }
ppklite.login( oParams );
<..... make a signature field and sign it .....>
ppklite.logout();
```

```

// PPKLite - Use UI to select a credential, when already logged in
ppklite.login(
  { oParams:
    { oEndUserSignCert: {},
      cMsg: "Select your Digital ID" },
      bUI : true
    } );

// PPKLite - Login and select signing credential
var oCert = { SHA1Hash: "00000000" };
ppklite.login(
  { oParams:
    { cDIPath: "/C/test/DPSmith.pfx",
      cPassword: "dps017",
      oEndUserSignCert: oCert,
      cMsg: "Select your Digital ID"
    },
      bUI : true
    } );

```

**Example 2**

```

// Use "Adobe.PPKMS" Security Handler Object
var ppkms = security.getHandler( "Adobe.PPKMS" );

// Select credential to use when signing
var oCert = myCerts[0];
ppkms.login( { oParams: { oEndUserSignCert: oCert } } );

```

**Example 3**

Use *Adobe.APS* Security Handler Object. This example uses **security.APSHandler**, see [Security Constants](#).

```

var aps = security.getHandler( security.APSHandler, true );
var oParams = { cUserName: "acrobat", cPassword: "adobedon" };
aps.login( oParams );
<..... encrypt a document using this handle and a policy id .....>
aps.logout ();

```

See [signatureSign](#) for details on signing a PDF document.

**logout**

5.0		©		
-----	--	---	--	--

Logs out for the [SecurityHandler Object](#). This method is used by *Adobe.PPKLite*, not by *Adobe.PPKMS*.

Also see the [login](#) method.

**Parameters**

None

**Returns**

Beginning in Acrobat 6.0, returns **true** if the logout succeeded, **false** otherwise. Previous Acrobat releases did not generate a return value.

**newDirectory**

6.0				
-----	--	--	--	--

Returns a new [Directory Object](#). The directory object must be activated using its [info](#) property before it is marked for persistence and can be used for searches. Existing directory objects can be discovered using the [directories](#) property.

**Parameters**

None

**Returns**

Returns a new [Directory Object](#)

**newUser**

5.0				
-----	--	--	--	--

This method supports enrollment with *Adobe.PPKLite* and *Adobe.PPKMS* security handlers by creating a new self-sign credential.

**NOTE:** (Security ): This method will not allow the user to overwrite an existing file.

**Parameters**

<b>cPassword</b>	(optional) The password necessary to access the password-protected Digital ID file. This parameter is ignored by <i>Adobe.PPKMS</i> .
<b>cDIPath</b>	(optional) The device-independent path to the password-protected Digital ID file. This parameter is ignored by <i>Adobe.PPKMS</i> . <b>NOTE:</b> (Security ): Beginning with Acrobat 6.0, the parameter <b>cDIPath</b> must be <a href="#">Safe Path</a> and end with the extension <code>.pdf</code> .
<b>oRDN</b>	(optional) The relative distinguished name (RDN) as an <a href="#">RDN Generic Object</a> containing the issuer or subject name for a certificate. The only required field is <b>cn</b> . If the country <b>c</b> is provided, it must be two characters, using the ISO 3166 standard (for example, 'US').

<b>oCPS</b>	(optional, version 6.0) A generic object containing certificate policy information that will be embedded in the Certificate Policy extension of the certificate. The object must contain property <b>oid</b> , which indicates the certificate policy object identifier. The other properties which may be present are <b>url</b> and (user) <b>notice</b> . The url is a URL that points to detailed information about the policy under which the certificate has been issued and user <b>notice</b> is a abridged version of the same, embedded in the certificate.
<b>bUI</b>	(optional, version 6.0) When <b>true</b> , the user interface can be used to enroll. This parameter is supported by all security handlers that support this method.
<b>cStore</b>	(optional, version 7.0) A string identifying the store where the generated credential has to be stored. For <i>Adobe.PPKLite</i> security handler, the valid store identifiers are "PKCS12" and "MSCAPI". If this parameter is omitted and <b>cDIPath</b> is provided, the generated credential is stored in a PKCS#12 file, else it is stored in the CAPI store.

**Returns**

**true** if successful, throws an exception if not successful.

**Example**

```
// Create a new PPKLite self-sign credential (Acrobat 5.0 syntax)
var ppklite = security.getHandler(security.PPKLiteHandler);
var oRDN = { cn: "Fred NewUser", c: "US" };
var oCPS = {oid: "1.2.3.4.5",
            url: "http://www.myca.com/mycps.html",
            notice: "This is a self generated certificate, hence the "
                  + "recipient must verify it's authenticity through an out "
                  + "of band mechansism" };
ppklite.newUser( "testtest", "/d/temp/FredNewUser.pfx", oRDN, oCPS);

// Alternate generic object syntax, allowing additional parameters
var oParams = {
    cPassword : "myPassword",
    cDIPath : "/d/temp/FredNewUser.pfx",
    oRDN : oRDN,
    oCPS : oCPS,
    bUI : false
};
ppklite.newUser( oParams );

// Use a certificate from an existing signed, field to create the RDN
var f = this.getField( "mySignature" );
f.signatureValidate();
var sigInfo = f.signatureInfo();
var certs = sigInfo.certificates;
var oSubjectDN = certs[0].subjectDN;
```



```
ppk-lite.newUser({
  cPassword: "dps017",
  cDIPath: "/c/temp/DPSmith.pfx",
  oRDN: oSubjectDN
});
```

## setPasswordTimeout

5.0		Ⓢ	ⓧ	
-----	--	---	---	--

Sets the number of seconds after which password should expire between signatures. This method is only supported by the *Adobe.PPKLite* security handler. For this handler the default timeout value for a new user is 0 (password always required).

### Parameters

<b>cPassword</b>	The password needed to set the timeout value.
<b>iTimeout</b>	The timeout value, in seconds. Set to 0 for always expire (that is, password always required). Set to 0x7FFFFFFF for never expire.

### Returns

Throws an exception if the user has not logged in to the *Adobe.PPKLite* Security Handler, or unsuccessful for any other reason.

### Example

This example logs in to the *PPKLite* security handler and sets the password timeout to 30 seconds. If the password timeout has expired—30 seconds in this example—the signer must provide a password. The password is not necessary if the password has not timed out.

```
var ppk-lite= security.getHandler( "Adobe.PPKLite" );
ppk-lite.login( "dps017", "/d/profiles/DPSmith.pfx" );
ppk-lite.setPasswordTimeout( "dps017", 30 );
```

## SignatureInfo Object

A generic JS object that contains the properties of a digital signature. Some properties are supported by all handlers, and additional properties can be supported.

The *SignatureInfo* object is returned by the field methods **field.signatureValidate** and **field.signatureInfo**, and is passed to the methods **FDF.signatureSign**, and **FDF.signatureValidate**.

Writable properties can be specified when signing the object, see the **field.signatureInfo** and **FDF.signatureSign**.

## SignatureInfo Object properties

All handlers define the following properties:

### SignatureInfo Object properties

Property	Type	Access	Version	Description
<b>buildInfo</b>	Object	R	6.0	An object containing software build and version information for the signature. The format of this object is not described in this document. An Acrobat technote may be produced that contains this information. The subject of this technote will be signature build properties dictionary.
<b>date</b>	Date Object	R	5.0	The date and time that the signature was created, returned as a JS date object.
<b>dateTrusted</b>	Boolean	R	7.0	This boolean indicates if the "date" is from a trusted source or not. If this value is not present, the date should be assumed from an untrusted source (e.g. signer's computer system time).
<b>handlerName</b>	String	R	5.0	The language independent name of the security handler that was specified as the Filter attribute in the signature dictionary. This is usually the name of the security handler that created the signature, but can also be the name of the security handler that the creator desires to be used when validating the signature.
<b>handlerUserName</b>	String	R	5.0	The language dependent name corresponding to security handler specified by <b>handlerName</b> . This is only available when the named security handler is available.
<b>handlerUIName</b>	String	R	5.0	The language dependent name corresponding to security handler specified by <b>handlerName</b> . This is only available when the named security handler is available.

SignatureInfo Object properties				
Property	Type	Access	Version	Description
<code>location</code>	String	R/W	5.0	Optional user specified location when signing. This can be a physical location (such as a city) or hostname.
<code>mdp</code>	String	R/W	6.0	The Modification Detection and Prevention (MDP) setting that was used to sign the field or <a href="#">FDF Object</a> being read, or the MDP setting to use when signing. Values are: <pre>allowNone allowAll default defaultAndComments</pre> See <a href="#">Modification Detection and Prevention (MDP) Values</a> for details. The value of <code>allowAll</code> , the default, means that MDP is not used for the signature, resulting in this not being an author signature.
<code>name</code>	String	R	5.0	Name of the user that created the signature.
<code>numFieldsAltered</code>	Number	R	5.0 only	Deprecated. The number of fields altered between the previous signature and this signature. Used only for signature fields. Beginning in Acrobat 7.0 the functionality offered by <a href="#">signatureGetModifications</a> should be used instead.
<code>numFieldsFilledIn</code>	Number	R	5.0 only	Deprecated. The number of fields filled-in between the previous signature and this signature. Used only for signature fields. Beginning in Acrobat 7.0 the functionality offered by <a href="#">signatureGetModifications</a> should be used instead.

SignatureInfo Object properties				
Property	Type	Access	Version	Description
<b>numPagesAltered</b>	Number	R	5.0 only	Deprecated. The number of pages altered between the previous signature and this signature. Used only for signature fields. Beginning in Acrobat 7.0 the functionality offered by <a href="#">signatureGetModifications</a> should be used instead.
<b>numRevisions</b>	Number	R	5.0	The number of revisions in the document. Used only for signature fields.
<b>reason</b>	String	R/W	5.0	User specified reason for signing.
<b>revision</b>	Number	R	5.0	The signature revision to which this signature field corresponds. Used only for signature fields.
<b>sigValue</b>	String	R	7.0	Raw bytes of the signature, as a hex encoded string.
<b>status</b>	Number	R	5.0	The validity status of the signature, computed during the last call to the <a href="#">signatureValidate</a> . See the return codes of the status property in the table “ <a href="#">status and idValidity Properties</a> ” on page 568.
<b>statusText</b>	String	R	5.0	The language dependent text string, suitable for user display, denoting the signature validity status, computed during the last call to the <a href="#">signatureValidate</a> .
<b>subFilter</b>	String	R/W	6.0	The format to use when signing. Consult the PDF Reference for a complete list of supported values. The known values used for public key signatures include adbe.pkcs7.sha1, adbe.pkcs7.detached, and adbe.x509.rsa_sha1. It is important that the caller know that a particular signature handler can support this format.

SignatureInfo Object properties				
Property	Type	Access	Version	Description
<b>timeStamp</b>	String	W	7.0	<p>The timeStamp server URL to use to get the signature timeStamped. The only schemes/transport protocols supported for fetching timeStamps are http(s).</p> <p>This is a write only property. If the signature is timestamped, during verification the property "dateTrusted" will be set to <b>true</b> (provided the timestamp signature is trusted) and the <b>verifyDate</b> and the signing date will be the same.</p>
<b>verifyDate</b>	Date Object	R	7.0	The date and time that the signature was verified (if the signature has been verified), returned as a JS date object.
<b>verifyHandlerName</b>	String	R	6.0	The language independent name of the security handler that was used to validate this signature. This will be <b>null</b> if the signature has not been validated, that is, if the status property has a value of 1
<b>verifyHandlerUIName</b>	String	R	6.0	The language dependent name corresponding to security handler specified by verifyHandlerName. This will be null if the signature has not been validated, that is, if the status property has a value of 1.

**SignatureInfo Object Public Key Security Handler Properties**

Public key security handlers may define the following additional properties:

**SignatureInfo Object Public Key Security Handler Properties**

Property	Type	Access	Version	Description
<b>appearance</b>	String	W	5.0	The name of the user-configured appearance to use when signing this field. PPKLite and PPKMS use the standard appearance handler, and in this situation, the appearance names can be found in the signature appearance configuration dialog of the user interface (menu Edit > Preferences > Digital Signatures in Acrobat 6.0). The default, when not specified, is to use the Standard Text appearance. Used only for visible signature fields.
<b>certificates</b>	Array	R	5.0	Array containing a hierarchy of certificates that identify the signer. The first element in the array is the signer's certificate, and subsequent elements include the chain of certificates up to the certificate authority that issued the signer's certificate. For self-signed certificates this array will contain only one entry.
<b>contactInfo</b>	String	R/W	5.0	User specified contact information for determining trust. For example, a telephone number that recipients of a document can use to contact the author to establish trust. This is not recommended for a scalable solution for establishing trust.
<b>byteRange</b>	Array	R	6.0	An array of numbers indicating the bytes that are covered by this signature.
<b>docValidity</b>	Number	R	6.0	The validity status of the document byte range digest portion of the signature, computed during the last call to <a href="#">signatureValidate</a> . All PDF document signature field signatures include a byte range digest. See <a href="#">Validity Values</a> for details of the return codes.

**SignatureInfo Object Public Key Security Handler Properties**

Property	Type	Access	Version	Description
<b>idPrivValidity</b>	Number	R	6.0	Returns the validity of the identity of the signer. This value is specific to the handler. See <a href="#">Private Validity Values</a> for values supported by the <i>Adobe.PPKLite</i> and <i>Adobe.PPKMS</i> handlers. This value is 0 unless the signature has been validated, that is, if the <b>status</b> property has a value of 1.
<b>idValidity</b>	Number	R	6.0	Returns the validity of the identity of the signer as number. See the return codes of the <b>idValidity</b> property in the table " <a href="#">status and idValidity Properties</a> " on page 568
<b>objValidity</b>	Number	R	6.0	The validity status of the object digest portion of the signature, computed during the last call to <a href="#">signatureValidate</a> . For PDF documents, signature field author signatures and document level application rights signatures include object digests. All FDF files are signed using object digests. See <a href="#">Validity Values</a> for details of the return codes.
<b>revInfo</b>	Object	R	7.0	A generic object containing two properties; "CRL" and "OCSP". Both of these properties are array of strings, where each string contains raw bytes of the revocation information that was used to carry out revocation checking of a certificate. The strings are hex encoded. For "CRL" the string represents a CRL whereas for "OCSP" the string represents an OCSP response. These properties are populated only if the application preference to populate them is turned on, as this data can potentially get very large.

---

**SignatureInfo Object Public Key Security Handler Properties**

Property	Type	Access	Version	Description
<b>trustFlags</b>	Number	R	6.0	The bits in this number indicate what the signer is trusted for. The value is valid only when the value of the status property is 4. These trust settings are derived from trust setting in the recipient's trust database, for example the Acrobat Address Book (Adobe.AAB). Bit assignments are: 1- trusted for signatures 2- trusted for certifying documents 3- trusted for dynamic content such as multimedia 4- Adobe internal use 5- the javascript in the PDF file is trusted to operate outside the normal PDF restrictions
<b>password</b>	String	W	5.0	Password required as authentication when accessing a private key that is to be used for signing. This may or may not be required, dependent on the policies of the security handler.

---

**status and idValidity Properties**

The following table list the codes returned by the Signature Info Object, **status** and **idValidity** properties.

Status Code	Description
-1	Not a signature field.
0	Signature is blank or unsigned.
1	Unknown status. This occurs if the signature has not yet been validated. This can occur if the document has not completed downloading over a network connection.
2	Signature is invalid.
3	Signature of document is valid, identity of signer could not be verified.
4	Signature of document is valid and identity of signer is valid.

---



## Validity Values

The following codes are returned by the `docValidity` and `objValidity` (See [SignatureInfo Object Public Key Security Handler Properties](#)), allowing a finer granularity of understanding of the validity of the signature than the `status` property.

Validity Values	
Status Code	Description
<code>kDSSigValUnknown</code>	Validity not yet determined.
<code>kDSSigValUnknownTrouble</code>	Validity could not be determined because of errors encountered during the validation process.
<code>kDSSigValUnknownBytesNotReady</code>	Validity could not be determined because all bytes are not available, for example when viewing a file in a web browser. Even when bytes are not immediately available, this value may not be returned if the underlying implementation blocks when bytes are not ready. Adobe makes no commitment regarding whether validation checks will block or not block, however the implementation in Acrobat 6.0 will block when validating <code>docValidity</code> and not block when validating <code>objValidity</code> .
<code>kDSSigValInvalidTrouble</code>	Validity for this digest was not computed because there were errors in the formatting or information contained in this signature. There is sufficient evidence to conclude that the signature is invalid.
<code>kDSSigValInvalidTrouble</code>	Validity for this digest is not used (e.g., no doc validity if no byte range).
<code>kDSSigValJustSigned</code>	The signature was just signed, so implicitly valid.
<code>kDSSigValFalse</code>	The digest or validity is invalid
<code>kDSSigValTrue</code>	The digest or validity is valid

## Private Validity Values

Verification of the validity of the signer's identity is specific to the handler that is being used to validate the identity. This value may contain useful information regarding an identity. The identity is returned in the `idPrivValidity` property. Values for *Adobe.PPKMS* and

Adobe.PPKLite security handlers are shown here. This value is also mapped to an **idValidity** value that is common across all handlers.

<b>Private Validity Values</b>			
<b>Status Code</b>	<b>idValidity Mapping</b>	<b>Security Handler</b>	<b>Description</b>
<b>kIdUnknown</b>	1 (unknown)	PPKMS, PPKLite	Validity not yet determined.
<b>kIdTrouble</b>	1 (unknown)	PPKMS, PPKLite	Could not determine validity because of errors, for example internal errors, or could not build the chain, or could not check basic policy.
<b>kIdInvalid</b>	2 (invalid)	PPKMS, PPKLite	Certificate is invalid: not time nested, invalid signature, invalid/unsupported constraints, invalid extensions, chain is cyclic.
<b>kIdNotTimeValid</b>	2 (invalid)	PPKMS, PPKLite	Certificate is outside its time window (too early, too late).
<b>kIdRevoked</b>	2 (invalid)	PPKMS	Certificate has been revoked.
<b>kIdUntrustedRoot</b>	1 (unknown)	PPKMS, PPKLite	Certificate has an untrusted root certificate.
<b>kIdBrokenChain</b>	2 (invalid)	PPKMS, PPKLite	Could not build a certificate chain up to a self-signed root certificate.
<b>kIdPathLenConstraint</b>	2 (invalid)	PPKLite	Certificate chain has exceeded the specified length restriction. The restriction was specified in Basic Constraints extension of one of the certificates in the chain.
<b>kIdCriticalExtension</b>	1 (unknown)	PPKMS	One of the certificates in the chain has an unrecognized critical extension.
<b>kIdJustSigned</b>	4 (valid)	PPKMS, PPKLite	Just signed by user (similar to kIdIsSelf)
<b>kIdAssumedValid</b>	3 (idunknown)	PPKMS	Certificate is valid to a trusted root, but revocation could not be checked and <i>was not required</i> .
<b>kIdIsSelf</b>	4 (valid)	PPKMS, PPKLite	Certificate is my credential (no further checking was done).

---

**Private Validity Values**

Status Code	idValidity Mapping	Security Handler	Description
<code>kIdValid</code>	4 (valid)	PPKMS, PPKLite	Certificate is valid to a trusted root (in the Windows or Acrobat Address Book).
<code>kIdRevocationUnknown</code>	?	PPKMS, PPKLite	Certificate is valid to a trusted root, but revocation could not be checked and was <i>required</i> by the user.

---

**Modification Detection and Prevention (MDP) Values**

Modification detection and prevention (MDP) settings control what changes are allowed to occur in a document before the signature becomes invalid. Changes are recorded outside of the byte range, for signature fields, and can include changes that have been incrementally saved as part of the document or changes that have occurred in memory between the time that a document is opened and when the signature is validated. MDP settings may only be applied to the first signature in a document. Use of MDP will result in an author signature. MDP has one of the following four values:

**allowAll:** Allow all changes to a document without any of these changes invalidating the signature. This results in MDP not being used for the signature. This was the behavior for Acrobat 4.0 through 5.1.

**allowNone:** Do not allow any changes to the document without invalidating the signature. Note that this will also lock down the author's signature.

**default:** Allow form field fill in if form fields are present in the document, otherwise do not allow any changes to the document without invalidating the signature.

**defaultAndComments:** Allow form field fill in if form fields are present in the document, and allow annotations (comments) to be added, deleted or modified, otherwise do not allow any changes to the document without invalidating the signature. Note that annotations can be used to obscure portions of a document and thereby affect the visual presentation of the document.

---

**SOAP Object**

The SOAP object allows remote procedure calls to be made to, or sends an XML Message to, a remote server from JavaScript.

The SOAP 1.1 protocol (see <http://www.w3.org/TR/SOAP/>) is used to marshall JavaScript parameters to a remote procedure call (either synchronously or asynchronously) and to unmarshall the result as a JavaScript object. The SOAP object also has the ability to

communicate with Web Services described by the Web Services Description Language (WSDL—see <http://www.w3.org/TR/wsdl>).

**NOTE:** SOAP methods **connect**, **request** and **response** are available only for documents open in Acrobat Professional and Acrobat Standard., and for documents with Form Export Rights(**F**) open in Adobe Reader 6.0 or later.

## SOAP Properties

### wireDump

6.0				
-----	--	--	--	--

If **true**, synchronous SOAP requests will cause the XML Request and Response to be dumped to the JavaScript Console. This is useful for debugging SOAP problems.

Type: Boolean

Access: R/W.

## SOAP Methods

### connect

6.0			<b>F</b>	
-----	--	--	----------	--

Takes the URL of a WSDL document (cURL) and converts it to a JavaScript object with callable methods corresponding to the web service.

The parameters to the method calls and the return values obey the rules specified for the **SOAP.request** method.

#### Parameters

<b>cURL</b>	The URL of a WSDL document. The <b>cURL</b> parameter must be an HTTP or HTTPS URL.
-------------	---

#### Returns

The result value from **SOAP.connect** is a WSDL Service Proxy object with a JavaScript method corresponding to each operation in the WSDL document provided at the URL. The parameters required for the method depend on the WSDL operation you are calling and how the operation encodes its parameters.

- If the WSDL operation is using the SOAP RPC encoding (as described in Section 7 of the SOAP 1.1 Specification) then the arguments to the service method are the same as the parameter order in the WSDL document.

- If the WSDL service is using the SOAP document/literal encoding then the function will have a single argument indicating the request message. The argument may be a JavaScript object literal describing the message or it may be either a string or a ReadStream Object with an XML fragment describing the message. The return value of the service method will correspond to the return value of the WSDL operation.

The JavaScript function objects corresponding to each web service method will use the following properties if they are set. The default is for none of the properties to be set.

Property	Description
<b>asyncHandler</b>	This property indicates that the Web Service method should be performed asynchronously. The property corresponds to the <a href="#">oAsync</a> parameter in <b>SOAP.request</b> .
<b>requestHeader</b>	This property indicates that the Web Service method should include a SOAP Header in the request. The property corresponds to the <a href="#">oReqHeader</a> parameter in <b>SOAP.request</b> .
<b>responseHeader</b>	This property indicates that the Web Service method should return a SOAP Header from the response. The property corresponds to the <a href="#">oRespHeader</a> parameter in <b>SOAP.request</b> .
<b>authenticator</b>	This property indicates how authentication should be handled for the Web Service method. The property corresponds to the <a href="#">oAuthenticate</a> parameter in <b>SOAP.request</b> .

## Exceptions

**SOAPError**, **NetworkError**

See the [Additional Notes on the Exceptions](#).

SOAP Faults will cause a **SOAPError** exception to be thrown. If there is a problem at the networking level, such as an unavailable endpoint, a **NetworkError** will be thrown.

## Example

A service WSDL Document URL is needed. These can be obtained from the "Round 2 Interop Services - using SOAP 1.2" section at the following URL:

<http://www.whitemesa.com/interop.htm>.

```
var cURL = <get a URL for this service from
           http://www.whitemesa.com/interop.htm>;

// Connect to the test service
var service = SOAP.connect(cURL);

// Print out the methods this service supports to the console
for(var i in service) console.println(i);

var cTestString = "This is my test string";

// Call the echoString service -- it is an RPC Encoded method
```

```

var result = service.echoString(cTestString);

// This should be the same as cTestString
console.println(result + " == " + cTestString);

// Call the echoInteger service -- JavaScript doesn't support integers
// so we make our own integer object.
var oTestInt =
{
    soapType: "xsd:int",
    soapValue: "10"
};
var result = service.echoInteger(oTestInt);

// This should be the same as oTestInt.soapValue
console.println(result + " == " + oTestInt.soapValue);

```

This produces the following output:

```

echoBase64
echoBoolean
echoDate
echoDecimal
echoFloat
echoFloatArray
echoHexBinary
echoInteger
echoIntegerArray
echoPolyMorph
echoPolyMorphArray
echoPolyMorphStruct
echoString
echoStringArray
echoStruct
echoStructArray
echoVoid
This is my test string == This is my test string
10 == 10

```

## queryServices

7.0		Ⓢ	
-----	--	---	--

Locate network services that have published themselves using DNS Service Discovery (DNS-SD). This method can locate services that have registered using Multicast DNS (mDNS) for location on a local networking link or through unicast DNS for location within an enterprise. The results of service location are always returned asynchronously and the query continues (with notification as services become available and/or unavailable) until it is stopped.

The result of querying for services is a set of service names which can be bound when needed by calling `resolveService`.

Services can either use a 3rd party mDNS responder to be located in the local network link or register themselves in a DNS server (either statically or dynamically) to be located within an enterprise networking environment.

**Parameters**

<b>cType</b>	The DNS SRV Service Name to search for. See <a href="#">cType</a> for more details.
<b>oAsync</b>	An object that will be called when a service is located. See <a href="#">oAsync</a> for specifications.
<b>aDomains</b>	(optional) An array of domains to make the query in. See <a href="#">aDomains</a> for details.

See the [Standard Acrobat Exceptions](#).

**Returns**

A service query object which manages the duration of the query. The query will continue until one of the following conditions is met:

- the service query object returned from `queryServices ()` is garbage collected.
- the `stop ()` method of the service query object returned from `queryServices ()` is called.

Method	Description
<code>stop</code>	Causes the query to terminate. This method can be called from a notification callback but the operation will not stop until idle processing time.

**Exceptions**

Standard Acrobat Exceptions.

**Additional Notes on the Parameters of `SOAP.queryServices`**

- **cType**  
The `cType` parameter indicates a DNS SRV Service Name that should be queried for. Some possible examples are:
  - `"http"`: Locate Web Servers
  - `"ftp"`: Locate FTP Servers
  - See the DNS SRV Service Name Registry for more examples
- **oAsync**

The **oAsync** object is a notification object which will be notified when services are located on the network or services which had previously been reported are removed. The notification methods will not be called until the **queryServices ()** method returns and are called during idle processing, The **oAsync** parameter should implement the following methods:

Method	Description
<b>addServices</b>	This method is called when available services matching the query are located. The parameter is an array of service description objects (see below) for the services that have been added.
<b>removeServices</b>	This method is called when services (which had previously been introduced by calling the <b>addServices ()</b> notification method) are no longer available. The parameter is an array of service description object (see below) for the services that have been removed.  <b>NOTE:</b> In Acrobat 7.0, only services located through mDNS (i.e. in the "local." domain) are updated dynamically.

The service description object passed to **addServices ()** and **removeServices ()** have the following properties:

Property	Description
<b>name</b>	The unicode display name of the service.
<b>domain</b>	The DNS domain in which the service was located. If the service was located in the local networking link then the domain name will be "local."
<b>type</b>	The DNS SRV Service Name of the service that was located – this will be the same as the <b>cType</b> parameter passed to <b>queryServices ()</b> . This can be useful when the same notification callback is being used for multiple queries.

- **aDomains**

The **aDomains** parameter indicates an array of domains that the query should be made for. The only valid domains are:

Domains	Description
<b>ServiceDiscovery.local</b>	Search for services in the local networking link using Multicast DNS (mDNS). This is useful for finding network services in an <i>ad hoc</i> networking environment but network services will only be located within the scope of the current network router.



Domains	Description
<b>ServiceDiscovery.DNS</b>	Search for services in the default DNS domain using unicast DNS. This is useful for locating network services in the context of a DNS server but typically requires IS assistance to register a service and is less dynamic.

### Example

This example code will produce different output depending on where it is run.

```
var oNotifications =
{
  addServices: function(services)
  {
    for(var i = 0; i < services.length; i++)
      console.println("ADD: " + services[i].name + " in domain "
        + services[i].domain);
  }
  removeServices: function(services)
  {
    for(var i = 0; i < services.length; i++)
      console.println("DEL: " + services[i].name + " in domain "
        + services[i].domain);
  }
};
SOAP.queryServices({
  cType:"http",
  oAsync:oNotifications,
  aDomains:[ServiceDiscovery.local, ServiceDiscovery.DNS]
});
```

The output depends on the current network environment; if there are no services advertised by DNS Service Discovery, the example will produce no output. The following is a representative output:

```
ADD: My Web Server in domain local.
ADD: Joe's Web Server in domain local.
ADD: Example.org Web Server in domain example.org.
```

## resolveService

7.0		Ⓢ	
-----	--	---	--

This method allows a service name to be bound to a network address and port in order for a connection to be made. The connection information is returned asynchronously and should be treated as temporary since the network location of a service may change over time (for example, if a DHCP lease expires or if a service moves to a new server).

**Parameters**

<b>cType</b>	The DNS SRV Service Name to resolve.
<b>cDomain</b>	The domain that the service was located in.
<b>cService</b>	The service name to resolve.
<b>oAsync</b>	An object that will be called when the service is resolved. See <a href="#">oAsync</a> for additional details of this parameter.

**Returns**

A service query object which manages the duration of the resolve. The resolve will continue until one of the following conditions is met:

- the service query object returned from **resolveService ()** is garbage collected.
- the **resolve ()** method of the **oAsync** object is called indicating that the operation completed (either by resolving the service, error or a timeout).
- the **stop ()** method of the service query object returned from **resolveService ()** is called.

Method	Description
<b>stop</b>	Causes the resolve to terminate. This method can be called from a notification callback but the operation will not stop until idle processing time.

**Exceptions**

Standard Acrobat Exceptions.

**Additional Notes on the Parameters of SOAP.resolveService**

- **oAsync**

The **oAsync** object is a notification object which will be called when the service is resolved. The notification methods will not be called until the **resolveService ()** method returns and are called during idle processing. The **oAsync** parameter should implement the following method:

Method	Description
<b>resolve</b>	This method is called with two parameters ( <b>nStatus</b> and <b>oInfo</b> ) when the service is resolved or if it cannot be resolved. The parameter <b>nStatus</b> is the state indicating if the service could be resolved (see below). If the service was successfully resolved then the <b>oInfo</b> object, an instance of the <b>ServiceInfo</b> object (see below), specifies the connection information.

The value of the **nStatus** parameter passed to the resolve method is one of the following:

Value	Description
0	The service was successfully resolved.
1	The service timed out before being resolved. The default timeout in Acrobat 7 is 60 seconds.
-1	There was an networking error trying to resolve the service.

The **ServiceInfo** object passed to the resolve method has the following properties:

Property	Description
<b>target</b>	The IP address or DNS name of the machine supplying the service.
<b>port</b>	The port on the machine supplying the service.
<b>info</b>	An object with name / value pairs that the service has supplied. For example, in the case of an HTTP service, the path property will contain the path on the webservice so that the service URL would be <code>http://&lt;target&gt;:&lt;port&gt;/&lt;info["path"]&gt;</code> .

### Example

This example code will produce different output depending on where it is run. If there are no services advertised by DNS Service Discovery, this example will produce no output.

```
var oNotifications =
{
  resolve: function(status, info)
  {
    if(status == 0)
      console.println("RESOLVE: http://"
        + info.target + ":" + info.port + "/"
        + info.info.path);
    else console.println("ERROR: " + status);
  }
};
SOAP.resolveService({
  cType: "http",
  cDomain: "local.",
  cService: "Joe's Web Server",
  oAsync: oNotifications
});
```

The output depends on the current network environment – the following is a representative output:

```
RESOLVE: http://127.0.0.1:80/index.html
```

**request**

6.0			<b>F</b>	
-----	--	--	----------	--

Initiates a remote procedure call (RPC) or sends an XML message to a SOAP HTTP endpoint. The method will either wait for the endpoint to reply (synchronous processing) or call a method on the notification object (asynchronous processing).

**Parameters**

<b>cURL</b>	The URL for a SOAP HTTP Endpoint. See <a href="#">cURL</a> below for more details.
<b>oRequest</b>	An object that specifies the remote procedure name and parameters or the XML message to send. See <a href="#">oRequest</a> for more details on this object.
<b>oAsync</b>	(optional) An object that specifies that the method invocation will occur asynchronously. The default is for the request to be made synchronously. See <a href="#">oAsync</a> for more details.
<b>cAction</b>	(optional) In SOAP 1.1, this parameter is passed as the SOAPAction header. In SOAP 1.2, this parameter is passed as the action parameter in the Content-Type header. The default is for the action to be an empty string. See <a href="#">cAction</a> for additional comments.
<b>bEncoded</b>	(optional) Encoded the request using the SOAP Encoding described in the SOAP Specification. Otherwise, the literal encoding is used. The default is <b>true</b> .
<b>cNamespace</b>	(optional) A namespace for the message schema when the request does not use the SOAP Encoding. The default is to omit the schema declaration.
<b>oReqHeader</b>	(optional, version 7.0) An object that specifies a SOAP header to be included with the request. The default is to send a request with only a SOAP Body. See <a href="#">oReqHeader</a> for more details on this object.
<b>oRespHeader</b>	(optional, version 7.0) An object that will be populated with the SOAP headers returned when the method completes. The default is for the eaders to not be returned. See <a href="#">oRespHeader</a> for additional specifics.

---

<b>cVersion</b>	(optional, version 7.0) The version of the SOAP protocol to use when generating the XML Message – either 1.1 or 1.2. The default is to use "SOAPVersion.version_1_1". See <a href="#">cVersion</a> below.
<b>oAuthenticate</b>	(optional, version 7.0) An object that specifies the type of authentication scheme to use and to provide credentials . The default is for an interactive UI to be displayed if HTTP authentication is encountered. See <a href="#">oAuthenticate</a> for more details.
<b>cResponseStyle</b>	(optional, version 7.0) The style of message description to use when generating the response object – one of "JS", "XML" or "Message". The default is "JS". See <a href="#">cResponseStyle</a> for greater description.

---

See the [Additional Notes on the Parameters of SOAP.request](#).

## Returns

A response object if the method was called synchronously or nothing if the method was called asynchronously. See the description of **cResponseStyle** above for the object description.

An object literal. See the [Additional Notes on the Return Value](#)

## Exceptions

**SOAPError, NetworkError**

See the [Additional Notes on the Exceptions](#).

SOAP Faults will cause a **SOAPError** to be thrown. If there is a problem at the networking level, such as an unavailable endpoint, a **NetworkError** will be thrown.

## Additional Notes on the Parameters of SOAP.request

- **cURL**

The parameter **cURL** is the URL for a SOAP HTTP Endpoint. The URL method must be one of

- http—Connect to a server at a URI on a port. For example, <http://serverName:portNumber/URI>
- https—Connect to a secured (SSL) server at a URI on a port. For example, <https://serverName:portNumber/URI>

- **oRequest**

The **oRequest** parameter is an object literal that specifies the remote procedure name and the parameters to call. The object literal uses the fully qualified method name of the

remote procedure as the key. The namespace should be separated from the method name by a *colon*; for example, if the namespace for the method is <http://mydomain/methods> and the method name is `echoString()` then the fully qualified name would be <http://mydomain/methods:echoString>. The value of this key is an object literal, each key is a parameter of the method, and the value of each key is the value of the corresponding parameter of the method. For example:

```
oRequest: {
  "http://soapinterop.org/:echoString": {inputString: "Echo!"}
}
```

When passing parameters to a remote procedure, JavaScript types are bound to SOAP types automatically as listed in the table:

JavaScript Type	SOAP Type
String	xsd:string
Number	xsd:float
Date	xsd:dateTime
Boolean	xsd:boolean
<a href="#">ReadStream Object</a>	SOAP-ENC:base64
Array	SOAP-ENC:Array
Other	No type information

**NOTE:** The **xsd** namespace refers to the XML Schema Datatypes namespace (<http://www.w3.org/2001/XMLSchema>). The **SOAP-ENC** namespace refers to the SOAP Encoding namespace (<http://schemas.xmlsoap.org/soap/encoding/>).

The `oRequest` object supports the following properties:

Property	Description
<b>soapType</b>	<p>This is the SOAP Type that will be used for the value when generating the SOAP message; this is useful when a datatype is needed other than the automatic datatype binding described above. The type should be namespace qualified using the <code>&lt;namespace&gt;:&lt;type&gt;</code> notation, for example</p> <pre>http://mydomain/types:myType</pre> <p>However the <b>xsd</b> (the XMLSchema Datatypes namespace), <b>xsi</b> (the XMLSchema Instance namespace) and SOAP-ENC (the SOAP Encoding namespace) namespaces are implicitly defined in the SOAP message so the <b>soapType</b> can use these, as in <b>xsd:int</b> for the XMLSchema Datatype Integer type.</p>

Property	Description
<b>soapValue</b>	<p>(Version 6.0) This is the value that will be used when generating the SOAP message. It can be a string or a <a href="#">ReadStream Object</a>. The <b>soapValue</b> is passed <i>unescaped</i> (i.e., will not be XML Entity escaped); for example "&lt;" is not converted to "&amp;lt;" in the XML Message. Consequently the <b>soapValue</b> parameter can be a raw XML fragment which will be passed to the XML Message.</p> <p>(Version 7.0) The <b>soapValue</b> property could previously be a string or a stream. It can now also be an array of nodes which will be an ordered set of children of the node in the request message.</p>
<b>soapName</b>	<p>This is the element name that will be used when generating the SOAP message instead of the key name in the object literal. For example, integers are not supported in JavaScript, but an integer parameter to a SOAP method can be constructed as follows</p> <pre>var oIntParameter = {     soapType: "xsd:int",     soapValue: "1" };</pre> <p>Later, the <b>oRequest</b> parameter for the <b>SOAP.request</b> method might be</p> <pre>oRequest: {     "http://soapinterop.org/:echoInteger":         { inputInteger: oIntParameter } }</pre> <p>The <a href="#">Example 1</a> that follows the description of the <b>SOAP.request</b> illustrates this technique.</p>
<b>soapAttributes</b>	<p>(Version 7.0) An object specifying XML attributes to be included when building the element corresponding to the request node. The object keys are the attribute names and the corresponding value is the attribute value.</p>

Property	Description						
<b>soapQName</b>	<p>(Version 7.0) An object specifying the namespace qualified name (QName) of the request node. For example, in the following element <code>&lt;ns:name xmlns:ns="urn:example.org"&gt;</code> the element name is a QName consisting of a local name ("local") and a namespace ("urn:example.org").</p> <p>This object has two properties:</p> <table border="1"> <thead> <tr> <th>Property</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><b>localName</b></td> <td>A string indicating the local name of the QName</td> </tr> <tr> <td><b>nameSpace</b></td> <td>A string indicating the namespace of the QName.</td> </tr> </tbody> </table>	Property	Description	<b>localName</b>	A string indicating the local name of the QName	<b>nameSpace</b>	A string indicating the namespace of the QName.
Property	Description						
<b>localName</b>	A string indicating the local name of the QName						
<b>nameSpace</b>	A string indicating the namespace of the QName.						
<b>soapAttachment</b>	<p>(Version 7.0) A boolean indicating that the <b>soapValue</b> contents of the node should be encoded as an attachment according to the SwA specification. The <b>soapValue</b> must be a stream if the corresponding <b>soapAttachment</b> property is <b>true</b>, otherwise an exception will be thrown.</p>						
<b>soapParamOrder</b>	<p>(Version 7.0) An array indicating the order in which RPC parameters should be sent to the server. The array is a set of strings with the parameter names. This value is only applicable when <b>bEncoding</b> is <b>true</b>.</p>						

- **oAsync**

(Version 6.0) The **oAsync** object literal must have a function called **response** which will be called with two parameters (**oResult** and **cURI**) when the response returns. **oResult** is the same result object that would have been returned from the request call if it was called synchronously. **cURI** is the URI of the endpoint that the request was made to.

(Version 7.0) The **oAsync** object response callback now has the following parameters:

Parameter	Description
<b>response</b>	The response object from the SOAP request.
<b>uri</b>	The URI that the SOAP request was made to.
<b>exception</b>	An exception object (see the exceptions below) if there was an error, <b>null</b> otherwise.



Parameter	Description
<b>header</b>	A response SOAP header (see the description of the <b>oRespHeader</b> parameter) or <b>null</b> if there are no response headers.

- **cAction**

The **cAction** parameter is the SOAPAction header for the method. The SOAPAction is a URN written to an HTTP header used by firewalls and servers to filter SOAP requests. The WSDL file for the SOAP service or the SOAP service description will usually describe the SOAPAction header required (if any).

- **oReqHeader**

The **oReqHeader** object specifies a request header to include with the SOAP request. The object is specified in the same way as the **oRequest** object except for two additional properties that can be specified in the request description:

Property	Description
<b>soapActor</b>	The recipient (or actor specified as a URI) that the SOAP header should be processed by. The default is that the header will be processed by the first recipient to process the request.
<b>soapMustUnderstand</b>	A boolean indicating that the request body can not be interpreted if this header type is not understood by the recipient. The default is that understanding the header is optional.

- **oRespHeader**

The **oRespHeader** object is populated once the function returns if the function is being called synchronously (the header will be passed to the **oAsync** callback method otherwise). See the description of the **cResponseStyle** parameter for the object format.

- **cVersion**

The **cVersion** parameter indicates the SOAP Version to use when generating the message. The **cVersion** parameter is one of:

Value	Description
<b>SOAPVersion.version_1_1</b>	(Default) Encode the message using the SOAP 1.1 protocol.
<b>SOAPVersion.version_1_2</b>	Encode the message using the SOAP 1.2 protocol.

- **oAuthenticate**

The **oAuthenticate** object specifies how to handle HTTP authentication or specifies credentials to use for Web Service Security. The default is to present a user interface to

the user to handle HTTP authentication challenges for BASIC and DIGEST authentication modes. The **oAuthenticate** object can have the following properties:

Property	Description
<b>Username</b>	A string containing the username to use for authentication.
<b>Password</b>	A string containing the authentication credential to use.
<b>UsePlatformAuth</b>	A boolean indicating that platform authentication should be used. If platform authentication is enabled, the <b>Username</b> and <b>Password</b> are ignored and the underlying platform networking code is used. This may cause an authentication UI to be shown to the user and/or the credentials of the currently logged in user to be used. The default is <b>false</b> and is only supported on the Windows platform.

- **cResponseStyle**

The **cResponseStyle** parameter is an enumerated type indicating how the return value (in the case of the SOAP Body) and the **oRespHeader** object (in the case of a SOAP header) will be structured:

Property	Description
<b>SOAPMessageStyle.JS</b>	(Default) The response will be an object describing the SOAP Body (or SOAP Header) of the returned message (this is the result that Acrobat 6.0 produced). This is recommended when using the SOAP encoding for the request but is not ideal when using the literal encoding – using the XML or Message style is better.
<b>SOAPMessageStyle.XML</b>	The response will be a stream object containing the SOAP Body (or SOAP Header) as an XML fragment. If there are any attachments associated with the response, the Stream object will have an object property <b>oAttachments</b> . The object keys are the unique names for the attachment parts and the value must be a Stream object containing the attachment contents.

Property	Description
<b>SOAPMessageStyle.Message</b>	<p>The response will be an object describing the SOAP Body (or SOAP Header) corresponding to the XML Message. This differs from the JS response style in the following ways:</p> <ul style="list-style-type: none"> <li>● XML Elements are returned as an array of objects rather than an object to maintain order and allow elements with the same name.</li> <li>● XML Attributes are preserved using the <b>soapAttributes</b> property.</li> <li>● Namespaces are processed and returned in the <b>soapName</b> and <b>soapQName</b> properties.</li> <li>● The content of an element is in the <b>soapValue</b> property.</li> </ul>

A [ReadStream Object](#) is an object literal that represents a stream of data. The object literal should contain a function called **read**, which takes the form:

```
var readStreamObject = {
    read: function(nBytes) {...};
}
```

The **read()** method takes the number of bytes to read and returns a hex encoded string with the data from the stream. The **read()** method returns a zero length string to indicate end of stream. Alternatively, you can use the **SOAP.streamFromString** function to create a read stream.

### Additional Notes on the Return Value

If there is no **oAsync** parameter (that is, a synchronous request) then **request** returns the result from the SOAP method. Otherwise, nothing is returned. The SOAP types in the result are mapped to JavaScript types as follows:

SOAP Type	JavaScript Type
xsd:string	String
xsd:integer	Number
xsd:float	Number
xsd:dateTime	Date
xsd:boolean	Boolean
xsd:hexBinary	<a href="#">ReadStream Object</a>
xsd:base64Binary	<a href="#">ReadStream Object</a>
SOAP-ENC:base64	<a href="#">ReadStream Object</a>

SOAP Type	JavaScript Type
SOAP-ENC:Array	Array
No Type Information	String

### Additional Notes on the Exceptions

- SOAPError: This exception is thrown when the SOAP endpoint returns a SOAPFault. The SOAPError Exception object has the following properties:

Property	Description
<b>faultCode</b>	A string indicating the SOAP Fault Code for this fault.
<b>faultActor</b>	A string indicating the SOAP Actor that generated the fault.
<b>faultDetail</b>	A string indicating detail associated with the fault.

- NetworkError: This exception is thrown when there is a failure from the underlying HTTPtransport layer or in obtaining a Network connection. The NetworkError Exception object has the following property:

Property	Description
<b>statusCode</b>	An HTTP Status code or -1 if the Network connection could not be made.

- Standard Acrobat Exceptions can also be thrown.

**NOTE:** If the method was called asynchronously then the exception object may be passed to the **response ()** callback method.

### Example 1

A service WSDL Document URL is needed. These can be obtained from the "Round 2 Interop Services - using SOAP 1.2" section at the following URL:  
<http://www.whitemesa.com/interop.htm>.

```
var cURL = <get a URL for this service from
           http://www.whitemesa.com/interop.htm>;

var cTestString = "This is my test string";

// Call the echoString SOAP method -- it is an RPC Encoded method
var response = SOAP.request(
{
  cURL: cURL,
  oRequest: {
    "http://soapinterop.org/:echoString": {
      inputString: cTestString
    }
  }
},
```

```

        cAction: "http://soapinterop.org/"
    });

    var result =
    response["http://soapinterop.org/:echoStringResponse"] ["return"];

    // This should be the same as cTestString
    console.println(result + " == " + cTestString);

    // Call the echoInteger SOAP method -- JavaScript doesn't support
    // integers so we make our own integer object.
    var oTestInt =
    {
        soapType: "xsd:int",
        soapValue: "10"
    };

    var response = SOAP.request(
    {
        cURL: cURL,
        oRequest: {
            "http://soapinterop.org/:echoInteger": {
                inputInteger: oTestInt
            }
        },
        cAction: "http://soapinterop.org/"
    });

    var result =
    response["http://soapinterop.org/:echoIntegerResponse"] ["return"];

    // This should be the same as oTestInt.soapValue
    console.println(result + " == " + oTestInt.soapValue);

```

This produces the following output:

```

This is my test string == This is my test string
10 == 10

```

## Example 2

This example illustrates setting a SOAP Header and getting it back.

```

var cURL = <URL of a Service>;
var NS = "http://adobe.com/FEAT/:";
var oHeader = {};
oHeader[NS + "testSession"] =
{
    soapType: "xsd:string",
    soapValue: "Header Test String"
};
var oResultHeader = {};
var oRequest = {};

```

```
oRequest[NS + "echoHeader"] = {};
var response = SOAP.request(
{
  cURL: cURL,
  oRequest: oRequest,
  cAction: "http://soapinterop.org/",
  oReqHeader: oHeader,
  oRespHeader: oResultHeader
});
```

**Example 3**

This example illustrate the use of HTTP Authentication.

```
var oAuthenticator =
{
  Username: "myUserName",
  Password: "myPassword"
};
var response = SOAP.request(
{
  cURL: cURL,
  oRequest: {
    "http://soapinterop.org/:echoString":
    {
      inputString: cTestString
    }
  },
  cAction: "http://soapinterop.org/",
  oAuthenticate: oAuthenticator
});
```

**response**

6.0			<b>F</b>
-----	--	--	----------

This method Initiates a remote procedure call (RPC) or sends an XML message to a SOAP HTTP endpoint without waiting for a reply.

**Parameters**


---

<b>cURL</b>	<p>The URL for a SOAP HTTP Endpoint. The URL method must be one of</p> <ul style="list-style-type: none"> <li>● <b>http</b>—Connect to a server at a URI on a port. For example, <a href="http://serverName:portNumber/URI">http://serverName:portNumber/URI</a></li> <li>● <b>https</b>—Connect to a secured (SSL) server at a URI on a port. For example, <a href="https://serverName:portNumber/URI">https://serverName:portNumber/URI</a></li> </ul> <p>See <b>cURL</b> under <b>SOAP.request</b>.</p>
-------------	--

---

---

<b>oRequest</b>	An object that specifies the remote procedure name and parameters or the XML message to send. See <a href="#">oRequest</a> under <b>SOAP.request</b> .
<b>cAction</b>	(optional) The SOAP Action header for this request as specified by the SOAP Specification. The default is for the SOAP Action to be empty. See <a href="#">cAction</a> under <b>SOAP.request</b> .
<b>bEncoded</b>	(optional) Encoded the request using the SOAP Encoding described in the SOAP Specification. The default is <b>true</b> .
<b>cNamespace</b>	(optional) A namespace for the message schema when the request does not use the SOAP Encoding (the <b>bEncoded</b> flag is <b>false</b> ). The default is to have no namespace.
<b>oReqHeader</b>	(optional, version 7.0) An object that specifies a SOAP header to be included with the request. The default is to send a request with only a SOAP Body. See <a href="#">oReqHeader</a> under <b>SOAP.request</b> .
<b>cVersion</b>	(optional, version 7.0) The version of the SOAP protocol to use. The default is to use "SOAPVersion.version_1_1". See <a href="#">cVersion</a> under <b>SOAP.request</b> .
<b>oAuthenticate</b>	(optional, version 7.0) An object that specifies the type of authentication scheme to use and to provide credentials. The default is for an interactive UI to be displayed if HTTP authentication is encountered. See <a href="#">oAuthenticate</a> under <b>SOAP.request</b> .

---

### Returns

Boolean

### Exceptions

If there is a problem at the networking level, such as an unavailable endpoint, a **NetworkError** will be thrown.

### Example

See the [Example 1](#) that follows the **SOAP.request** method.

## streamDecode

6.0			
-----	--	--	--

This function allows the **oStream** object to be decoded with the specified encoding type, **cEncoder**. It returns a [ReadStream Object](#) (see [request](#)) which will have been decoded appropriately. Typically this be would used to access data returned as part of a SOAP method when it was encoded in Base64 or Hex encoding.

### Parameters

<b>oStream</b>	A stream object to be decoded with the specified encoding type.
<b>cEncoder</b>	Permissible values for this string are "hex" (for Hex encoded) and "base64" (Base 64 encoded).

### Returns

[ReadStream Object](#)

## streamDigest

7.0			
-----	--	--	--

This function allows the **oStream** object to be digested with the specified encoding type, **cEncoder**. It returns a [ReadStream Object](#) which will have the computed digest of the **oStream**. Typically this would be used to compute a digest to validate the integrity of the original data stream or as part of an authentication scheme for a Web Service.

### Parameters

<b>oStream</b>	A stream object to compute the digest of using the specified message digest algorithm.	
<b>cEncoder</b>	The digest algorithm to use. The <b>cEncoder</b> parameter must be one of the following values:	
	Digest Algorithm	Description
	<b>StreamDigest.MD5</b>	Digest the content using the MD5 Digest Algorithm (see RFC 1321)
	<b>StreamDigest.SHA1</b>	Digest the content using the SHA-1 Digest Algorithm (see RFC 3174)



**Returns**

A [ReadStream Object](#) with the binary digest of the stream. The result must be converted to a text format (such as Base64 or HEX) using `SOAP.streamEncode()` to be used as a string.

**Example**

```
var srcStm = SOAP.streamFromString("This is a string I want to digest");
var digest = SOAP.streamDigest(srcStm, StreamDigest.SHA1);
```

**streamEncode**

6.0			
-----	--	--	--

This function allows the `oStream` object to be encoded with the specified encoding type, `cEncoder`. It returns a [ReadStream Object](#) (see [request](#)) which will have the appropriate encoding applied. Typically this would be used to pass data as part of a SOAP method when it must be encoded in Base64 or Hex encoding.

**Parameters**

<code>oStream</code>	A stream object to be encoded with the specified encoding type.
<code>cEncoder</code>	Permissible values for this string are "hex" (for Hex encoded) and "base64" (Base 64 encoded).

**Returns**

[ReadStream Object](#)

**streamFromString**

6.0			
-----	--	--	--

This function converts a string to a [ReadStream Object](#) (see [request](#)). Typically this would be used to pass data as part of a SOAP method.

**Parameters**

<code>cString</code>	The string to be converted
----------------------	----------------------------

**Returns**

[ReadStream Object](#)

## stringFromStream

6.0			
-----	--	--	--

This function converts a [ReadStream Object](#) (see [request](#)) to a string. Typically, this would be used to examine the contents of a stream object returned as part of a response to a SOAP method.

### Parameters

---

<b>oStream</b>	<a href="#">ReadStream Object</a> to be converted.
----------------	--

---

### Returns

String

---

## Sound Object

5.0			
-----	--	--	--

This object is the representation of a sound that is stored in the document. The array of all **sound** objects can be obtained from `doc.sounds`. See also `doc` methods [getSound](#), [importSound](#), and [deleteSound](#).

---

## Sound Properties

### name

The name associated with this sound object.

*Type: String*

*Access: R.*

### Example

```
console.println("Dumping all sound objects in this document.");
var s = this.sounds;
for (var i = 0; i < this.sounds.length; i++)
    console.println("Sound[" + i + "]=" + s[i].name);
```

---

## Sound Methods

### play

Plays the sound asynchronously.

#### Parameters

None

#### Returns

Nothing

### pause

Pauses the currently playing sound. If the sound is already paused then the sound play is resumed.

#### Parameters

None

#### Returns

Nothing

### stop

Stops the currently playing sound.

#### Parameters

None

#### Returns

Nothing

---

## Span Object

6.0			
-----	--	--	--

A span object is used to represent a length of text and its associated properties in a rich text form field or annotation. The rich text value of a form field or annotation consists of an array of span objects representing the text and formatting of the annotation. It is important to note that the span objects are a copy of the rich text value of the field or annotation. Use the `field.richValue`, `event.richValue` ( and `richChange`, `richChangeEx`), or `annot.richContents` to modify and reset the rich text value to update the field.

---

## Span Properties

### alignment

The horizontal alignment of the text. Alignment for a line of text is determined by the first span on the line. The values of **alignment** are

```
left
center
right
```

The default value is **left**.

*Type: String*                      *Access: R/W.*

The example following [superscript](#) illustrates the usage of **alignment**.

### fontFamily

The font family used to draw the text. It is an array of family names to be searched for in order. The first entry in the array is the font name of the font to use; the second entry is an optional generic family name to use if an exact match of the first font is not found. The generic family names are

```
symbol, serif, sans-serif, cursive, monospace, fantasy
```

The default generic family name is **sans-serif**.

*Type: Array*                      *Access: R/W.*

#### Example

Set the [defaultStyle](#) font family for a rich text field.

```
f = this.getField("Text1");
style = f.defaultStyle;

// if Courier Std is not found on the user's system, use a monospace
style.fontFamily = ["Courier Std", "monospace" ];
f.defaultStyle = style;
```

### fontStretch

Specifies the normal, condensed or extended face from a font family to be used to draw the text. The values of **fontStretch** are

```
ultra-condensed, extra-condensed, condensed, semi-condensed, normal,
semi-expanded, expanded, extra-expanded, ultra-expanded
```

The default value is **normal**.

*Type: String*                      *Access: R/W.*

## fontStyle

Specifies the text is drawn with an italic or oblique font.

```
italic
normal
```

The default is **normal**.

*Type: String*                      *Access: R/W.*

## fontWeight

The weight of the font used to draw the text. For the purposes of comparison, normal is anything under 700 and bold is greater than or equal to 700. The values of **fontWeight** are

```
100, 200, 300, 400, 500, 600, 700, 800, 900
```

The default value is 400.

*Type: Number*                      *Access: R/W.*

## text

The text within the span.

*Type: String*                      *Access: R/W.*

The example following [superscript](#) illustrates the usage of **text**.

## textColor

The RGB color to be used to draw the text. The value of **textColor** is a color array, see the [Color Object](#) for a description of color array. The default color is black.

*Type: Color Array*                      *Access: R/W.*

The example following [superscript](#) illustrates the usage of **textColor**.

## textSize

The point size of the text. The value of **textSize** can be any number between 0 and 32767 inclusive. A text size of zero means to use the largest point size that will allow all text data to fit in the field's rectangle.

The default text size is 12.0.

*Type: Number*                      *Access: R/W.*

The example following [field.richValue](#) illustrates the usage of **textSize**.

## strikethrough

If **strikethrough** is **true**, the text is drawn with a strikethrough. The default is **false**.

*Type: Boolean*

*Access: R/W.*

## subscript

Specifies the text is subscript. If **true**, subscript text is drawn with a reduced point size and a lowered baseline. The default is **false**.

*Type: Boolean*

*Access: R/W.*

## superscript

Specifies the text is superscript. If **true**, superscript text is drawn with a reduced point size and a raised baseline. The default is **false**.

*Type: Boolean*

*Access: R/W.*

### Example

Write rich text to a rich text field using various properties. See **field.richValue** for more details and examples.

```
var f = this.getField("myRichField");

// need an array to hold the span objects
var spans = new Array();

// each span object is an object, so we must create one
spans[0] = new Object();
spans[0].alignment = "center";
spans[0].text = "The answer is x";

spans[1] = new Object();
spans[1].text = "2/3";
spans[1].superscript = true;

spans[2] = new Object();
spans[2].superscript = false;
spans[2].text = ". ";

spans[3] = new Object();
spans[3].underline = true;
spans[3].text = "Did you get it right?";
spans[3].fontStyle = "italic";
spans[3].textColor = color.red;
```

```
// now assign our array of span objects to the field using
// field.richValue
f.richValue = spans;
```

## underline

If **underline** is **true**, the text is underlined. The default is **false**.

Type: Boolean      Access: R/W.


The example following **superscript** illustrates the usage of **underline**.

---

## Spell Object

5.0				
-----	--	--	--	--

This object allows users to check the spelling of Comments and Form Fields and other spelling domains. To be able to use the **spell** object, the user must have installed the Acrobat Spelling plug-in and the spelling dictionaries.

**NOTE:** (Adobe Reader ) The **spell** object is not available in versions of the Adobe Reader prior to 7.0. In Adobe Reader 7.0, all properties and methods—with the exception of **customDictionaryCreate**, **customDictionaryDelete** and **customDictionaryExport**—are accessible.


---

## Spell Properties

### available

5.0				
-----	--	--	--	--

**true** if the **spell** object is available.

**NOTE:** (Adobe Reader ) For Adobe Reader, this property is available only for Adobe Reader 7.0 or later.

Type: Boolean      Access: R.

### Example


```
console.println("Spell checking available: " + spell.available);
```

## dictionaryNames

5.0				
-----	--	--	--	--

An array of available dictionary names. A subset of this array can be passed to [check](#), [checkText](#), and [checkWord](#), and to [spellDictionaryOrder](#) to force the use of a specific dictionary or dictionaries and the order they should be searched.

A listing of valid dictionary names for the user's installation can be obtained by executing `spell.dictionaryNames` from the console.

**NOTE:** (Adobe Reader ) For Adobe Reader, this property is available only for Adobe Reader 7.0 or later.


Type: Array

Access: R.

## dictionaryOrder

5.0				
-----	--	--	--	--

The dictionary array search order specified by the user on the Spelling Preferences panel. The Spelling plug-in will search for words first in the `doc.spellDictionaryOrder` array if it has been set for the document, and then it will search this array of dictionaries.

**NOTE:** (Adobe Reader ) For Adobe Reader, this property is available only for Adobe Reader 7.0 or later.

Type: Array

Access: R.


## domainNames

5.0				
-----	--	--	--	--

The array of spelling domains that have been registered with the Spelling plug-in by other plug-ins. A subset of this array can be passed to [check](#) to limit the scope of the spell check.

Depending on the user's installation, valid domains can include:

- Everything
- Form Field
- All Form Fields
- Comment
- All Comments

**NOTE:** (Adobe Reader ) For Adobe Reader, this property is available only for Adobe Reader 7.0 or later.

Type: Array


Access: R.



## languages

6.0				
-----	--	--	--	--

This property returns the array of available ISO 639-2/3166-1 language/country codes. A subset of this array can be passed to the [check](#), [checkText](#), [checkWord](#), and [customDictionaryCreate](#) methods, and to the `doc.spellLanguageOrder` property to force the use of a specific language or languages and the order they should be searched.

**NOTE:** (Adobe Reader ) For Adobe Reader, this property is available only for Adobe Reader 7.0 or later.

Type: Array

Access: R.

Depending on the user's installation, valid language/country codes can include:

Code	Description	Code	Description
ca_ES	Catalan	el_GR	Greek
cs_CZ	Czech	hu_HU	Hungarian
da_DK	Danish	it_IT	Italian
nl_NL	Dutch	nb_NO	Norwegian – Bokmal
en_CA	English – Canadian	nn_NO	Norwegian – Nynorsk
en_GB	English – UK	pl_PL	Polish
en_US	English – US	pt_BR	Portuguese – Brazil
fi_FI	Finish	pt_PT	Portuguese
fr_CA	French – Canadian	ru_RU	Russian
fr_FR	French	es_ES	Spanish
de_DE	German	sv_SE	Swedish
de_CH	German – Swiss	tr_TR	Turkish

**NOTE:** In version 7.0, the entries in this array are different from the entries returned in version 6.0. On input from JavaScript, the Acrobat 6.0 ISO codes are internally mapped onto the new ISO codes in order to not break any JavaScript code developed for Acrobat 6.0. Codes are not translated on output.

### Example


List all available language codes.

```
console.println( spell.languages.toSource() );
```

## languageOrder

6.0				
-----	--	--	--	--

This property returns the dictionary search order as an array of ISO 639-2, 3166 language codes. This is the order specified by the user on the Spelling Preferences panel. The Spelling plug-in will search for words first in the `doc.spellLanguageOrder` array if it has been set for the document, and then it will search this array of languages.

**NOTE:** (Adobe Reader ) For Adobe Reader, this property is available only for Adobe Reader 7.0 or later.

Type: Array

Access: R.

### Example




Get a listing of the dictionary search order.

```
console.println( spell.languageOrder.toSource() );
```

---

## Spell Methods

### addDictionary

				
---	---	--	---	--

Adds a dictionary to the list of available dictionaries .

A dictionary actually consists of four files: `DDDxxxxxx.hyp`, `DDDxxxxxx.lex`, `DDDxxxxxx.clx`, and `DDDxxxxxx.env`. The **cFile** parameter must be the device-independent path of the `.hyp` file. For example, `"/c/temp/testdict/TST.hyp"`. Spelling will look in the parent directory of the `TST.hyp` file for the other three files. All four file names must start with the same unique 3 characters to associate them with each other, and they must end with the dot three extensions listed above, even on a Macintosh.

**NOTE:** Beginning with Acrobat 6.0, this method is no longer supported. The return value of this method is always **false**. Use the [customDictionaryOpen](#) method.

### Parameters

<b>cFile</b>	The device-independent path to the dictionary files.
<b>cName</b>	The dictionary name used in the spelling dialog and can be used as the input parameter to the <a href="#">check</a> , <a href="#">checkText</a> , and <a href="#">checkWord</a> .

---



**bShow** (optional) When **true** (the default), Spelling combines the **cName** value with "User: " and shows that name in all lists and menus. For example if **cName** is "Test", Spelling adds "User: Test" to all lists and menus. When **false**, Spelling does not show this custom dictionary in any lists or menus.

---


**Returns**

**false**


**addWord**

5.0				
-----	---	---	--	--

Adds a new word to a dictionary. See also the [removeWord](#).

**NOTES:** (Security ): Beginning with Acrobat 7.0, this method is allowed only during console or batch events. See also [Privileged versus Non-privileged Context](#).

Internally, the Spell Check Object scans the user "Not-A-Word" dictionary and removes the word if it is listed there. Otherwise, the word is added to the user dictionary. The actual dictionary is not modified.

**NOTE:** (Adobe Reader ) For Adobe Reader, this method is available only for Adobe Reader 7.0 or later.

**Parameters**

---

<b>cWord</b>	The new word to add.
<b>cName</b>	(optional) The dictionary name or language code. An array of the currently installed dictionaries can be obtained using <a href="#">dictionaryNames</a> or <a href="#">languages</a> .

---


**Returns**

**true** if successful, otherwise, **false**.

**check**

5.0				
-----	--	--	--	--

Presents the Spelling dialog to allow the user to correct misspelled words in form fields, annotations, or other objects.

**NOTE:** (Adobe Reader ) For Adobe Reader, this method is available only for Adobe Reader 7.0 or later.

**Parameters**

---

<b>aDomain</b>	(optional) An array of document objects that should be checked by the Spelling plug-in, for example form fields or comments. When you do not supply an array of domains the "Everything" domain will be used. An array of the domains that have been registered can be obtained using the <a href="#">domainNames</a> .
<b>aDictionary</b>	(optional) The array of dictionary names or language codes that the spell checker should use. The order of the dictionaries in the array is the order the spell checker will use to check for misspelled words. An array of the currently installed dictionaries can be obtained using <a href="#">spell.dictionaryNames</a> or <a href="#">spell.languages</a> . When this parameter is omitted the <a href="#">spellDictionaryOrder</a> list will be searched followed by the <a href="#">dictionaryOrder</a> list.

---

**Returns**

**true** if the user changed or ignored all of the flagged words. When the user dismisses the dialog before checking everything the method returns **false**.


**Example**

```
var dictionaries = ["de", "French", "en-GB"];
var domains = ["All Form Fields", "All Annotations"];
if (spell.check(domains, dictionaries) )
    console.println("You get an A for spelling.");
else
    console.println("Please spell check this form before you submit.");
```

**checkText**

5.0				
-----	--	--	--	--

Presents the spelling dialog to allow the user to correct misspelled words in the specified string.

**NOTE:** (Adobe Reader ) For Adobe Reader, this method is available only for Adobe Reader 7.0 or later.

**Parameters**

---

<b>cText</b>	The string to check.
--------------	----------------------

---

---

**aDictionary** (optional) The array of dictionary names or language codes that the spell checker should use. The order of the dictionaries in the array is the order the spell checker will use to check for misspelled words. An array of the currently installed dictionaries can be obtained using `spell.dictionaryNames` or `spell.languages`. When this parameter is omitted the `spellDictionaryOrder` list will be searched followed by the `dictionaryOrder` list.

---

**Returns**

The result from the spelling dialog in a new string.


**Example**

```
var f = this.getField("Text Box") // a form text box
f.value = spell.checkText(f.value); // let the user pick the dictionary
```

**checkWord**

5.0				
-----	--	--	--	--

Checks the spelling of a specified word.

**NOTE:** (Adobe Reader ) For Adobe Reader, this method is available only for Adobe Reader 7.0 or later.

**Parameters**


---

<b>cWord</b>	The word to check.
<b>aDictionary</b>	(optional) The array of dictionary names or language codes that the spell checker should use. The order of the dictionaries in the array is the order the spell checker will use to check for misspelled words. An array of the currently installed dictionaries can be obtained using <code>spell.dictionaryNames</code> or <code>spell.languages</code> . When this parameter is omitted the <code>spellDictionaryOrder</code> list will be searched followed by the <code>dictionaryOrder</code> list.

---

**Returns**

A `null` object if the word is correct, otherwise an array of alternative spellings for the unknown word.

**Example 1**

```
var word = "subpinna"; /* misspelling of "subpoena" */
var dictionaries = ["English"];
var f = this.getField("Alternatives") // alternative spellings listbox
f.clearItems();
f.setItems(spell.checkWord(word, dictionaries));
```

**Example 2**


The following script goes through the document and marks with a squiggle annot any misspelled word. The contents of the squiggle annot contains the suggested alternative spellings. The script can be executed from the console, as a mouse up action within the document, a menu, or as a batch sequence.

```
var ckWord, numWords;
for (var i = 0; i < this.numPages; i++ )
{
    numWords = this.getPageNumWords(i);
    for (var j = 0; j < numWords; j++)
    {
        ckWord = spell.checkWord(this.getPageNthWord(i, j))
        if ( ckWord != null )
        {
            this.addAnnot({
                page: i,
                type: "Squiggly",
                quads: this.getPageNthWordQuads(i, j),
                author: "A. C. Acrobat",
                contents: ckWord.toString()
            });
        }
    }
}
```

**customDictionaryClose**

6.0				
-----	--	--	--	--

Closes a custom dictionary that was opened using [customDictionaryOpen](#) or [customDictionaryCreate](#).

**NOTE:** (Adobe Reader ) For Adobe Reader, this method is available only for Adobe Reader 7.0 or later.

**Parameters**

<b>cName</b>	Dictionary name used when this dictionary was opened or created.
--------------	--

**Returns**

**true** if successful, **false** on failure.

## customDictionaryCreate

6.0		Ⓢ	ⓧ	
-----	--	---	---	--

Use this method to create a new custom dictionary file and add it to the list of available dictionaries.

**NOTES:** (Security Ⓢ): This method is allowed only during console, menu or batch events. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

### Parameters

<b>cName</b>	Dictionary name used in the spelling dialog and can be used as the input parameter to <a href="#">check</a> , <a href="#">checkText</a> , and <a href="#">checkWord</a> methods.
<b>cLanguage</b>	(optional) Use this parameter to associate this dictionary with a language. A list of available languages can be obtained from the <code>spell.languages</code> property.
<b>bShow</b>	(optional) If <b>true</b> , the default, spelling will combine the <b>cName</b> parameter with "User: " and show that name in all lists and menus. For Example, if <b>cName</b> is "Test", spelling will add "User: Test" to all lists and menus. When <b>bShow</b> is <b>false</b> , Spelling will not show this custom dictionary in any lists or menus.

### Returns

**true** if successful, **false** on failure. This method will fail if the user does not have read and write permission to this directory.

### Example

Open this document, the *Acrobat JavaScript Scripting Reference*, in Acrobat and execute the following script in the console. This script goes through the bookmarks and extracts the first word of each bookmark. If that word is already in a dictionary, it is discarded. An unknown word—assumed to be the name of an Acrobat JavaScript object, property or method—is added into a newly created dictionary called "JavaScript".

```
spell.customDictionaryCreate("JavaScript", "en", true);
function GetJSTerms(bm, nLevel)
{
    var newWord = bm.name.match(re);
    var ckWord = spell.checkWord( newWord[0] );
    if ( ckWord != null )
    {
        var cWord = spell.addWord( newWord[0], "JavaScript");
        if ( cWord ) console.println( newWord[0] );
    }
}
```

```

        if (bm.children != null)
        for (var i = 0; i < bm.children.length; i++)
        GetJSTerms(bm.children[i], nLevel + 1);
    }
    console.println("\nAdding New words to the \"JavaScript\" "
        + "dictionary:");
    var re = /^\\w+\\/;
    GetJSTerms(this.bookmarkRoot, 0);

```

## customDictionaryDelete

6.0		Ⓢ	ⓧ	
-----	--	---	---	--

Use this method to close and delete a custom dictionary file that was opened via [customDictionaryOpen](#) or [customDictionaryCreate](#).

**NOTES:** (Security Ⓢ): This method is allowed only during console, menu or batch events. See also [Privileged versus Non-privileged Context](#).

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

### Parameters

<b>cName</b>	The name of the dictionary to be deleted. This is the name used when this dictionary was opened or created.
--------------	---

### Returns

**true** if successful, **false** on failure. This method will fail if the user does not have sufficient file system permission.

### Example

Delete a custom dictionary.

```
spell.customDictionaryDelete("JavaScript");
```

## customDictionaryExport


6.0		Ⓢ	ⓧ	
-----	--	---	---	--

This method will export a custom dictionary to a new file that was opened using the spell methods [customDictionaryOpen](#) or [customDictionaryCreate](#).

The user will be prompted for an export directory. The custom dictionary will be saved there as a .clam file using the dictionary name and the language specified on [customDictionaryCreate](#). For example if the dictionary name is "JavaScript" and the "en" language as specified when it was created then the export file name will be JavaScript-eng.clam.



Exported custom dictionaries can be used in subsequent [customDictionaryOpen](#) calls.

**NOTES:** (Security Privileged versus Non-privileged Context.

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged, see [JavaScript Execution through the Menu](#) for details.

### Parameters

<b>cName</b>	The dictionary name used when this dictionary was opened or created.
--------------	--

### Returns

**true** if successful, **false** on failure. This method will fail if the user does not have sufficient file system permission.

### Example

Export a custom dictionary for distribution to other users. The exported dictionary can then be sent to other users. (See the example that follows [customDictionaryCreate](#).)


```
spell.customDictionaryExport("JavaScript");
```

## customDictionaryOpen

6.0				
-----	--	--	--	--

Use this method to add an custom export dictionary to the list of available dictionaries. See [customDictionaryExport](#).

**NOTE:** A custom dictionary file can be created using the [customDictionaryCreate](#) and [customDictionaryExport](#) methods.

**NOTE:** (Adobe Reader ) For Adobe Reader, this method is available only for Adobe Reader 7.0 or later.

### Parameters

<b>cDIPath</b>	The device independent path to the custom dictionary file.
<b>cName</b>	Dictionary name used in the spelling dialog and can be used as the input parameter to <a href="#">check</a> , <a href="#">checkText</a> , and <a href="#">checkWord</a> methods

---

<b>bShow</b>	(optional) If <b>true</b> , the default, Spelling will combine the <b>cName</b> parameter with "User: " and show that name in all lists and menus. For Example if <b>cName</b> is "Test", Spelling will add "User: Test" to all lists and menus. When <b>bShow</b> is <b>false</b> , Spelling will not show this custom dictionary in any lists or menus.
--------------	---

---

**Returns**

**true** if successful, **false** on failure. This method will fail if the user does not have read permission for the file.

**Example**

This example continues the ones begun following [customDictionaryCreate](#) and [customDictionaryExport](#).

Add an custom export dictionary to the list of available dictionaries. The user places the custom export dictionary any any folder for which there is read/write permission. One particular choice is the user `dictionaries` folder. This location of this folder can be obtained from the `app.getPath` method.

```
app.getPath("user", "dictionaries");
```

Once the export dictionary has been placed, listing it can be made automatic by adding some folder level JavaScript. The path to the user `JavaScripts` can be obtained by executing

```
app.getPath("user", "javascript");
```

Finally, create an `.js` file in this folder and add the line

```
var myDictionaries = app.getPath("user", "dictionaries");
spell.customDictionaryOpen( myDictionaries, "JavaScripts", true);
```


The next time Acrobat is started, the "JavaScript" dictionary will be open and available.

**ignoreAll**

6.0				
-----	--	--	--	--

Adds or removes a word from the Spelling ignored-words list of the current document.

**NOTE:** A document must be open in the viewer or this method will throw an exception.

**NOTE:** (Adobe Reader ) For Adobe Reader, this method is available only for Adobe Reader 7.0 or later.

**Parameters**

<b>cWord</b>	The word to be added or removed from the ignored list.
<b>bIgnore</b>	(optional) If <b>true</b> (the default), the word is added to the document ignored word list; if <b>false</b> , the word is removed from the ignored list.

**Returns**

**true** if successful. An exception is thrown if there is no document open in the viewer when this method is executed.

**Example**

```
var bIgnored = spell.ignoreAll("foo");
if (bIgnored) console.println("\foo\" will be ignored);
```

**removeDictionary**

X	P		X	
---	---	--	---	--

Removes a user dictionary that was added via [addDictionary](#).

**NOTE:** Beginning with Acrobat 6.0, this method is no longer supported. The return value of this method is always **false**. Use the [customDictionaryClose](#) method.

**Parameters**

<b>cName</b>	The name of the dictionary to remove. Must be the same name as was used with <a href="#">addDictionary</a> .
--------------	--

**Returns**

**false**


**removeWord**

5.0	P			
-----	---	--	--	--

Removes a word from a dictionary. Words cannot be removed from user dictionaries that were created using either [customDictionaryCreate](#) or [customDictionaryExport](#).

See also [addWord](#).

**NOTE:** Internally the Spell Check object scans the user dictionary and removes the previously added word if it is there. Otherwise the word is added to the user's "Not-A-Word" dictionary. The actual dictionary is not modified.

**NOTE:** (Adobe Reader ) For Adobe Reader, this method is available only for Adobe Reader 7.0 or later.

### Parameters

<b>cWord</b>	The word to remove.
<b>cName</b>	(optional) The dictionary name or language code. An array of currently installed dictionaries can be obtained using <a href="#">dictionaryNames</a> or <a href="#">languages</a> .


### Returns

**true** if successful, **false** otherwise

## userWords

5.0				
-----	--	--	--	--

Gets the array of words a user has added to or removed from a dictionary. See also [addWord](#) and [checkWord](#).

**NOTE:** (Adobe Reader ) For Adobe Reader, this method is available only for Adobe Reader 7.0 or later.

### Parameters

<b>cName</b>	(optional) The dictionary name or language code. An array of currently installed dictionaries can be obtained using <a href="#">dictionaryNames</a> or <a href="#">languages</a> . If <b>cName</b> is not specified, the current default dictionary will be used. The current default dictionary is the first dictionary specified in the <b>Spelling</b> preferences dialog.
<b>bAdded</b>	(optional) When <b>true</b> , return the user's array of added words. When <b>false</b> , return the user's array of removed words. The default is <b>true</b> .

### Returns

The user's array of added or removed words.

### Example

List the words added to the "JavaScript" dictionary. (See the example that follows the description of [customDictionaryCreate](#).)

```
var aUserWords = spell.userWords({cName: "JavaScript"});
aUserWords.toSource();
```

---

## Statement Object

5.0				
-----	--	--	---	--

Use **statement** objects to execute SQL updates and queries, and retrieve the results of these operations. To create a statement object, use **connection.newStatement**.

See also:

- The [Connection Object](#).
- The [ADBC Object](#).
- [Column Generic Object](#), [ColumnInfo Generic Object](#), [Row Generic Object](#), [TableInfo Generic Object](#)

---

## Statement Properties

### columnCount

The number of columns in each row of results returned by a query. It is undefined in the case of an update operation.

*Type: Number*

*Access: R.*

### rowCount

The number of rows affected by an update. It is *not* the number of rows returned by a query. Its value is undefined in the context of a query.

*Type: Number*

*Access: R.*

---

## Statement Methods

### execute

Executes an SQL statement through the context of the Statement object. On failure, **execute** throws an exception.

**NOTE:** There is no guarantee that a client can do anything on a statement if an execute has neither failed nor returned all of its data.

**Parameters**


---

<b>cSQL</b>	The SQL statment to execute.
-------------	------------------------------

---

**Returns**

Nothing

**Example**

```
statement.execute("Select * from ClientData");
```

If the name of the database table or column name contains spaces, they need to be enclosed in escaped quotes. For example:

```
var execStr1 = "Select firstname, lastname, ssn from \"Employee Info\"";
var execStr2 = "Select \"First Name\" from \"Client Data\"";
statement.execute(execStr1);
statement.execute(execStr2);
```

A cleaner solution would be to enclose the whole SQL string with single quotes, then table names and column names can be enclosed with double quotes.

```
var execStr3 = 'Select "First Name","Second Name" from "Client Data" ';
statement.execute(execStr3);
```

See [getRow](#) and [nextRow](#) for extensive examples.

**getColumn**

Obtains a **column** object representing the data in the specified column.

**NOTE:** Once a column is retrieved with one of these methods, future calls attempting to retrieve the same column may fail.

**Parameters**


---

<b>nColumn</b>	The column from which to get the data. May be a column number or a string, the name of the desired column (see the <a href="#">ColumnInfo Generic Object</a> ).
<b>nDesiredType</b>	(optional) Which of the <a href="#">ADBC JavaScript Types</a> best represents the data in the column.

---

**Returns**

A [Column Generic Object](#) representing the data in the specified column, or **null** on failure.

## getColumnArray

Obtains an array of **column** objects, one for each column in the result set. A “best guess” is used to decide which of the [ADBC JavaScript Types](#) best represents the data in the column.

**NOTE:** Once a column is retrieved with one of these methods, future calls attempting to retrieve the same column may fail.

### Parameters

None

### Returns

An array of **column** objects, or **null** on failure as well as a zero-length array.

## getRow

Obtains a [Row Generic Object](#) representing the current row. This object contains information from each column. As for [getColumnArray](#), column data is captured in the “best guess” format.

A call to [nextRow](#) should precede a call to [getRow](#). Calling [getRow](#) twice, without an intervening call to [nextRow](#) yields a **null** return value for the second call to [getRow](#).

### Parameters

None

### Returns

A [Row Generic Object](#).

### Example 1

Every **Row** object contains a property for each column in a row of data. Consider the following example:

```
var execStr = "SELECT firstname, lastname, ssn FROM \"Employee Info\"";
statement.execute(execStr);
statement.nextRow();
row = statement.getRow();
console.println("The first name of the first person retrieved is: "
+ row.firstname.value);
console.println("The last name of the first person retrieved is: "
+ row.lastname.value);
console.println("The ssn of the first person retrieved is: "
+ row.ssn.value);
```

### Example 2

If the column name contains spaces, then the above syntax for accessing the row properties (for example,, **row.firstname.value**) does not work. Alternatively,

```
Connect = ADBC.newConnection("Test Database");
statement = Connect.newStatement();
```

```
var execStr = 'Select "First Name","Second Name" from "Client Data" ';
statement.execute(execStr);
statement.nextRow();

// Populate this PDF file
this.getField("name.first").value = row["First Name"].value;
this.getField("name.last").value = row["Second Name"].value;
```

## nextRow

Obtains data about the next row of data generated by a previously executed query. This must be called following a call to **execute** to acquire the first row of results.

### Parameters

None

### Returns

Nothing. Throws an exception on **failure** (if, for example, there is no next row).

### Example

The following example is a rough outline of how to create a series of buttons and Document Level JavaScripts to browse a database and populate a PDF form.

For the **getNextRow** button, defined below, the **nextRow()** is used to retrieve the next row from the database, unless there is an exception thrown (indicating that there is no next row), in which case, we reconnect to the database, and use **nextRow()** to retrieve the first row of data (again).

```
/* Button Script */
// getConnected button
if (getConnected())
    populateForm(statement.getRow());

// a getNextRow button
try {
    statement.nextRow();
} catch(e) {
    getConnected();
}
var row = statement.getRow();
populateForm(row);

/* Document Level JavaScript */
// getConnected() Doc Level JS
function getConnected()
{
    try {
        ConnectADBCdemo = ADBC.newConnection("ADBCdemo");
        if (ConnectADBCdemo == null)
            throw "Could not connect";
    }
}
```



```

statement = ConnectADBCdemo.newStatement();
if (statement == null)
    throw "Could not execute newStatement";
if (statement.execute("Select * from ClientData"))
    throw "Could not execute the requested SQL";
if (statement.nextRow())
    throw "Could not obtain next row";
return true;
} catch(e) {
    app.alert(e);
    return false;
}
}
// populateForm()
/* Maps the row data from the database, to a corresponding text field
in the PDF file. */
function populateForm(row)
{
    this.getField("firstname").value = row.FirstName.value;
    this.getField("lastname").value = row.LastName.value;
    this.getField("address").value = row.Address.value;
    this.getField("city").value = row.City.value;
    this.getField("state").value = row.State.value;
    this.getField("zip").value = row.Zipcode.value;
    this.getField("telephone").value = row.Telephone.value;
    this.getField("income").value = row.Income.value;
}

```

---

## TableInfo Generic Object

This generic JS object contains basic information about a table, and is returned by `connection.getTableList`. It contains the following properties.

Property	Type	Access	Description
<b>name</b>	String	R	The identifying name of a table. This string could be used in SQL statements to identify the table that the <b>tableInfo</b> object is associated with.
<b>description</b>	String	R	A string that contains database-dependent information about the table.

---

## Template Object

Template objects are named pages within the document. These pages may be hidden or visible and can be copied or *spawned*. They are typically used to dynamically create content (for example, to add pages to an invoice on overflow).

See also the [Doc Object `templates`](#) property, and methods [`createTemplate`](#), [`getTemplate`](#), and [`removeTemplate`](#).

---

## Template Properties

### hidden

5.0	Ⓓ		Ⓕ	
-----	---	--	---	--

Determines whether the template is hidden. Hidden templates cannot be seen by the user until they are spawned or are made visible. When an invisible template is made visible it is appended to the document.

**NOTE:** Setting this property in Adobe Reader (prior to version 5.1) generates an exception. For Adobe Reader 5.1 and 6.0, setting this property depends on Advanced Forms Feature document rights. For Adobe Reader 7.0, it is not possible to set this property under any circumstances.

*Type: Boolean*

*Access: R/W.*

### name

5.0			
-----	--	--	--

The name of the template which was supplied when the template was created.

*Type: String*

*Access: R.*

---

## Template Methods

### spawn

5.0	Ⓓ		Ⓕ	
-----	---	--	---	--

Creates a new page in the document based on the template.

## Parameters

<b>nPage</b>	(optional) The 0-based index of the page number after which or on which the new page will be created, depending on the value of <b>bOverlay</b> . The default is 0.
<b>bRename</b>	(optional) Whether form fields on the page should be renamed. The default is <b>true</b> .
<b>bOverlay</b>	(optional) When <b>true</b> (the default), the template is overlaid on the specified page. When <b>false</b> , it is inserted as a new page before the specified page. To append a page to the document, set <b>bOverlay</b> to <b>false</b> and set <b>nPage</b> to the number of pages in the document. <b>NOTE:</b> For certified documents, or documents with “Advanced Form Features rights” (F), the <b>bOverlay</b> parameter is disabled; this means that a template cannot be overlaid for these types of documents.
<b>oXObject</b>	(optional, version 6.0) The value of this parameter is the return value of an earlier call to <b>spawn</b> .

## Returns

Prior to Acrobat 6.0, this method returned nothing. Now, **spawn** returns an object representing the page contents of the page spawned. This return object can then be used as the value of the optional parameter **oXObject** for subsequent calls to **spawn**.

**NOTE:** Repeatedly spawning the *same* page can cause a large inflation in the file size. To avoid this file size inflation problem, **spawn** now returns an object that represents the page contents of the spawned page. This return value can be used as the value of the **oXObject** parameter in subsequent calls to the **spawn** method to spawn the same page.

### Example 1

This example spawns all templates and appends them one by one to the end of the document.

```
var a = this.templates;
for (i = 0; i < a.length; i++)
    a[i].spawn(this.numPages, false, false);
```

### Example 2 (version 6.0)

The following example spawns the same template 31 times using the **oXObject** parameter and return value. Using this technique avoids overly inflating the file size.

```
var t = this.templates;
var T = t[0];
var XO = T.spawn(this.numPages, false, false);
for (var i=0; i<30; i++) T.spawn(this.numPages, false, false, XO);
```

---

## Thermometer Object

6.0			
-----	--	--	--

This object is a combined status window/progress bar that indicates to the user that a lengthy operation is in progress. To acquire a **thermometer** object, use **app.thermometer**.

### Example

The following is a general example that illustrates how to use all properties and methods of the **thermometer** object.

```
var t = app.thermometer;           // acquire a thermometer object
t.duration = this.numPages;
t.begin();
for ( var i = 0; i < this.numPages; i++)
{
    t.value = i;
    t.text = "Processing page " + (i + 1);
    if (t.cancelled) break;        // break if operation cancelled
    ... process the page ...
}
t.end();
```

---

## Thermometer Properties

### cancelled

Whether the user wants to cancel the current operation. The user can indicate to the script the desire to terminate the operation by pressing the escape key on the Windows and Unix platforms and command-period on the Macintosh platform.

*Type: Boolean*

*Access: R.*

### duration

Sets the value that corresponds to a full thermometer display. The thermometer is subsequently filled in by setting its **value**. The default duration is 100.

*Type: Number*

*Access: R/W.*

### value

Sets the current value of the thermometer and updates the display. The allowed value ranges from 0 (empty) to the value set in the **duration**. For example, if the thermometer's

duration is 10, the current value must be between 0 and 10, inclusive. If value is less than zero, it is set to zero. If value is greater than duration, it is set to duration.

*Type: Number*

*Access: R/W.*

## text

Sets the text string that is displayed by the thermometer.

*Type: String*

*Access: R/W.*

---

## Thermometer Methods

### begin

Initializes the thermometer and displays it with the current value as a percentage of the duration.

#### Parameters

None

#### Returns

Nothing

#### Example

Count words on each page of current document, report running total and use thermometer to track progress.

```
var t = app.thermometer; // acquire a thermometer object
t.duration = this.numPages;
t.begin();
var cnt=0;
for ( var i = 0; i < this.numPages; i++)
{
    t.value = i;
    t.text = "Processing page " + (i + 1);
    cnt += getPageNumWords(i);
    console.println("There are " + cnt + "words in this doc.");
    if (t.cancelled) break;
}
t.end();
```

### end

Draws the thermometer with its current value set to the thermometer's duration (a full thermometer), then removes the thermometer from the display.

**Parameters**

None

**Returns**

Nothing

---

**TTS Object**

4.05			
------	--	--	--

The JavaScript **TTS** object allows users to transform text into speech. To be able to use the **TTS** object, the user's machine must have a Text-To-Speech engine installed on it. The Text-To-Speech engine will render text as digital audio and then "speak it". It has been implemented mostly with accessibility in mind but it could potentially have many other applications, bringing to life PDF documents.

This is currently a Windows-only feature and requires that the Microsoft Text to Speech engine be installed in the operating system.

The **TTS** object is present on both the Windows and Mac platforms (since it is a JavaScript object); however, it is disabled on the Mac.

**NOTE:** Acrobat 5.0 has taken a very different approach to providing accessibility for disabled users by integrating directly with popular screen readers. Some of the features and methods defined in 4.05 for the TTS object have been deprecated as a result as they conflict with the screen reader. The TTS object remains, however, as it still has useful functionality in its own right that might be popular for multi-media documents.

---

**TTS Properties****available**

**true** if the TTS object is available and the Text-To-Speech engine can be used.

Type: *Boolean*

Access: *R*.

**Example**

```
console.println("Text to speech available: " + tts.available);
```

**numSpeakers**

The number of different speakers available to the current text to speech engine. See also the [speaker](#) and the [getNthSpeakerName](#).

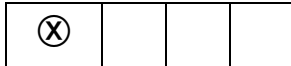
*Type: Integer**Access: R.*

## pitch

Sets the baseline pitch for the voice of a speaker. The valid range for pitch is from 0 to 10, with 5 being the default for the mode.

*Type: Integer**Access: R/W.*

## soundCues



Deprecated. Now returns only **false**.

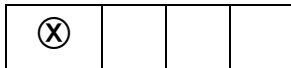
*Type: Boolean**Access: R/W.*

## speaker

Allows users to specify different speakers with different tone qualities when performing text-to-speech. See also the [numSpeakers](#) and the [getNthSpeakerName](#).

*Type: String**Access: R/W.*

## speechCues



Deprecated. Now returns only **false**.

*Type: Boolean**Access: R/W.*

## speechRate

Sets the speed at which text will be spoken by the Text-To-Speech engine. The value for **speechRate** is expressed in number of words per minute.

*Type: Integer**Access: R/W.*

## volume

Sets the volume for the speech. Valid values are from 0 (mute) to 10 (loudest).

*Type: Integer**Access: R/W.*

---

## TTS Methods

### getNthSpeakerName

Gets the *n*th speaker name in the currently installed text to speech engine (see also [numSpeakers](#) and [speaker](#)).

#### Parameters

---

<b>nIndex</b>	The index of the desired speaker name.
---------------	--

---

#### Returns

The name of the specified speaker.

#### Example

Enumerate through all of the speakers available.

```
for (var i = 0; i < tts.numSpeakers; i++) {
    var cSpeaker = tts.getNthSpeakerName(i);
    console.println("Speaker[" + i + "] = " + cSpeaker);
    tts.speaker = cSpeaker;
    tts.qText ("Hello");
    tts.talk();
}
```

### pause

Immediately pauses text-to-speech output on a TTS object. Playback of the remaining queued text can be resumed via [resume](#).

#### Parameters

None

#### Returns

Nothing



## qSilence

Queues a period of silence into the text.

### Parameters

---

<b>nDuration</b>	The amount of silence in milliseconds.
------------------	--

---

### Returns

Nothing

## qSound

Puts the specified sound into the queue in order to be performed by [talk](#). It accepts one parameter, **cSound**, from a list of possible sound cue names. These names map directly to sound files stored in the SoundCues folder, if it exists.

```
tts.qSound("DocPrint"); // Plays DocPrint.wav
```

The SoundCues folder should exist at the program level for the viewer, for example, C:\Program Files\Adobe\Acrobat 5.0\SoundCues.

**NOTE:** Windows only—**qSound** can handle only 22KHz, 16 bit PCM .wav files. These should be at least one second long in order to avoid a queue delay problem in MS SAPI. In case the sound lasts less than one second, it should be edited and have a silence added to the end of it.

### Parameters

---

<b>cSound</b>	The sound cue name to use.
---------------	----------------------------

---

### Returns

Nothing

## qText

Puts text into the queue in order to be performed by [talk](#).

### Parameters

---

<b>cText</b>	The text to convert to speech.
--------------	--------------------------------

---

### Returns

Nothing

### Example

```
tts.qText("Hello, how are you?");
```

## reset

Stops playback of current queued text and flushes the queue. Playback of text cannot be resumed via [resume](#). Additionally, it resets all the properties of the TTS object to their default values.

### Parameters

None

### Returns

Nothing

## resume

Resumes playback of text on a paused TTS object.

### Parameters

None

### Returns

Nothing

## stop

Stops playback of current queued text and flushes the queue. Playback of text cannot be resumed with [resume](#).

### Parameters

None

### Returns

Nothing

## talk

Sends whatever is in the queue to be spoken by the Text-To-Speech engine. If text output had been paused, [talk](#) resumes playback of the queued text.

### Parameters

None

### Returns

Nothing

### Example

```
tts.qText("Hello there!");  
tts.talk();
```

---

## this Object

In JavaScript the special keyword **this** refers to the current object. In Acrobat the current object is defined as follows:

- In an object method, it is the object to which the method belongs.
- In a constructor function, it is the object being constructed.
- In a function defined in one of the Folder Level JavaScripts files, it is undefined. It is recommended that calling functions pass the document object to any function at this level that needs it.
- In a Document level script or Field level script it is the document object and therefore can be used to set or get document properties and functions.

For example, assume that the following function was defined at the Plug-in folder level:

```
function PrintPageNum(doc)  
{  
    /* Print the current page number to the console. */  
    console.println("Page = " + doc.page);  
}
```

The following script outputs the current page number to the console (twice) and then prints the page:

```
/* Must pass the document object. */  
PrintPageNum(this);  
/* Same as the previous call. */  
console.println("Page = " + this.pageNum);  
/* Prints the current page. */  
this.print(false, this.pageNum, this.pageNum);
```

## Variable and Function Name Conflicts

Variables and functions that are defined in scripts are parented off of the *this* object. For example:

```
var f = this.getField("Hello");
```

is equivalent to

```
this.f = this.getField("Hello");
```

with the exception that the variable *f* can be garbage collected at any time after the script is run.

Acrobat JavaScript programmers should avoid using property and method names from the [Doc Object](#) as variable names. Use of method names when after the reserved word "var" will throw an exception, as the following line illustrates:

```
var getField = 1; // TypeError: redeclaration of function getField
```

Use of property names will not throw an exception, but the value of the property may not be altered if the property refers to an object:

```
// "title" will return "1", but the document will now be named "1".
var title = 1;
```

```
// property not altered, info still an object
var info = 1; // "info" will return [object Info]
```

The following is an example of avoiding variable name clash.

```
var f = this.getField("mySignature"); // uses the ppklite sig handler

// use "Info" rather than "info" to avoid a clash
var Info = f.signatureInfo();

// some standard signatureInfo properties
console.println("name = " + Info.name);
```

## Util Object

A static JavaScript object that defines a number of utility methods and convenience functions for string and date formatting and parsing.

## Util Methods

### iconStreamFromIcon

7.0				
-----	--	--	--	--

This method converts a XObject based [Icon Generic Object](#) into an [Icon Stream Generic Object](#).

#### Parameters

<b>oIcon</b>	An Icon Generic Object to be converted into an Icon Stream Generic Object.
--------------	--

#### Returns

[Icon Stream Generic Object](#)

It allows an icon obtained via `Doc.importIcon` or from `doc.getIcon` to be used in a function like `app.addToolButton`, which would otherwise accept only a [Icon Stream Generic Object](#) as input parameter.

### Example

Import an icon to the document level named icons tree and add a toolButton to the app.

```
this.importIcon("myIcon", "/C/temp/myIcon.jpg", 0);
var oIcon = util.iconStreamFromIcon(this.getIcon("myIcon"));
app.addToolButton({
  cName: "myButton",
  oIcon: myIcon,
  cExec: "console.println('My Button!');",
  cTooltext: "My button!",
  nPos: 0
});
```

## printf

3.01			
------	--	--	--

Formats one or more values as a string according to a format string. This is similar to the C function of the same name. This method converts and formats incoming arguments into a result string according to a format string (**cFormat**).

The format string consists of two types of objects:

- Ordinary characters, which are copied to the result string
- Conversion specifications, each of which causes conversion and formatting of the next successive argument to **printf()**.

Each conversion specification is constructed as follows:

```
%[,nDecSep] [cFlags] [nWidth] [.nPrecision] cConvChar
```

The following table describes the components of a conversion specification.

<b>nDecSep</b>	<p>Preceded by a comma character (,), is a digit from 0 to 3 which indicates the decimal/separator format:</p> <ul style="list-style-type: none"> <li>● 0 - comma separated, period decimal point.</li> <li>● 1 - no separator, period decimal point.</li> <li>● 2 - period separated, comma decimal point.</li> <li>● 3 - no separator, comma decimal point.</li> </ul>
----------------	--

---

<b>cFlags</b>	Only valid for numeric conversions and consists of a number of characters (in any order), which will modify the specification: <ul style="list-style-type: none"><li>● <b>+</b> - specifies that the number will always be formatted with a sign.</li><li>● <b>space</b> - if the first character is not a sign, a space will be prefixed.</li><li>● <b>0</b> - specifies padding to the field with leading zeros.</li><li>● <b>#</b> - which specifies an alternate output form. For <i>f</i> the output will always have a decimal point.</li></ul>
<b>nWidth</b>	A number specifying a minimum field width. The converted argument will be formatted in so that it is at least this many characters wide, including the sign and decimal point, and may be wider if necessary. If the converted argument has fewer characters than the field width it will be padded on the left to make up the field width. The padding character is normally a space, but is 0 if zero padding flag is present.
<b>nPrecision</b>	A number, preceded by a period character (.), which specifies the number of digits after the decimal point for float conversions.
<b>cConvChar</b>	One of: <ul style="list-style-type: none"><li>● <b>d</b> - integer, interpret the argument as an integer (truncating if necessary).</li><li>● <b>f</b> - float, interpret the argument as a number.</li><li>● <b>s</b> - string, interpret the argument as a string.</li><li>● <b>x</b> - hexadecimal, interpret the argument as an integer (truncating if necessary) and format in unsigned hexadecimal notation.</li></ul>

---

## Parameters

---

<b>cFormat</b>	The format string to use.
----------------	---------------------------

---

## Returns

A result string (**cResult**) formatted as specified.

## Example

```
var n = Math.PI * 100;
console.clear();
console.show();
console.println(util.printf("Decimal format: %d", n));
console.println(util.printf("Hex format: %x", n));
console.println(util.printf("Float format: %.2f", n));
console.println(util.printf("String format: %s", n));
```

## Output

```
Decimal format: 314
Hex format: 13A
Float format: 314.16
String format: 314.159265358979
```

## printd

3.01			
------	--	--	--

The **printd** method returns a date using the format specified by the **cFormat** parameter.

### Parameters

<b>cFormat</b>	<p>A string that represents the date and time format desired. This string is a pattern of supported substrings that are place-holders for date and time data. Recognized date and time strings are given in the table <a href="#">cFormat String Patterns with bXFAPicture set to false</a>.</p> <ul style="list-style-type: none"> <li>• (version 5.0) Beginning with version 5.0, <b>cFormat</b> can also be a number. See the table <a href="#">Quick formats with bXFAPicture set to false</a> for supported number values for <b>cFormat</b>.</li> <li>• (version 7.0) When <b>bXFAPicture</b> is <b>true</b>, the parameter <b>cFormat</b> is interpreted using the XFA Picture Clause format.</li> </ul>
<b>oDate</b>	<p><b>oDate</b> is a date object to format.</p> <p>A date object can be obtained from the <b>Date ()</b> constructor of core JavaScript, or from the <b>util.scand</b> method.</p>
<b>bXFAPicture</b>	<p>(optional, version 7.0) A boolean, which if <b>true</b>, the value of the <b>cFormat</b> parameter is interpreted using the XFA Picture Clause format, which gives extensive support for localized times and dates. See <a href="#">XFA Picture Clauses with bXFAPicture set to true</a> for additional discussion.</p> <p>The default is <b>false</b>.</p>

### Returns

The formatted date string.

### cFormat String Patterns with bXFAPicture set to false.

String	Effect	Example	Version
mmmm	Long month	September	
mmm	Abbreviated month	Sept	
mm	Numeric month with leading zero	09	
m	Numeric month without leading zero	9	
dddd	Long day	Wednesday	
ddd	Abbreviated day	Wed	
dd	Numeric date with leading zero	03	

String	Effect	Example	Version
d	Numeric date without leading zero	3	
yyyy	Long year	1997	
yy	Abbreviate Year	97	
HH	24 hour time with leading zero	09	
H	24 hour time without leading zero	9	
hh	12 hour time with leading zero	09	
h	12 hour time without leading zero	9	
MM	minutes with leading zero	08	
M	minutes without leading zero	8	
ss	seconds with leading zero	05	
s	seconds without leading zero	5	
tt	am/pm indication	am	
t	single digit am/pm indication	a	
j	Japanese Emperor Year (abbreviated) <b>NOTE:</b> (version 7.0) This format string has been deprecated. Use the XFA Picture Clause format.		6.0
jj	Japanese Emperor Year <b>NOTE:</b> (version 7.0) This format string has been deprecated. Use the XFA Picture Clause format.		6.0
\	use as an escape character		



**Quick formats with `bXFAPicture` set to false**

Value	Description	Example	Version
0	PDF date format	D:20000801145605+07'00'	5.0
1	Universal	2000.08.01 14:56:05 +07'00'	5.0
2	Localized string	2000/08/01 14:56:05	5.0

**XFA Picture Clauses with `bXFAPicture` set to true**

The section devoted to date and time pictures in the documents *XFA-Picture Clause 2.0 Specification* and *XFA-Picture Clause Version 2.2 – CCJK Addendum* provide the documentation for the strings that make up the `cFormat` parameter of `util.printd` in the case the `bXFAPicture` parameter is set to `true`.

See the [Adobe Web Documentation](#) section for a link to these two documents.

**Example 1**

To format the current date in long format, for example, you would use the following script:

```
var d = new Date();
console.println("Today is " + util.printd("mmm dd, yyyy", d));
```

**Example 2 (Version 5.0)**

```
// display date in a local format
console.println(util.printd(2, new Date() ));
```

**Example 3 (Version 7.0)**

This example illustrates the XFA-Picture Clause.

```
// execute in console
console.println(
    util.printd("EEE, 'the' D 'of' MMMM, YYYY", new Date(), true));
// the output on this day is
Tue, the 13 of July, 2004
```

Locale-Sensitive Picture Clauses. Normally processing of picture clauses occurs in the ambient locale. It is possible however to indicate that picture processing be done in a specific locale. This is of use when formatting or parsing data that is locale-specific and different from the ambient locale. The syntax for this extension to compound picture clauses is:

```
category-name(locale-name){picture-symbols}
```

The code executed in the console,

```
util.printd("date(fr){DD MMMM, YYYY}", new Date(), true)
```

yields the output on this day,

```
13 juillet, 2004
```

The XFA-Picture Clause gives extensive support for Chinese, Chinese (Taiwan), Japanese, and Korean (CCJK) times and dates. The example below, a custom format script of a text field, gives the current date formatted for a Japanese locale.

```
event.value = util.printd("date(ja){ggYY/M/D}", new Date(), true)
```

## printx

3.01			
------	--	--	--

Formats a source string, **cSource**, according to a formatting string, **cFormat**. A valid format for **cFormat** is any string which may contain special masking characters:

Value	Effect
?	Copy next character.
X	Copy next alphanumeric character, skipping any others.
A	Copy next alpha character, skipping any others.
9	Copy next numeric character, skipping any others.
*	Copy the rest of the source string from this point on.
\	Escape character.
>	Uppercase translation until further notice.
<	Lowercase translation until further notice.
=	Preserve case until further notice (default).

### Parameters

<b>cFormat</b>	The formatting string to use.
<b>cSource</b>	The source string to use.

### Returns

The formatted string.

### Example

To format a string as a U.S. telephone number, for example, use the following script:

```
var v = "aaa14159697489zzz";
v = util.printx("9 (999) 999-9999", v);
console.println(v);
```

## scand

4.0			
-----	--	--	--

Converts the supplied date, **cDate**, into a JavaScript date object according to rules of the supplied format string, **cFormat**. This routine is much more flexible than using the date constructor directly.

**NOTE:** Given a two digit year for input, **scand** resolves the ambiguity as follows: if the year is less than 50 then it is assumed to be in the 21st century (that is, add 2000), if it is greater than or equal to 50 then it is in the 20th century (add 1900). This heuristic is often known as the *Date Horizon*.

The supplied date **cDate** should be in the same format as described by **cFormat**.

### Parameters

<b>cFormat</b>	The rules to use for formatting the date. <b>cFormat</b> uses the same syntax as found in <a href="#">printd</a> .
<b>cDate</b>	The date to convert.

### Returns

The converted **date** object, or **null** if the conversion fails.

### Example 1

```
/* Turn the current date into a string. */
var cDate = util.printd("mm/dd/yyyy", new Date());
console.println("Today's date: " + cDate);
/* Parse it back into a date. */
var d = util.scand("mm/dd/yyyy", cDate);
/* Output it in reverse order. */
console.println("Yet again: " + util.printd("yyyy mmmm dd", d));
```

### Example 2

The method will return **null** if the conversions fails, this can occur if the user inputs a data different than what is expected. In this case, simple test the return value for **null**.

```
var d= util.scand("mm/dd/yyyy", this.getField("myDate").value);
if ( d== null )
    app.alert("Please enter a valid date of the form" +
        " \"mm/dd/yyyy\".")
else {
    console.println("You entered the date: "
        + util.printd("mmmm dd, yyyy",d));
}
```

## spansToXML

6.0			
-----	--	--	--

This method converts an array of [Span Objects](#) into an XML(XFA) String as described in the PDF 1.5 Specification.

### Parameters

An array of <a href="#">Span Objects</a>	An array of span objects to be converted into an XML string.
--	--

### Returns

String

### Example

This example gets the value of a rich text field, turns all of the text blue, converts it to an XML string and then prints it to the console

```
var f = getField("Text1");
var spans = f.richValue;
for(var index = 0; index < spans.length; index++)
    spans[index].textColor = color.blue;
console.println(util.spansToXML(spans));
```

## streamFromString

7.0			
-----	--	--	--

This function converts a string to a [ReadStream Object](#).

### Parameters

<b>cString</b>	The string to be converted into a <a href="#">ReadStream Object</a> .
<b>cCharset</b>	(optional) The encoding for the string in <b>cString</b> . The options are: utf-8, utf-16, Shift-JIS, BigFive, GBK, UHC. The default is utf-8.

### Returns

String

### Example

This example takes the reponse given in a text field of this document, and appends this response to an attached document.

```
var v = this.getField("myTextField").value;
var oFile = this.getDataObjectContents("MyNotes.txt");
```

```
var cFile = util.stringFromStream(oFile, "utf-8");
cFile += "\r\n" + cFile;
oFile = util.streamFromString( cFile, "utf-8");
this.setDataObjectContents("MyNotes.txt", oFile);
```

This example uses [Doc.getDataObjectContents](#), [util.stringFromStream](#) and [Doc.setDataObjectContents](#).

## stringFromStream

7.0			
-----	--	--	--

This function converts a [ReadStream Object](#) to a string.

### Parameters

<b>oStream</b>	<a href="#">ReadStream Object</a> to be converted into a string.
<b>cCharSet</b>	(optional) The encoding for the string in <b>oStream</b> . The options are: utf-8, utf-16, Shift-JIS, BigFive, GBK, UHC. The default is utf-8.

### Returns

[ReadStream Object](#)

### Example

Assume there is a text file embedded in this document. This example reads the contents of the attachment, and displays it in the multiline text field.

```
var oFile = this.getDataObjectContents("MyNotes.txt");
var cFile = util.stringFromStream(oFile, "utf-8");
this.getField("myTextField").value = cFile;
```

This example uses [getDataObjectContents](#) to get the file stream of the attached document.

## xmlToSpans

6.0			
-----	--	--	--

This method converts an XML(XFA) String as described in the PDF 1.5 Specification to an array of span objects suitable for specifying as the richValue or richContents of a field or annotation.

**Parameters**


---

a string	An XML (XFA) string to be converted to an array of <a href="#">Span Objects</a> .
----------	---

---

**Returns**

An Array of [Span Objects](#)

**Example**

Get the rich text string from "Text1", convert it to XML, then convert back again to an array of span objects and repopulate the text field.

```
var f = getField("Text1");
var spans = f.richValue;
var str = util.spansToXML(spans);
var spans = util.xmlToSpans(str);
f.richValue = spans;
```

**XFAObject Object**

6.0.2			
-------	--	--	--

The XFAObject object corresponds to the appModel in the XFA Scripting reference. All the XFA documents are located at <http://partners.adobe.com/asn/tech/pdf/xmlformspec.jsp>.

An XFAObject Object is returned by the **XMLData.parse** and **XMLData.applyXPath** methods.

**Example**

The following code detects whether the PDF document was created by Adobe Designer and has XML forms, or has Acrobat forms.

```
if ( typeof xfa == "object" ) {
    if ( this.dynamicXFAForm ) {
        console.println("This is a dynamic XML form.");
    }
    else {
        console.println("This is a static XML form.");
    }
}
else console.println("This is an Acrobat Form.");
```

**XMLData Object**

XMLData is a static object that allows the creation of a JavaScript object representing an XML document tree, and permits the manipulation of arbitrary XML documents via the XFA

Data Dom. In XFA, there are several other Dom's parallel to the Data Dom, but for the purpose of the XMLData Object only the Data Dom is used.

PDF documents which return `true` to the `doc.dynamicXFAForm` property can use the XMLData object, but cannot have its form fields manipulated by that object, as the two data DOMs are isolated from each other.

---

## XMLData Object Methods

### applyXPath

7.0			
-----	--	--	--

The **applyXPath** method permits the manipulation and query of an XML document via XPath expressions. XPath expressions evaluate to one of the known four types: Boolean, Number, String, Node-set. In JavaScript, they are returned, respectively, as the following types: Boolean, Number, String, Object.

If an object is returned, this object is of type **XFAObject** (see the [XFAObject Object](#)), which represents either a tree started by a single node, or by a list of nodes (a tree list). The type of this object is the same as the one returned by the **XMLData.parse**.

See the [References](#) section for the link to the W3C document, *XML Path Language (XPath)*, for the details of the XPath language.

**NOTE:** XFA provides a type of query mechanism, the SOM expressions, similar to that of XPath. XPath is widely used in the XML community, we provide the extra **applyXPath** method so that users can chose what query mechanism to use.

#### Parameters

<b>oXml</b>	An XFAObject object representing an XML document tree. <b>NOTE:</b> An exception is thrown when the value of this parameter is a nodeList XFA object instead the root of an XML tree, as required.
<b>cXPath</b>	A string parameter with the XPATH query to be performed on the document tree.

#### Returns

Boolean, Number, String, or XFAObject.

#### Example

Consider the following XML data string, the family tree of the "Robot" family. The method **XMLData.applyXPath** can return a boolean, a number, a string or a XFAObject. This example illustrates each of these return types, in the process of extracting information from the given set of XML data.

```

var cXMLDoc = "<family name = 'Robat'>\
  <grandad id = 'm1' name = 'A.C.' gender='M'>\
    <child> m2 </child>\
    <personal>\
      <income>100000</income>\
    </personal>\
  </grandad>\
  <dad id = 'm2' name = 'Bob' gender='M'>\
    <parent> m1 </parent>\
    <spouse> m3 </spouse>\
    <child> m4 </child>\
    <child> m5 </child>\
    <child> m6 </child>\
    <personal>\
      <income>75000</income>\
    </personal>\
  </dad>\
  <mom id = 'm3' name = 'Mary' gender='F'>\
    <spouse> m2 </spouse>\
    <personal>\
      <income>25000</income>\
    </personal>\
  </mom>\
  <daugther id = 'm4' name = 'Sue' gender='F'>\
    <parent> m2 </parent>\
    <personal>\
      <income>40000</income>\
    </personal>\
  </daugther>\
  <son id = 'm5' name = 'Jim' gender='M'>\
    <parent> m2 </parent>\
    <personal>\
      <income>35000</income>\
    </personal>\
  </son>\
  <daughter id = 'm6' name = 'Megan' gender='F'>\
    <parent> m2 </parent>\
    <personal>\
      <income>30000</income>\
    </personal>\
  </daughter>\
</family>";
var myXML= XMLData.parse( cXMLDoc, false);

```

The following line illustrates a return value of an XFAObject.

#### Get mom's data

```

var a = XMLData.applyXPath(myXML, "//family/mom")
a.saveXML('pretty');
<?xml version="1.0" encoding="UTF-8"?>
<mom id="m3" name="Mary" gender="F">
  <spouse> m2 </spouse>

```



```

    <personal>
      <income>25000</income>
    </personal>
  </mom>
  // get the income element value
  a.personal.income.value = "20000"; // change the income

```

Get dad's name, an attribute.

```

var b = XMLData.applyXPath(myXML, "//family/dad/@name");
b.saveXML('pretty');
<?xml version="1.0" encoding="UTF-8"?>
  name="Bob"
// assign to a variable
var dadsName = b.value; // dadsName = "Bob"

```

Get all attributes of dad node.

```

var b = XMLData.applyXPath( myXML, "//family/dad/attribute::*" );
for(var i=0; i < b.length; i++)
  console.println(b.item(i).saveXML('pretty'))

```

The loop above outputs the following to the console.

```

<?xml version="1.0" encoding="UTF-8"?>
  id="m2"
<?xml version="1.0" encoding="UTF-8"?>
  name="Bob"
<?xml version="1.0" encoding="UTF-8"?>
  gender="M"

```

Extract particular information from this we have,

```

console.println("For attribute 2, we have " + b.item(2).name + " = '"
  + b.item(2).value + "'.");

```

which yields an output of

```

For attribute 2, we have gender = 'M'.

```

Get dad's second child.

```

var c = XMLData.applyXPath(myXML, "//family/dad/child[position()=2]");
c.saveXML('pretty')
<?xml version="1.0" encoding="UTF-8"?>
<child> m5 </child>

```

This is the **id** of dad's second child. In the examples below, we get the family data on this child.

The following illustrates a return value of string.

```

// calculate the value of dadsName using XPath methods.
var dadsName = XMLData.applyXPath(myXML, "string(//family/dad/@name)");
// dadsName is assigned a value of "Bob" with this one line.

```

Get the family info on dad's second child. The line that follows assigns **c = "m5"**, the return value of this call to **applyXPath** is a string. The function **normalize-space** converts its argument to a string and removes surrounding spaces.

```

var c = XMLData.applyXPath(myXML,
    "normalize-space(//family/dad/child[2])");
var d = "//*[@id = '\" + c + '\"]"; // Note: d= "//*[@id = 'm5']"
XMLData.applyXPath(myXML, d ).saveXML('pretty'); // show what we have
<son id="m5" name="Jim" gender="M">
  <parent> m2 </parent>
  <personal>
    <income>35000</income>
  </personal>
</son>

```

Now get the 6th child node of the family root, and display some info on same. The XPath functions **name ()** and **concat ()** are used.

```

var e = XMLData.applyXPath(myXML,
    "concat(name(//family/child::*[position()=6]), '=',
    //family/child::*[position()=6]/@name)");
console.println(e); // the output is "daughter=Megan"

```

Get the names of all members of the the "Robot" family.

```

e = XMLData.applyXPath(myXML, "//family/child:.*" );
for ( var i = 1; i <= e.length; i++ ) {
  var str = "string(//family/child:.*["+i+"]/@name)";
  console.println(XMLData.applyXPath(myXML, str));
}

```

the output is

```

A.C.
Bob
Mary
Sue
Jim
Megan

```

The following illustrates a return value of a boolean type.

```

var f = XMLData.applyXPath( myXML, "//family/dad/@id = 'm2'");
if ( f == true ) console.println("dad's id is 'm2'");
else console.println("dad's id is not 'm2'");

```

The following lines of code illustrate a return value of number.

```

// get dad's income
g = XMLData.applyXPath( myXML, "number(//family/dad/personal/income)");
// double dad's salary, implied conversion to a number type
console.println("Dad's double salary is " +
    XMLData.applyXPath( myXML, "//family/dad/personal/income * 2" ));

```

Now compute the total income of the family "Robot".

```

console.println("Total income of A.C. Robot's family is "
    + XMLData.applyXPath( myXML, "sum(//income)" ) + ".");

```

The above line write the following to the console.

```

Total income of A.C. Robot's family is 305000.

```

List the individual incomes.

```
var g = XMLData.applyXPath( myXML, "//income")
for ( var i =0; i< g.length; i++) console.println(g.item(i).value);
```

## parse

7.0			
-----	--	--	--

The **parse** method creates an object representing an XML document tree. The parameters to **XMLData.parse** are the same as the parameters to the **loadXML** method present in the XFA Data Dom.

The object of type XFAObject (see the [XFAObject Object](#)), returned by **parse**, represents either a tree headed by a single node, or a tree started by a list of nodes (a tree list).

### Parameters

<b>param1</b>	A string containing the XML document.
<b>param2</b>	(optional) A boolean which, if <b>true</b> , the root node of the XML document should be ignored. If <b>false</b> , the root node should not be ignored. The default value is <b>true</b> .

### Returns

XFAObject

### Example 1

Consider the XML document as first introduced in the example following the **XMLData.applyXPath** method.

```
var x = XMLData.parse( cXMLDoc, false );
var y = x.family.name;           // a XFAObject
console.println(y.value);       // output to console is "Robot"
```

Get info about **dad**

```
y = x.family.dad.id;           // a XFAObject
console.println(y.value);     // output to console is "m2"

y = x.family.dad.name.value;   // y = "Bob"
x.family.dad.name.value = "Robert"; // change name to "Robert"
y = x.family.dad.name.value;   // y = "Robert"

y = x.family.dad.personal.income.value; // y = "75000"
x.family.dad.personal.income.value = "80000"; // give dad a raise
```

**Example 2**

A create a simple XML document and manipulate it.

```
x = XMLData.parse("<a> <c>A.</c><d>C.</d> </a>", false);
x.saveXML("pretty");
```

The output of the previous line is

```
<?xml version="1.0" encoding="UTF-8"?>
<xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <a>
    <c>A.</c>
    <d>C.</d>
  </a>
</xfa:data>
```

Now create another simple document.

```
y = XMLData.parse("<b>Robat</b>", false);
y.saveXML("pretty");
```

The output of this line is

```
<?xml version="1.0" encoding="UTF-8"?>
<xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <b>Robat</b>
</xfa:data>
```

Append **y** onto **x**

```
x.nodes.append(y.clone(true).nodes.item(0));
x.saveXML("pretty");
```

The result is

```
<?xml version="1.0" encoding="UTF-8"?>
<xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <a>
    <c>A.</c>
    <d>C.</d>
  </a>
  <b>Robat</b>
</xfa:data>
```

Now execute

```
x.nodes.insert(y.clone(true).nodes.item(0), x.nodes.item(0));
x.saveXML("pretty")
```

to obtain

```
<?xml version="1.0" encoding="UTF-8"?>
<xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <b>Robat</b>
  <a>
    <c>A.</c>
    <d>C.</d>
  </a>
```

```

    <b>Robat</b>
  </xfa:data>

```

Now remove these two nodes.

```

x.nodes.remove( x.nodes.namedItem("b") );
x.nodes.remove( x.nodes.namedItem("b") );

```

Now, we are back to the original XML document.

```

<?xml version="1.0" encoding="UTF-8"?>
<xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <a>
    <c>A.</c>
    <d>C.</d>
  </a>
</xfa:data>

```

Try executing the following line

```

x.a.nodes.insert( y.clone(true).nodes.item(0), x.a.nodes.item(0));
x.saveXML("pretty");

```

yields the following output:

```

<?xml version="1.0" encoding="UTF-8"?>
<xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <a>
    <b>Robat</b>
    <c>A.</c>
    <d>C.</d>
  </a>
</xfa:data>

```

Now remove that node just inserted:

```

x.a.nodes.remove( x.a.nodes.namedItem("b"));

```

Now insert **y**, actually a clone of **y**, between first and second children of the element **a**

```

x.a.nodes.insert( y.clone(true).nodes.item(0), x.a.nodes.item(1));

```

This produces the following

```

<?xml version="1.0" encoding="UTF-8"?>
<xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <a>
    <c>A.</c>
    <b>Robat</b>
    <d>C.</d>
  </a>
</xfa:data>

```

Remove that node just inserted:

```

x.a.nodes.remove( x.a.nodes.namedItem("b"));

```

Finally, append **y** onto **a**

```

x.a.nodes.append( y.clone(true).nodes.item(0));

```

### yielding

```
<?xml version="1.0" encoding="UTF-8"?>
<xfa:data xmlns:xfa="http://www.xfa.org/schema/xfa-data/1.0/">
  <a>
    <c>A.</c>
    <d>C.</d>
    <b>Robat</b>
  </a>
</xfa:data>
```

# New Features and Changes

This section summarizes the new features and changes introduced in Acrobat 7.0 and prior.

---

## Acrobat 7.0 Changes

The “Acrobat Multimedia JavaScript Reference”, which appeared as a separate document in version 6.0.2, has been merged into the “Acrobat JavaScript Scripting Reference”. See the section [“Introduced in Acrobat 6.0.2” on page 662](#) for a listing of all Multimedia JavaScript objects, properties and methods.

Execution of JavaScript through a menu event is no longer privileged. For details see the paragraph [JavaScript Execution through the Menu](#) on page 680 for detailed discussion.

There is now support for executing privileged code in a non-privileged context. See [Privileged versus Non-privileged Context](#) on page 679 for details.

The App Folder JavaScript files are now pre-compiled to improve performance, see [Bytecode to Improve Performance](#).

The Adobe Reader now has a console window. There is a preference under **Edit > Preferences > General > JavaScript** to “Show console on errors and messages”. In addition to errors and exceptions, the console can also be opened programmatically with `console.show()`. See the [Console Object](#) for a few other details.

The debugging capability of the JavaScript Debugging window can be made available for Adobe Reader on Windows and Macintosh platforms. In order to debug within Adobe Reader, the JavaScript file `debugger.js` needs to be installed, and the windows registry needs to be edited appropriately. See the [Acrobat JavaScript Scripting Guide](#) for the technical details.

## Introduced in Acrobat 7.0

The following properties and methods are introduced in Acrobat 7:

---

[Alerter Object](#)

methods:

[dispatch](#)

---

---

**Annot Object**

properties:

- `callout`
- `caretSymbola`
- `creationDatea`
- `dashb`
- `delayb`
- `doCaption`
- `intent`
- `leaderExtend`
- `leaderLength`
- `lineEnding`
- `opacityb`
- `refType`
- `richDefaultsa`
- `seqNumb`
- `statea`
- `stateModela`
- `style`
- `subjecta`

---

**Annot3D Object**

properties:

- `activated`
- `context3D`
- `innerRect`
- `name`
- `page`
- `rect`

---

**App Object**

properties:

- `constants`

methods:

- `beginPriv`
- `browseForDoc`
- `endPriv`
- `execDialog`
- `launchURL`
- `trustedFunction`
- `trustPropagatorFunction`

---

**Dialog Object**

methods:

- `enable`
- `end`
- `load`
- `store`

---



---

Doc Object	properties: <code>docID<sup>a</sup></code> <code>dynamicXFAForm</code> <code>external</code> <code>hidden</code> <code>mouseX</code> <code>mouseY</code> <code>noautocomplete</code> <code>nocache</code> <code>requiresFullSave</code> methods: <code>addWatermarkFromFile</code> <code>addWatermarkFromText</code> <code>embedDocAsDataObject</code> <code>encryptUsingPolicy</code> <code>getAnnot3D</code> <code>getAnnots3D</code> <code>getDataObjectContents</code> <code>getOCGOrder</code> <code>openDataObject</code> <code>removeScript</code> <code>setDataObjectContents</code> <code>setOCGOrder</code>
Doc.media Object	methods: <code>getOpenPlayers</code>
Field Object	methods: <code>signatureGetModifications</code>
OCG Object	properties: <code>constants</code> <code>initState</code> <code>locked</code> methods: <code>getIntent</code> <code>setIntent</code>
PlayerInfo Object	methods: <code>honors</code>

---

---

<code>printParams Object</code>	properties: <code>nUpAutoRotate</code> <code>nUpNumPagesH</code> <code>nUpNumPagesV</code> <code>nUpPageBorder</code> <code>nUpPageOrder</code>
<code>Search Object</code>	properties: <code>attachments</code> <code>ignoreAccents</code> <code>objectMetadata</code> <code>proximityRange</code>
<code>Security Object</code>	Security Constants methods: <code>chooseSecurityPolicy</code> <code>getSecurityPolicies</code>
<code>SecurityPolicy Object</code>	SecurityPolicy Properties
<code>SOAP Object</code>	methods: <code>queryServices</code> <code>resolveService</code> <code>streamDigest</code>
<code>Util Object</code>	methods: <code>iconStreamFromIcon</code> <code>streamFromString</code> <code>stringFromStream</code>
<code>XMLData Object</code>	methods: <code>applyXPath</code> <code>parse</code>

---

a. Present in version 6.0, documented in version 7.0.  
b. Present in version 5.0, documented in version 7.0.

---

**Modified in Acrobat 7.0****Changed or Enhanced Objects, Methods, and Properties**

The following properties and methods have been changed or enhanced:

App Object	methods: <a href="#">addToolButton</a> <a href="#">execMenuItem</a> <a href="#">getPath</a> <a href="#">mailGetAddrs</a> <a href="#">openDoc</a>
Console Object	The console window now available in Adobe Reader.
Doc Object	methods: <a href="#">createTemplate</a> <a href="#">mailDoc</a> <a href="#">print</a> <a href="#">saveAs</a> <a href="#">submitForm</a>
Field Object	methods: <a href="#">signatureSetSeedValue</a>
Index Object	methods: <a href="#">build</a>
OCG Object	properties: <a href="#">name</a>
printParams Object	properties: <a href="#">pageHandling</a>
Search Object	properties: <a href="#">indexes</a>
Security Object	properties: <a href="#">handlers</a> methods: <a href="#">getHandler</a>
SecurityHandler Object	properties: <a href="#">digitalIDs</a> methods: <a href="#">login</a> <a href="#">newUser</a>

SOAP Object	methods: <code>connect</code> <code>request</code>
Spell Object	The Spell object is not available in Adobe Reader 7.0 or higher methods: <code>addWord</code>
Util Object	methods: <code>printd</code>

## Acrobat 6.0 Changes

The notion of a [Safe Path](#) is introduced for this version of Acrobat. See the link provided for details.

### Introduced in Acrobat 6.0

The following properties and methods are introduced in Acrobat 6:

ADBC Object	<a href="#">SQL Types</a>
AlternatePresentation Object	properties: <code>active</code> <code>type</code> methods: <code>start</code> <code>stop</code>
Annot Object	properties: <code>borderEffectIntensity</code> <code>borderEffectStyle</code> <code>inReplyTo</code> <code>richContents</code> <code>toggleNoView</code> methods: <code>getStateInModel</code> <code>transitionToState</code>

---

App Object	properties: <b>fromPDFConverters</b> <b>printColorProfiles</b> <b>printerNames</b> <b>runtimeHighlight</b> <b>runtimeHighlightColor</b> <b>thermometer</b> <b>viewerType</b> methods: <b>addToolButton</b> <b>getPath</b> <b>mailGetAddrs</b> <b>newFDF</b> <b>openFDF</b> <b>popUpMenuEx</b> <b>removeToolButton</b>
Bookmark Object	methods: <b>setAction</b>
Catalog Object	properties: <b>isIdle</b> <b>jobs</b> methods: <b>getIndex</b> <b>remove</b>
Certificate Object	properties: <b>keyUsage</b> <b>usage</b>
Collab Object	methods: <b>addStateModel</b> <b>removeStateModel</b>
Connection Object	methods: <b>close</b>

---

---

Dbg Object	properties: <b>bps</b> methods: <b>c</b> <b>cb</b> <b>q</b> <b>sb</b> <b>si</b> <b>sn</b> <b>so</b> <b>sv</b>
Directory Object	properties: <b>info</b> methods: <b>connect</b>
DirConnection Object	properties: <b>canList</b> <b>canDoCustomSearch</b> <b>canDoCustomUISearch</b> <b>canDoStandardSearch</b> <b>groups</b> <b>name</b> <b>uiName</b> methods: <b>search</b> <b>setOutputFields</b>

---

---

Doc Object	properties: <b>alternatePresentations</b> <b>documentFileName</b> <b>metadata</b> <b>permStatusReady</b> methods: <b>addLink</b> <b>addRecipientListCryptFilter</b> <b>addScript</b> <b>encryptForRecipients</b> <b>exportAsText</b> <b>exportXFADData</b> <b>getLegalWarnings</b> <b>getLinks</b> <b>getOCGs</b> <b>getPrintParams</b> <b>importXFADData</b> <b>newPage</b> <b>removeLinks</b> <b>setAction</b> <b>setPageAction</b> <b>setPageTabOrder</b>
Error Objects	properties: <b>fileName</b> <b>lineNumber</b> <b>message</b> <b>name</b> methods: <b>toString</b>
Event Object	properties: <b>fieldFull</b> <b>richChange</b> <b>richChangeEx</b> <b>richValue</b>

---

---

**FDf Object**

properties:

**deleteOption**  
**isSigned**  
**numEmbeddedFiles**

methods:

**addContact**  
**addEmbeddedFile**  
**addRequest**  
**close**  
**mail**  
**save**  
**signatureClear**  
**signatureSign**  
**signatureValidate**

---

**Field Object**

properties:

**buttonFitBounds**  
**comb**  
**commitOnSelChange**  
**defaultStyle**  
**radiosInUnison**  
**richText**  
**richValue**  
**rotation**

methods:

**getLock**  
**setLock**  
**signatureGetSeedValue**  
**signatureSetSeedValue**

---

**Index Object**

methods:

**build**

---

**Link Object**

properties:

**borderColor**  
**borderWidth**  
**highlightMode**  
**rect**

methods:

**setAction**

---



---

OCG Object	properties: <b>name</b> <b>state</b> methods: <b>setAction</b>
printParams Object	properties: <b>binaryOK</b> <b>bitmapDPI</b> <b>colorOverride</b> <b>colorProfile</b> <b>constants</b> <b>downloadFarEastFonts</b> <b>fileName</b> <b>firstPage</b> <b>flags</b> <b>fontPolicy</b> <b>gradientDPI</b> <b>interactive</b> <b>lastPage</b> <b>pageHandling</b> <b>pageSubset</b> <b>printAsImage</b> <b>printContent</b> <b>printerName</b> <b>psLevel</b> <b>rasterFlags</b> <b>reversePages</b> <b>tileLabel</b> <b>tileMark</b> <b>tileOverlap</b> <b>tileScale</b> <b>transparencyLevel</b> <b>usePrinterCRD</b> <b>useT1Conversion</b>
Report Object	properties: <b>style</b>

---

---

Search Object	properties: docInfo docText docXMP bookmarks ignoreAsianCharacterWidth jpegExif legacySearch markup matchWholeWord wordMatching
Security Object	methods: chooseRecipientsDialog getSecurityPolicies importFromFile
SecurityHandler Object	properties: digitalIDs directories directoryHandlers signAuthor signFDF methods: newDirectory
SOAP Object	properties: wireDump methods: connect request response streamDecode streamEncode streamFromString stringFromStream

---

---

Span Object	properties: alignment fontFamily fontStretch fontStyle fontWeight text textColor textSize strikethrough subscript superscript underline
Spell Object	properties: languages languageOrder methods: customDictionaryClose customDictionaryCreate customDictionaryExport customDictionaryOpen ignoreAll
Thermometer Object	properties: cancelled duration value text methods: begin end
Util Object	methods: printd spansToXML xmlToSpans

---

**Modified in Acrobat 6.0****Changed or Enhanced Objects, Methods, and Properties**

The following properties and methods have been changed or enhanced:

---

App Object	methods: <code>addMenuItem</code> <code>alert</code> <code>listMenuItems</code> <code>listToolbarButtons</code> <code>response</code>
Doc Object	properties: <code>layout</code> <code>zoomType</code> methods: <code>createDataObject</code> <code>exportAsFDF</code> <code>exportAsXFDF</code> <code>exportDataObject</code> <code>flattenPages</code> <code>getField</code> (see <a href="#">Extended Methods</a> ) <code>getURL</code> <code>importDataObject</code> <code>importIcon</code> <code>print</code> <code>saveAs</code> <code>spawnPageFromTemplate</code> <code>submitForm</code>
Event Object	properties: <code>changeEx</code>
Field Object	properties: <code>name</code> methods: <code>buttonImportIcon</code> <code>signatureInfo</code> <code>signatureSign</code> <code>signatureValidate</code>

---

Global Object	Persistent global data only applies to variables of type Boolean, Number or String. Acrobat 6.0 has reduced the maximum size of global persistent variables from 32 k to 2-4 k. Any data added to the string after this limit is dropped.
Search Object	methods: <a href="#">query</a>
SecurityHandler Object	<p>The following were introduced in Acrobat 5.0 as properties and methods of the <b>PPKLite Signature Handler Object</b>. In Acrobat 6.0 they are properties and methods of the <a href="#">SecurityHandler Object</a>. All of these have new descriptions, and some have additional parameters.</p> <p><b>NOTE:</b> When signing using JavaScript methods, the user's digital signature profile must be a .pfx file, not an .apf, as in prior versions of Acrobat. To convert an .apf profile to the new .pfx type, use the UI (<b>Advanced &gt; Manage Digital IDs &gt; My Digital ID Files &gt; Select My Digital ID File</b>) to import the .apf profile.</p>
	<p>properties:</p> <ul style="list-style-type: none"> <li><a href="#">appearances</a></li> <li><a href="#">isLoggedIn</a></li> <li><a href="#">loginName</a></li> <li><a href="#">loginPath</a></li> <li><a href="#">name</a></li> <li><a href="#">signInvisible</a></li> <li><a href="#">signVisible</a></li> <li><a href="#">uiName</a></li> </ul> <p>methods:</p> <ul style="list-style-type: none"> <li><a href="#">login</a></li> <li><a href="#">logout</a></li> <li><a href="#">newUser</a></li> <li><a href="#">setPasswordTimeout</a></li> </ul>
Template Object	methods: <a href="#">spawn</a>

### Extended Methods

The `doc.getField` method has been extended in Acrobat 6.0 so that it retrieves the `field` object of individual widgets. See [Field Access from JavaScript](#) for a discussion of widgets and how to work with them.

## Deprecated in Acrobat 6.0

---

<a href="#">Search Object</a>	properties: <a href="#">soundex</a> <a href="#">thesaurus</a>
<a href="#">Spell Object</a>	methods: <a href="#">addDictionary</a> <a href="#">removeDictionary</a>

---

## Introduced in Acrobat 6.0.2

The following objects, properties and methods are introduced in Acrobat 6.0.2:

---

<a href="#">XFAObject Object</a>	
----------------------------------	--

---

The following table lists the objects, properties and methods of the Multimedia plugin. In Acrobat 6.0.2, multimedia JavaScript was documented in a separate document called the "Acrobat Multimedia JavaScript Reference".

---

<a href="#">App Object</a>	properties: <a href="#">media</a> <a href="#">monitors</a>
----------------------------	--

---

---

**App.media Object**

## properties:

`align`  
`canResize`  
`closeReason`  
`defaultVisible`  
`ifOffScreen`  
`layout`  
`monitorType`  
`openCode`  
`over`  
`pageEventNames`  
`raiseCode`  
`raiseSystem`  
`renditionType`  
`status`  
`trace`  
`version`  
`windowType`

## methods:

`addStockEvents`  
`alertFileNotFound`  
`alertSelectFailed`  
`argsDWIM`  
`canPlayOrAlert`  
`computeFloatWinRect`  
`constrainRectToScreen`  
`createPlayer`  
`getAltTextData`  
`getAltTextSettings`  
`getAnnotStockEvents`  
`getAnnotTraceEvents`  
`getPlayers`  
`getPlayerStockEvents`  
`getPlayerTraceEvents`  
`getRenditionSettings`  
`getURLData`  
`getURLSettings`  
`getWindowBorderSize`  
`openPlayer`  
`removeStockEvents`  
`startPlayer`

---

---

Doc Object	properties: <code>innerAppWindowRect</code> <code>innerDocWindowRect</code> <code>media</code> <code>outerAppWindowRect</code> <code>outerDocWindowRect</code> <code>pageWindowRect</code>
Doc.media Object	properties: <code>canPlay</code> methods: <code>deleteRendition</code> <code>getAnnot</code> <code>getAnnots</code> <code>getRendition</code> <code>newPlayer</code>
Event Object	A new Screen type used with Multimedia along with associated event names.
Events Object	methods: <code>add</code> <code>dispatch</code> <code>remove</code>

---



---

EventListener Object	methods: afterBlur afterClose afterDestroy afterDone afterError afterEscape afterEveryEvent afterFocus afterPause afterPlay afterReady afterScript afterSeek afterStatus afterStop onBlur onClose onDestroy onDone onError onEscape onEveryEvent onFocus onGetRect onPause onPlay onReady onScript onSeek onStatus onStop
Marker Object	properties: frame index name time
Markers Object	properties: player methods: get

---

---

MediaOffset Object	properties: <b>frame</b> <b>marker</b> <b>time</b>
MediaPlayer Object	properties: <b>annot</b> <b>defaultSize</b> <b>doc</b> <b>events</b> <b>hasFocus</b> <b>id</b> <b>innerRect</b> <b>iisOpen</b> <b>isPlaying</b> <b>markers</b> <b>outerRect</b> <b>page</b> <b>settings</b> <b>uiSize</b> <b>visible</b> methods: <b>close</b> <b>open</b> <b>pause</b> <b>play</b> <b>seek</b> <b>setFocus</b> <b>stop</b> <b>triggerGetRect</b> <b>where</b>
MediaReject Object	properties: <b>rendition</b>
MediaSelection Object	properties: <b>selectContext</b> <b>players</b> <b>rejects</b> <b>rendition</b>

---

---

**MediaSettings Object**

properties:

- autoPlay
- baseURL
- bgColor
- bgOpacity
- endAt
- data
- duration
- floating
- layout
- monitor
- monitorType
- page
- palindrome
- players
- rate
- repeat
- showUI
- startAt
- visible
- volume
- windowType

---

**Monitor Object**

properties:

- colorDepth
- isPrimary
- rect
- workRect

---

**Monitors Object**

methods:

- bestColor
- bestFit
- desktop
- document
- filter
- largest
- leastOverlap
- mostOverlap
- nonDocument
- primary
- secondary
- select
- tallest
- widest

---

---

PlayerInfo Object	properties: <b>id</b> <b>mimeTypes</b> <b>name</b> <b>version</b> methods: <b>canPlay</b> <b>canUseData</b>
PlayerInfoList Object	methods: <b>select</b>
Rendition Object	properties: <b>altText</b> <b>doc</b> <b>fileName</b> <b>type</b> <b>uiName</b> methods: <b>getPlaySettings</b> <b>select</b> <b>testCriteria</b>
ScreenAnnot Object	properties: <b>altText</b> <b>alwaysShowFocus</b> <b>display</b> <b>doc</b> <b>events</b> <b>extFocusRect</b> <b>innerDeviceRect</b> <b>noTrigger</b> <b>outerDeviceRect</b> <b>page</b> <b>player</b> <b>rect</b> methods: <b>hasFocus</b> <b>setFocus</b>

---

---

## Acrobat 5.0 Changes

### Introduced in Acrobat 5.0

---

<a href="#">ADBC Object</a>	methods: <a href="#">getDataSourceList</a> <a href="#">newConnection</a>
<a href="#">Annot Object</a>	properties: <a href="#">alignment</a> <a href="#">AP</a> <a href="#">arrowBegin</a> <a href="#">arrowEnd</a> <a href="#">author</a> <a href="#">contents</a> <a href="#">doc</a> <a href="#">fillColor</a> <a href="#">hidden</a> <a href="#">modDate</a> <a href="#">name</a> <a href="#">noView</a> <a href="#">page</a> <a href="#">point</a> <a href="#">points</a> <a href="#">popupRect</a> <a href="#">print</a> <a href="#">rect</a> <a href="#">readOnly</a> <a href="#">rotate</a> <a href="#">strokeColor</a> <a href="#">textFont</a> <a href="#">type</a> <a href="#">soundIcon</a> <a href="#">width</a> methods: <a href="#">destroy</a> <a href="#">getProps</a> <a href="#">setProps</a>

---

---

App Object

properties:

`activeDocs`  
`fs`  
`plugIns`  
`viewerVariation`

methods:

`addMenuItem`  
`addSubMenu`  
`clearInterval`  
`clearTimeout`  
`listMenuItems`  
`listToolbarButtons`  
`newDoc`  
`openDoc`  
`popupMenu`  
`setInterval`  
`setTimeout`

---

Bookmark Object

properties:

`children`  
`color`  
`doc`  
`name`  
`open`  
`parent`  
`style`

methods:

`createChild`  
`execute`  
`insertChild`  
`remove`

---

Color Object

methods:

`convert`  
`equal`

---

Connection Object

methods:

`newStatement`  
`getTableList`  
`getColumnList`

---

---

**Data Object**

properties:  
creationDate  
modDate  
MIMEType  
name  
path  
size

---

**Doc Object**

properties:  
bookmarkRoot  
disclosed (5.0.5)  
icons  
info  
layout  
securityHandler  
selectedAnnots  
sounds  
templates  
URL

methods:  
addAnnot  
addField  
addIcon  
addThumbnails  
addWeblinks  
bringToFront  
closeDoc  
createDataObject  
createTemplate  
deletePages  
deleteSound  
exportAsXFDF  
exportDataObject  
extractPages  
flattenPages  
getAnnot  
getAnnots  
getDataObject  
getIcon  
getPageBox  
getPageLabel

---

---

`getPageNthWord`  
`getPageNthWordQuads`  
`getPageRotation`  
`getPageTransition`  
`getSound`  
`importAnXFDF`  
`importDataObject`  
`importIcon`  
`importSound`  
`importTextData`  
`insertPages`  
`movePage`  
`print`  
`removeDataObject`  
`removeField`  
`removeIcon`  
`removeTemplate`  
`removeThumbnails`  
`removeWeblinks`  
`replacePages`  
`saveAs`  
`selectPageNthWord`  
`setPageBoxes`  
`setPageLabels`  
`setPageRotations`  
`setPageTransitions`  
`submitForm`  
`syncAnnotScan`

---

Event Object

properties:  
`changeEx`  
`keyDown`  
`targetName`

---



---

**Field Object**

## properties:

`buttonAlignX`  
`buttonAlignY`  
`buttonPosition`  
`buttonScaleHow`  
`buttonScaleWhen`  
`currentValueIndices`  
`doNotScroll`  
`doNotSpellCheck`  
`exportValues`  
`fileSelect`  
`multipleSelection`  
`rect`  
`strokeColor`  
`submitName`  
`valueAsString`

## methods:

`browseForFileToSubmit`  
`buttonGetCaption`  
`buttonGetIcon`  
`buttonSetCaption`  
`buttonSetIcon`  
`checkThisBox`  
`defaultIsChecked`  
`isBoxChecked`  
`isDefaultChecked`  
`setAction`  
`signatureInfo`  
`signatureSign`  
`signatureValidate`

---

**FullScreen Object**

## properties:

`backgroundColor`  
`clickAdvances`  
`cursor`  
`defaultTransition`  
`escapeExits`  
`isFullScreen`  
`loop`  
`timeDelay`  
`transitions`  
`usePageTiming`  
`useTimer`

---

Global Object	<p>methods:</p> <ul style="list-style-type: none"> <li><code>subscribe</code></li> </ul>
Identity Object	<p>properties:</p> <ul style="list-style-type: none"> <li><code>corporation</code></li> <li><code>email</code></li> <li><code>loginName</code></li> <li><code>name</code></li> </ul>
Index Object	<p>properties:</p> <ul style="list-style-type: none"> <li><code>available</code></li> <li><code>name</code></li> <li><code>path</code></li> <li><code>selected</code></li> </ul>
PlayerInfo Object	<p>properties:</p> <ul style="list-style-type: none"> <li><code>certified</code></li> <li><code>loaded</code></li> <li><code>name</code></li> <li><code>path</code></li> <li><code>version</code></li> </ul>
PPKLite Signature Handler Object (now listed under the <code>SecurityHandler</code> Object)	<p>properties</p> <ul style="list-style-type: none"> <li><code>appearances</code></li> <li><code>isLoggedIn</code></li> <li><code>loginName</code></li> <li><code>loginPath</code></li> <li><code>name</code></li> <li><code>signInvisible</code></li> <li><code>signVisible</code></li> <li><code>uiName</code></li> </ul> <p>methods:</p> <ul style="list-style-type: none"> <li><code>login</code></li> <li><code>logout</code></li> <li><code>newUser</code></li> <li><code>setPasswordTimeout</code></li> </ul>

---

Report Object	properties: <code>absIndent</code> <code>color</code> <code>absIndent</code> methods: <code>breakPage</code> <code>divide</code> <code>indent</code> <code>outdent</code> <code>open</code> <code>mail</code> <code>Report</code> <code>save</code> <code>writeText</code>
Search Object	properties: <code>available</code> <code>indexes</code> <code>markup</code> <code>maxDocs</code> <code>proximity</code> <code>refine</code> <code>soundex</code> <code>stem</code> methods: <code>addIndex</code> <code>getIndexForPath</code> <code>query</code> <code>removeIndex</code>
Security Object	properties: <code>handlers</code> <code>validateSignaturesOnOpen</code> methods: <code>getHandler</code>

---

Spell Object	properties: available dictionaryNames dictionaryOrder domainNames methods: addDictionary addWord check checkText checkWord removeDictionary removeWord userWords
Statement Object	properties: columnCount rowCount methods: execute getColumn getColumnArray getRow nextRow
Template Object	properties: hidden name methods: spawn

### Modified in Acrobat 5.0

- The console can act as an editor and can execute JavaScript code.
- The following properties and methods have been changed or enhanced:

App Object	language execMenuItem
Doc Object	exportAsFDF print submitForm
Event Object	type

---

Field Object	<code>textFont</code> <code>value</code> <code>buttonImportIcon</code> <code>getItemAt</code>
Util Object	<code>printd</code>

---

- The section related to [Event Object](#) has been greatly enhanced to facilitate better understanding of the Acrobat JavaScript Event model.

## Deprecated in Acrobat 5.0

The following properties and methods have been deprecated:

---

App Object	<code>fullscreen</code> <code>numPlugIns</code> <code>getNthPlugInName</code>
Doc Object	<code>author</code> <code>creationDate</code> <code>creationDate</code> <code>keywords</code> <code>modDate</code> <code>numTemplates</code> <code>producer</code> <code>title</code> <code>getNthTemplate</code> <code>spawnPageFromTemplate</code>
Field Object	<code>hidden</code>
TTS Object	<code>soundCues</code> <code>speechCues</code>

---

## Modified in Acrobat 5.05

- A new symbol has been added to the quick bar denoting which methods are missing from Acrobat™ Approval™.
- In the [Doc Object](#), the property `disclosed` has been added.

## Modified in Adobe 5.1 Reader

A new column has been added to the [Quick Bars](#) that summarize availability, and the meanings of the fourth and fifth columns has changed. They now indicate the availability of a property or method in the Adobe Reader and Acrobat Approval respectively.

- The symbols that appear in the fourth column indicate whether a property or method is available in Adobe Reader, and also whether access depends on document rights in the Acrobat 5.1 Reader.
- The fifth column indicates whether a property or method is available in Acrobat Approval.

Access to the following properties and methods has changed for the Adobe 5.1 Reader:

---

<a href="#">Annot Object</a>	properties:		
	<a href="#">alignment</a>	<a href="#">modDate</a>	<a href="#">rect</a>
	<a href="#">AP</a>	<a href="#">name</a>	<a href="#">readOnly</a>
	<a href="#">arrowBegin</a>	<a href="#">noView</a>	<a href="#">rotate</a>
	<a href="#">arrowEnd</a>	<a href="#">page</a>	<a href="#">strokeColor</a>
	<a href="#">author</a>	<a href="#">point</a>	<a href="#">textFont</a>
	<a href="#">contents</a>	<a href="#">points</a>	<a href="#">type</a>
	<a href="#">doc</a>	<a href="#">popupRect</a>	<a href="#">soundIcon</a>
	<a href="#">fillColor</a>	<a href="#">print</a>	<a href="#">width</a>
	<a href="#">hidden</a>		
	methods:		
	<a href="#">destroy</a>		
	<a href="#">getProps</a>		
	<a href="#">setProps</a>		
<hr/>			
<a href="#">Doc Object</a>	properties:		
	<a href="#">selectedAnnots</a>		
	methods:		
	<a href="#">addAnnot</a>	<a href="#">importAnXFDF</a>	
	<a href="#">addField</a>	<a href="#">importDataObject</a>	
	<a href="#">exportAsFDF</a>	<a href="#">mailDoc</a>	
	<a href="#">exportAsXFDF</a>	<a href="#">mailForm</a>	
	<a href="#">getAnnot</a>	<a href="#">spawnPageFromTemplate</a>	
	<a href="#">getAnnots</a>	<a href="#">submitForm</a>	
	<a href="#">getNthTemplate</a>	<a href="#">syncAnnotScan</a>	
	<a href="#">importAnFDF</a>		
<hr/>			
<a href="#">Template Object</a>	methods:		
	<a href="#">spawn</a>		

---



# Security and Technical Notes

In this section, security implementations to Acrobat are summarized, other technical notes are presented as well.

---

## Security Notes

Through the various iterations of the Acrobat software, changes have been made to improve the overall security profile of Acrobat. In this section, these various changes are cataloged.

### Safe Path

The following is an Acrobat 6.0 addition.

A security posture Acrobat has taken concerns all JavaScript methods that write data to the local hard drive based on a path passed to it by one of its parameters. All paths are required to a *safe path*: For windows, the path cannot point to a system critical folder, for example, a root, windows or system directory. However, this is not the only requirement for a path to be safe; a path is also subject to certain, unspecified tests as well.

For many of the methods in question, the file name must have an extension appropriate to the type of data that is to be saved; some methods may have a no-overwrite restriction. These additional restrictions are noted in the documentation.

Generally, when a path is judged to be “not safe”, a **NotAllowedError** (see the [Error Objects](#)) exception is thrown and the method fails.

### Privileged versus Non-privileged Context

In versions of Acrobat prior to 7.0, certain security restricted methods could only be executed in a *privileged context*<sup>1</sup>, typically, during a console, batch, menu or application initialization event. All other events (for example, page open and mouse up events) are considered *non-privileged*.

Beginning with Acrobat 7.0, menu events are no longer considered privileged, see the paragraph, “[JavaScript Execution through the Menu](#)”, below.

In Acrobat 7.0, the concept of a trusted function is introduced. Trusted functions allow privileged code—code that normally requires a privileged context to execute—to execute in *non-privileged* contexts. For details and examples, see `app.trustedFunction()`.

---

1. Beginning with version 6.0, there is an exception to this statement. Security restricted methods can execute in a non-privileged context *provided* the document is Certified by the document author for embedded JavaScript. See the description of [column 3](#) of the quick bar retarding security for additional details.

---

## Technical Notes

### JavaScript Execution through the Menu

Beginning with Acrobat 7.0, execution of JavaScript through a menu event is no longer privileged. To execute a security restricted method (Ⓢ) through a menu event, one of the following must be true:

1. Under **Edit > Preferences > General > JavaScript**, the item labeled "Enable menu items JavaScript execution privileges" must be checked.
2. The method must be executed through a trusted function. For details and examples, see `app.trustedFunction()`.

The paragraph titled [Privileged versus Non-privileged Context](#) also should be reviewed.

### Bytecode to Improve Performance

In versions of Acrobat previous to 7.0, the JavaScript files `AForm.js`, `ADBC.js`, `Annots.js`, `AnWizard.js`, `media.js`, and `SOAP.js` resided in the App JavaScript folder. Beginning with Acrobat 7.0, these files will not be shipped with Acrobat Professional, Acrobat Standard or Adobe Reader. In their place, a pre-compiled bytecode is used in order to improve performance. The `debugger.js` file in the App folder is not included in the bytecode.

Files in the User JavaScript folder will not be included in the pre-compiled bytecode file.

Acrobat prefers that users put their own `.JS` files in the User JavaScript folder, the same place where `glob.js` resides. JavaScript code that sets up menu-items (`addItem`) should be put in `config.js` in the User JavaScript folder. The location of this folder can be found programmatically by executing `app.getPath("user", "javascript")` from the console.