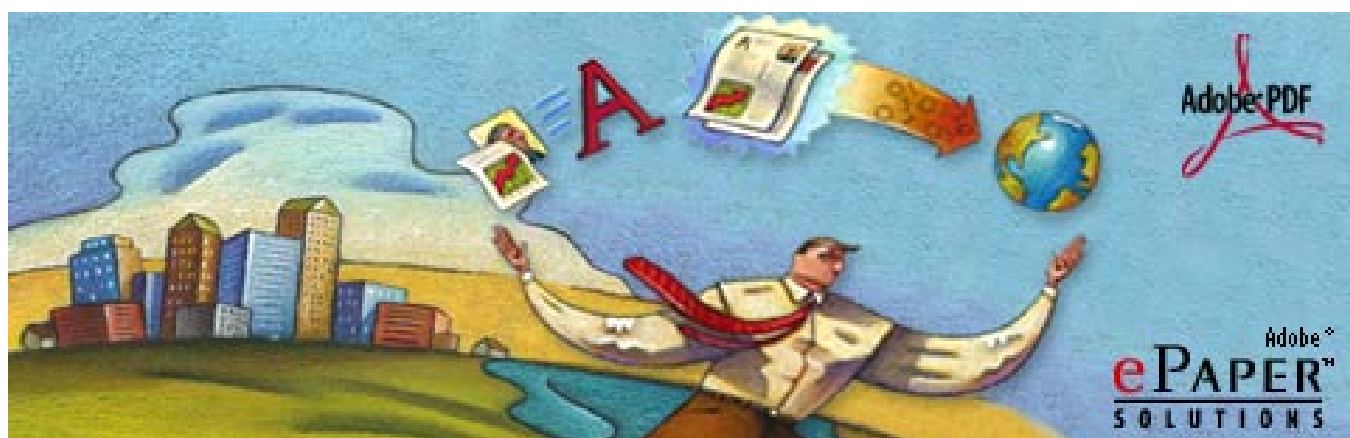




Adobe® Acrobat®



テクニカルノート # 5186

Acrobat JavaScript Object Specification

バージョン **5.0.5**

改訂日：2001 年 9 月 14 日

© 2001 Adobe Systems Incorporated. All rights reserved.

このドキュメントの記載内容は情報として使用するためにのみ提供されており、予告なしに変更されることがあります。また、それらについて Adobe Systems Incorporated（アドビシステムズ社）はいかなる責任も負いません。アドビシステムズ社は、このドキュメントに関して、誤りや不正確な記述があった場合にも、いかなる責任または義務も負いません。このドキュメントに記載しているソフトウェアはライセンス契約の下に提供されており、その条件に基づいた場合にのみ使用またはコピーが許可されます。

Adobe、Adobe ロゴ、Acrobat、Acrobat Capture、および Distiller は、アドビシステムズ社の商標です。Microsoft および Windows は Microsoft 社の米国およびその他の国における登録商標であり、ActiveX は同社の商標です。Macintosh は、Apple Computer, Inc. の米国およびその他の国における登録商標です。PowerPC は、International Business Machines Corporation の商標です。UNIX は、X/Open Co. Ltd. により特にライセンス許可された米国およびその他の国における登録商標です。その他のすべての製品名またはブランド名は、各所有者の商標です。

目次

目次	1
はじめに	15
Acrobat™ JavaScript によろこそ	15
JavaScript とは	15
Acrobat™ JavaScript とは	15
このマニュアルで使用する表記	16
ヒント	16
クイックバー	16
ヘルプが必要なときには	17
参考資料	18
Acrobat™ JavaScript のはじめに	20
JavaScript の使用場所	20
ドキュメント内の JavaScript	20
ドキュメント外の JavaScript	23
簡単な例	24
コア言語の特性	25
データ型	25
変数	27
未定義の変数	28
コメント	29
句読点	29
メソッドのパラメータ指定	30
メソッドのクイックヘルプ	31
例外処理	31
Acrobat 5.0 での JavaScript の編集	32
ドキュメント内のすべての JavaScript の編集	32
外部エディタ	33
内部エディタ内での Tab キーの使用	33
対話型 JavaScript コンソール	33
JavaScript の実行	33
信頼性の高いコードを作成するためのヒント	34
PDF フォームでの JavaScript の使用	36
コードの作成	36
フィールドの操作	37
相互依存するフィールドの構成	38
フォーマットスクリプトと検証スクリプト	38
拡張フォーマット	39
PDF と HTML の違い	40

5.0 の新機能	41
5.0 のその他の変更	44
5.05 の変更	44
ADBC オブジェクト	46
ADBC プロパティ	47
SQL 型	47
JavaScript 型	47
ADBC メソッド	48
getDataSourceList	48
newConnection	48
Annot オブジェクト	50
JavaScript による注釈へのアクセス	50
Annot プロパティ	51
alignment	51
AP	52
arrowBegin	52
arrowEnd	53
attachIcon	53
author	53
contents	53
doc	54
fillColor	54
gestures	55
hidden	55
modDate	55
name	55
notelcon	56
noView	56
page	56
point	56
points	57
popupOpen	57
popupRect	58
print	58
quads	58
rect	58
readOnly	59
rotate	59
strokeColor	59
textFont	59
textSize	60
type	61
soundIcon	61
width	61

Annot メソッド	61
destroy	61
getProps	62
setProps	62
App オブジェクト	64
App オブジェクトのプロパティ	64
activeDocs	64
calculate	64
focusRect	65
formsVersion	65
fs	65
fullscreen	65
language	66
numPlugIns	67
openInPlace	67
platform	67
plugIns	67
toolbar	68
toolbarHorizontal	68
toolbarVertical	68
viewerType	69
viewerVariation	69
viewerVersion	69
App オブジェクトのメソッド	69
addMenuItem	69
addSubMenu	70
alert	71
beep	72
clearInterval	72
clearTimeout	73
execMenuItem	73
getNthPlugInName	74
goBack	75
goForward	75
hideMenuItem	75
hideToolbarButton	75
listMenuItems	76
listToolbarButtons	76
mailMsg	77
newDoc	77
openDoc	78
popUpMenu	79
response	80
setInterval	80
setTimeout	81

Bookmark オブジェクト	83
Bookmark オブジェクトのプロパティ	83
children	83
color	83
doc	84
name	84
open	84
parent	84
style	85
Bookmark オブジェクトのメソッド	85
createChild	85
execute	86
insertChild	86
remove	86
 カラー配列	88
Color オブジェクト	88
Color オブジェクトのプロパティ	89
Color オブジェクトのメソッド	89
convert	89
equal	90
 Connection オブジェクト	91
Connection オブジェクトのメソッド	91
newStatement	91
getTableList	91
getColumnList	92
 Console オブジェクト	94
Console オブジェクトのメソッド	94
show	94
hide	94
println	94
clear	94
 Data オブジェクト	95
Data オブジェクトのプロパティ	95
creationDate	95
modDate	95
MIMETYPE	95
name	95
path	96
size	96
 Doc オブジェクト	97
JavaScript による Doc オブジェクトへのアクセス	97

Doc	オブジェクトのプロパティ	97
	author	97
	baseURL	98
	bookmarkRoot	98
	calculate	98
	creator	99
	creationDate	99
	dataObjects	99
	delay	99
	dirty	100
	disclosed	100
	external	100
	filesize	100
	icons	101
	info	101
	keywords	102
	layout	103
	modDate	103
	numFields	103
	numPages	103
	numTemplates	103
	path	104
	pageNum	104
	producer	104
	securityHandler	104
	selectedAnnots	105
	sounds	105
	spellDictionaryOrder	105
	subject	106
	templates	106
	title	106
	URL	107
	zoom	107
	zoomType	107
Doc	オブジェクトのメソッド	108
	addAnnot	108
	addField	108
	addIcon	110
	addThumbnails	110
	addWeblinks	110
	bringToFront	111
	calculateNow	111
	closeDoc	111
	createDataObject	112
	createTemplate	112
	deletePages	113

deleteSound	113
exportAsFDF	114
exportAsXFDF	115
exportDataObject	116
extractPages	116
flattenPages	117
getAnnot	118
getAnnots	118
getDataObject	120
getField	120
getIcon	121
getNthFieldName	121
getNthTemplate	121
getPageBox	122
getPageLabel	122
getPageNthWord	123
getPageNthWordQuads	123
getPageNumWords	124
getPageRotation	124
getPageTransition	125
getSound	125
getTemplate	126
getURL	126
gotoNamedDest	126
importAnFDF	127
importAnXFDF	127
importDataObject	128
importIcon	129
importSound	130
importTextData	131
insertPages	131
mailDoc	132
mailForm	133
movePage	133
print	134
removeDataObject	135
removeField	135
removeIcon	135
removeTemplate	135
removeThumbnails	136
removeWeblinks	136
replacePages	137
resetForm	137
saveAs	138
scroll	139
selectPageNthWord	139

setPageBoxes	139
setPageLabels.....	140
setPageRotations.....	141
setPageTransitions	141
spawnPageFromTemplate	142
submitForm	143
syncAnnotScan	145
Event オブジェクト	146
イベントのタイプ／名前の組み合わせ	146
App / Init	146
Batch / Exec	146
Bookmark / Mouse Up	146
Console / Exec	147
Doc / DidPrint	147
Doc / DidSave	147
Doc / Open	147
Doc / WillClose	148
Doc / WillPrint	148
Doc / WillSave	148
External / Exec	148
Field / Blur	149
Field / Calculate	149
Field / Focus	149
Field / Format	150
Field / Keystroke	150
Field / Mouse Down	151
Field / Mouse Enter	151
Field / Mouse Exit	151
Field / Mouse Up	151
Field / Validate	152
Link / Mouse Up	152
Menu / Exec	152
Page / Open	153
Page / Close	153
ドキュメントイベント処理	153
フォームイベント処理	154
Event オブジェクトのプロパティ	154
change	154
changeEx	154
commitKey.....	155
keyDown	155
modifier	156
name	156
rc	156
selEnd	156

selStart	156
shift	156
source	157
target	157
targetName	157
type	158
value	158
willCommit	159
Field オブジェクト	160
JavaScript によるフィールドへのアクセス	160
Field オブジェクトのプロパティ	160
alignment	160
borderStyle	160
buttonAlignX	161
buttonAlignY	161
buttonPosition	162
buttonScaleHow	162
buttonScaleWhen	163
calcOrderIndex	163
charLimit	163
currentValueIndices	164
defaultValue	165
doNotScroll	165
doNotSpellCheck	165
delay	165
display	166
doc	166
editable	167
exportValues	167
fileSelect	167
fillColor	168
hidden	168
highlight	169
lineWidth	169
multiline	170
multipleSelection	170
name	171
numItems	171
page	171
password	171
print	172
readonly	172
rect	172
required	173
strokeColor	173

style	174
submitName	174
textColor	174
textFont	175
textSize	176
type	176
userName	176
value	177
valueAsString	178
Field オブジェクトのメソッド	178
browseForFileToSubmit	178
buttonGetCaption	178
buttonGetIcon	179
buttonImportIcon	179
buttonSetCaption	180
buttonSetIcon	181
checkThisBox	182
clearItems	182
defaultIsChecked	182
deleteItemAt	183
getArray	183
getItemAt	184
insertItemAt	185
isBoxChecked	185
isDefaultChecked	186
setAction	186
setFocus	187
setItems	187
signatureInfo	188
signatureSign	190
signatureValidate	191
FullScreen オブジェクト	193
FullScreen オブジェクトのプロパティ	193
backgroundColor	193
clickAdvances	193
cursor	193
defaultTransition	194
escapeExits	194
isFullScreen	194
loop	194
timeDelay	195
transitions	195
usePageTiming	195
useTimer	196

Global オブジェクト	197
Global オブジェクトのプロパティ	197
Global オブジェクトのメソッド	197
setPersistent	197
subscribe	198
Identity オブジェクト	200
Identity オブジェクトのプロパティ	200
corporation	200
email	200
loginName	200
name	200
Index オブジェクト	201
Index オブジェクトのプロパティ	201
available.....	201
name	201
path	201
selected.....	201
PlugIn オブジェクト	202
PlugIn オブジェクトのプロパティ	202
certified	202
loaded	202
name	202
path	203
version	203
PPKLite 署名ハンドラオブジェクト	204
PPKLite オブジェクトのプロパティ	205
appearances	205
isLoggedIn	206
loginName	206
loginPath	206
name	206
signInvisible	207
signVisible	207
uiName	207
PPKLite オブジェクトのメソッド	207
login	207
logout	208
newUser	208
setPasswordTimeout	209
Report オブジェクト	210
Report オブジェクトのプロパティ	210

size	210
absIndent	210
color	210
Report オブジェクトのメソッド	211
breakPage	211
divide	211
indent	211
outdent	212
open	212
save	212
mail	213
Report	213
writeText	213
Search オブジェクト	215
Search オブジェクトのプロパティ	215
available	215
indexes	215
matchCase	215
maxDocs	216
proximity	216
refine	216
soundex	216
stem	216
thesaurus	217
Search オブジェクトのメソッド	217
addIndex	217
getIndexForPath	217
query	217
removeIndex	218
Security オブジェクト	219
Security オブジェクトのプロパティ	219
handlers	219
validateSignaturesOnOpen	219
Security オブジェクトのメソッド	219
getHandler	219
Sound オブジェクト	221
Sound オブジェクトのプロパティ	221
name	221
Sound オブジェクトのメソッド	221
play	221
pause	221
stop	221

Spell オブジェクト	222
Spell オブジェクトのプロパティ	222
available	222
dictionaryNames	222
dictionaryOrder	222
domainNames	223
Spell オブジェクトのメソッド	223
addDictionary	223
addWord	224
check	224
checkText	225
checkWord	225
removeDictionary	226
removeWord	227
userWords	227
Statement オブジェクト	228
Statement オブジェクトのプロパティ	228
columnCount	228
rowCount	228
Statement オブジェクトのメソッド	228
execute	228
getColumn	229
getColumnArray	230
getRow	230
nextRow	231
Template オブジェクト	233
Template オブジェクトのプロパティ	233
hidden	233
name	233
Template オブジェクトのメソッド	233
spawn	233
TTS オブジェクト	235
TTS オブジェクトのプロパティ	235
available	235
numSpeakers	235
pitch	235
soundCues	236
speaker	236
speechCues	236
speechRate	236
volume	236

TTS オブジェクトのメソッド	237
getNthSpeakerName	237
pause	237
qSilence	237
qSound	237
qText	238
reset	238
resume	238
stop	238
talk	239
this オブジェクト	240
変数名と関数名の衝突	240
Util オブジェクト	242
Util オブジェクトのメソッド	242
printf	242
printd	243
printx	245
scand	245
Acrobat JavaScript に関する簡単な FAQ	247
JavaScript はどこにあり、どのように使用するのですか？	247
フォルダレベルの JavaScript	247
文書レベル	248
フィールドレベル	248
フォームフィールドにはどのように名前を付けるのですか？	248
日付オブジェクトはどのように使用するのですか？	249
日付から文字列への変換	250
文字列から日付への変換	250
日付の演算	252
どうすればドキュメントを保護できますか？	253
ドキュメントに対するアクセス制限	253
権限の制限	253
電子署名	254
署名フィールドに署名した後、どうすればドキュメントを	
ロックできますか？	254
どうすればドキュメントへのアクセスを簡単にできますか？	255
ドキュメントのメタデータ	256
フィールドの説明	256
タブ順序の設定	256
TTS オブジェクトの使用	256
標準の動作	256
どうすれば JavaScript でグローバル変数を定義できますか？	257
グローバル変数の永続化	257
どうすればフォームデータを電子メールアドレスに送信できますか？	258

どうすれば他のフィールドの値に応じてフィールドを非表示に できますか？	258
どうすれば別に関われているフォームのフィールド値を現在のフォーム から参照できますか？	258
どうすればキー入力を 1 つずつインターセプトできますか？	259
どうすればネストされたポップアップメニューを作成できますか？	259
どうすれば独自の色を作成できますか？	259
どうすればユーザ入力を促すダイアログを表示できますか？	259
どうすれば JavaScript で URL を取り出せますか？	260
どうすればフィールドのホットヘルプテキストを動的に 変更できますか？	260
どうすればズーム倍率をプログラムで変更できますか？	260
どうすればマウスが特定の領域に入った（あるいは出た）かを 判断できますか？	260
回転ユーザスペースおよびデフォルトユーザスペースとは何ですか？	260
回転ユーザスペース	261
デフォルトユーザスペース	261
座標の関係	261
どうすればフォームフィールドをプログラムで作成できますか？	263
ボタン	263
チェックボックス	266
コンボボックス	267
リストボックス	271
ラジオボタン	272
署名	273
テキスト	275
クイックリファレンス：フォーム	277
表示方法：全フィールド	277
アクション：全フィールド	278
オプション：ボタン	279
オプション：チェックボックスとラジオボタン	280
オプション：コンボボックスとリストボックス	281
オプション：テキストフィールド	282
フォーマット：コンボボックスとテキスト	283
検証：コンボボックスとテキスト	284
計算：コンボボックスとテキスト	285
署名	286
選択の変更	287
どうすれば注釈をプログラムで作成できますか？	288
Circle および Square 注釈	289
Line 注釈	290
Stamp 注釈	291
FreeText 注釈	292
Text 注釈	293
Ink 注釈	294
Highlight、Strikeout、Underline、Squiggle	295

5.0.5 Acrobat JavaScript Object 仕様

はじめに

Acrobat™ JavaScript によるこそ

Acrobat 5.0 JavaScript リファレンスマニュアルによるこそ。以下のページには、PDF ドキュメントで JavaScript を使用するために必要な情報がすべて記載されています。Acrobat に実装されている強力な JavaScript の機能を活用することにより、フォームや他のドキュメントをカスタマイズして、その外観、機能性、対話性を大幅に向上させることができます。

このマニュアルは、Acrobat JavaScript のオブジェクト、プロパティ、メソッドのリファレンスとして使用できます。また、JavaScript プログラミングの基本や、Acrobat JavaScript のプロパティやメソッドを分かりやすく示す多くの例や、関連するプログラミング手法についても記載しています。

JavaScript とは

JavaScript は強力なオブジェクト指向のスクリプト言語で、Web ページの対話性を拡張するために Netscape Communications によって開発されました。JavaScript は、本来は Netscape ブラウザ用に設計されましたが、様々な用途で利用されるようになり、汎用的なプログラミング言語として急速に普及しました。現在では国際標準化機構の ISO-16262 規格として認められています。(欧州コンピュータ製造工業会 (ECMA) によって標準化された JavaScript の最初のバージョンは、ECMAScript として知られています)。JavaScript のコア言語にはたび重なる改訂が加えられ、最新バージョンは 1.5 です。Acrobat 5.0 ではこのバージョンが使用されています。

Acrobat™ JavaScript とは

コア JavaScript は、数多くの実装の様々なニーズに対応するため、拡張されてきました。クライアントサイド（ブラウザベース）の JavaScript では、ブラウザのウィンドウおよびそのコンテンツを操作するためのオブジェクトやメソッドが追加されました。サーバサイドの JavaScript では、データベースの照会や、その他一般的なサーバサイドの処理を行うため、File および Database オブジェクトが追加されました。そして Acrobat™ JavaScript では、PDF ドキュメントの内容に簡単にアクセス（および操作）できるようにするため、オブジェクトやメソッドが追加され、言語が拡張されました。このマニュアルでは、こうした操作を可能にするために言語に加えられた、さまざまな拡張機能について詳しく説明します。

ブラウザベースの JavaScript を扱い易くしている特徴（厳密でないデータ型、C に似た構文、組み込みの Math および String クラス）は、Acrobat™ JavaScript にも適用されます。実際、Acrobat™ JavaScript はコア JavaScript の上に構築されているので、Math、String、Date、Array、および RegExp（正規表現）クラスなど、ブラウザベースの JavaScript でおなじみのコア言語機能はすべて、Acrobat™ JavaScript でも使用できます。

ここではスペースの都合上、コア JavaScript 機能に関する詳しい説明は省略しています。このマニュアルでは、ブラウザ環境で一般的に使用されている JavaScript に関して、ある程度の知識があることを前提としています（コア言語について詳しくは、Web サイト

<http://developer.netscape.com/docs/manuals/index.html?content=javascript.html> をご覧いただくか、書店のコンピュータプログラミングのコーナーをお訪ねください）。

JavaScript プログラミングの経験が既にある方にとっても、ブラウザベースの JavaScript から Adobe Acrobat JavaScript へ移行する際、このマニュアルが役に立つでしょう。また、このマニュアルでは、PDF ドキュメントに関連する情報へアクセスする方法を学ぶことができます。例えば、コンピュータのプラットフォームに関する情報を取得したり、ドキュメントが Acrobat または Acrobat Reader のどちらで開かれているかという動作環境を判別したり、ズーム倍率やファイルサイズなどのプロパティ情報にアクセスすることもできます。また、組み込みのメソッドを使用してフォームのフィールド情報にアクセスしたり、フォームの内容（外観属性も含む）を実行時に変更する方法も習得できます。

このマニュアルで使用する表記

ヒント

このマニュアルでは、ヒントを次のようなボックスに表記します。

ヒント：ここに参考となるヒントが示されます。

このボックスに示されるコメントは参考にしていただきたい情報で、必ずしも読む必要はありません。しかしここには、コーディングの際にぶつかりそうな問題や状況について、役に立つ「現実的な」解決策が示されます。

クイックバー

ほとんどすべてのプロパティまたはメソッドの先頭には、次のように一目で分かるマークを「クイック」バーに表示しています。

例：



最初の列は、プロパティまたはメソッドを使用できるソフトウェアのバージョンを示します。このマニュアルは Acrobat バージョン 5.0 を対象としているので、旧バージョンとの互換性の問題を考慮しています。バージョン番号が明記されている場合、プロパティまたはメソッドの使用は、そのバージョン以降の Acrobat に限定されます。下位互換を必要とする場合、プ


ロパティまたはメソッドにアクセスする前に、明記されているバージョンよりもフォームのバージョンが大きいことをスクリプトでチェックする必要があります。


例：


```
if (typeof app.formsVersion != "undefined" && app.formsVersion >= 5.0) {  
    // Perform version specific operations.  
}
```


最初の列が空白の場合は、互換性のチェックは必要ありません。


Acrobat Forms 用に JavaScript が拡張されるにつれ、一部のプロパティやメソッドは、より柔軟性に富む適切なものに置き換えられてきました。これらの古いメソッドは、現在は使用するべきではないため、バージョン列に Mr. Unhappy ☹ が表示されます。

 2 つ目の列が空白でない場合、そのメソッドまたはプロパティの使用によって PDF ドキュメントが変更されることを意味します。ドキュメントを保存すると、メソッドによる変更も保存されます。

 2 つ目の列には設定マークが表示されることもあります。このマークは、プロパティによってドキュメントが変更されることは無いが、アプリケーション設定が永続的に変更される可能性があることを示します。

 3 つ目の列が空白でない場合は、このメソッドまたはプロパティがセキュリティ上の理由により、特定のイベント時（バッチイベント、アプリケーションの起動、コンソール内での実行など）にしか使用できないことを示します。Acrobat の各種イベントについては、[Event オブジェクト](#)の節を参照してください。

 4 つ目の列が空白でない場合は、このメソッドまたはプロパティが Acrobat Reader で使用できないことを示します。

 4 つ目の列には、メソッドが Acrobat Approval で使用できないことを示すマークが表示されることもあります。Acrobat Approval で使用できないメソッドは、Acrobat Reader でも使用できません。

ヘルプが必要なときには

PDF 用の JavaScript だけでなく、JavaScript 一般に役立つリソースとして、次のように数多くの Web サイトがあります。

- <http://www.adobe.com/support/forums/main.html>- アドビシステムズ社では、Acrobat や Acrobat Reader を含むすべての Adobe 製品専用のオンラインフォーラムを提供しています。
- <http://www.adobe.com/support/database.html>- アドビシステムズ社では、フォーラムに加えて、一般的な質問に回答する検索可能なサポートデータベースを公開しています。

- <http://www.pdfzone.com/resources/lists.html>-PDFZone の Web サイトでは、PDF フォーム、PDF 開発、Acrobat の使用、その他のトピックに関するディスカッションフォーラムを提供しています。このフォーラムには世界中からエキスパートが参加しており、ユーザの疑問に答えてくれます。
- <http://forum.planetpdf.com/>-PlanetPDF の Web サイトの中でも人気のあるこのサイトでは、初心者、開発者、フォームインテグレータ、その他を対象にしたディスカッションフォーラムを提供しています。PDFZone 同様、PlanetPDF Forum（旧名称は AcroBuddies）には世界各国からエキスパートが参加しています。
- PlanetPDF（<http://www.planetpdf.com>）には、JavaScript サンプルを専門に取り扱う [Planet PDF Developers](#) というセクションがあります。

コア JavaScript 言語に関する最終的な拠り所は、その作成元である Netscape Communications です。JavaScript に関する現在のマニュアルのリストについては、<http://developer.netscape.com/docs/manuals/index.html?content=javascript.html> を参照してください。

参考資料

コア JavaScript 1.4 のマニュアル

Acrobat 5.0 で使用されている JavaScript 1.5 の完全なマニュアルは、Web サイト <http://developer.netscape.com/docs/manuals/javascript.html> から入手できます。

『Core JavaScript Guide, Part I. Core Language Features』、Netscape Communications Corporation 著。このマニュアルの Part I では、コア JavaScript 言語の概念を記載しています。このマニュアルは、html および PDF のどちらのファイル形式でも入手できます。<http://developer.netscape.com/docs/manuals/js/core/jsguide15/contents.html> を参照してください。注意：このマニュアルの残りの部分は、コア JavaScript に対する Netscape の拡張機能を扱っており、Acrobat 環境には適用されません。

『Core JavaScript Reference, Part I. Object Reference and Part II. Language Elements』、Netscape Communications Corporation 著。Part I および II はコア JavaScript 言語のリファレンスで、html および PDF のどちらのファイル形式でも入手できます。<http://developer.netscape.com/docs/manuals/js/core/jsref15/contents.html> を参照してください。注意：このマニュアルの残りの部分は、コア JavaScript に対する Netscape の拡張機能を扱っており、Acrobat 環境には適用されません。

弊社の Web サイトから入手できるマニュアル

『PDF Reference, second edition, Adobe Portable Document Format, Version 1.3』、アドビシステムズ社著。2000 年 7 月 Addison-Wesley より出版。このマニュアルは書店でも購入でき

ますが、<http://partners.adobe.com/asn/developer/acrosdk/docs/PDFRef.pdf> から入手することもできます。

『*Technical Note #5190, Acrobat Viewer plug-in API Overview*』 Acrobat ビューアのプラグイン API に備わっているオブジェクトおよびメソッドの概要を記載しています。このマニュアルは、Adobe Acrobat プラグインの SDK CD-ROM または弊社の Web サイト

<http://partners.adobe.com/asn/developer/acrosdk/docs/apiovr.pdf> から入手できます。

『*Technical Note #5167, Acrobat Viewer plug-in API Development*』 さまざまなプラットフォームで Acrobat ビューアのプラグインを開発する方法について説明しています。このマニュアルは、Adobe Acrobat プラグインの SDK CD-ROM または弊社の Web サイト

<http://partners.adobe.com/asn/developer/acrosdk/docs/getstart/DevelopmentOverview.pdf> から入手できます。

『*Technical Note #5191, Acrobat Viewer plug-in API On-Line Reference*』 Acrobat ビューアのプラグイン API に備わっているオブジェクトおよびメソッドについて詳しく説明しています。このマニュアルは、Adobe Acrobat プラグインの SDK CD-ROM または弊社の Web サイト

<http://partners.adobe.com/asn/developer/acrosdk/docs/apiref.pdf> から入手できます。

弊社の CD で提供されるマニュアル

Acrobat CD で提供されるマニュアルおよびサンプルでは、JavaScript のさまざまな使用法を説明しています。

『*Batch Sequences*』：バッチシーケンスの詳細、および JavaScript を使用した一般的な問題の解決方法を記載しています。

『*Getting Started with ADBC*』：Acrobat Database Connectivity オブジェクトを介してデータベース情報にアクセスするための入門書で、メールをマージするバッチシーケンスのサンプルが含まれています。

『*Tutorial on form field authoring*』：フォームフィールドを作成および操作し、請求書を認証する手順をステップごとに説明しています。

『*Forms System Implementation Notes*』：フォームを送信する際の Acrobat と Web サーバとの間のインタフェースについて説明しています。

『*What's In A Name*』：フィールド名によって、フォームの認証および処理にどのような違いが生じるかを記載しています。

Acrobat™ JavaScript のはじめに

JavaScript プログラミングにより、PDF ドキュメントの作成に画期的で新しい展望が広がりました。JavaScript を使用すると、比較的小さな労力で、PDF ドキュメントを無数の方法で拡張することができます。例えば、JavaScript を使用して次のような処理を行うことができます。

- ・ メニューのコマンドをプログラムから実行する (Acrobat または Reader)。
- ・ システムのビープを鳴らしたり、警告を出す。
- ・ ユーザがフォームに入力する際に、その入力内容を決められた書式に変換する。
- ・ フォーム内で、互いに依存しあう複数のフィールドを構成する。
- ・ ユーザアクションに応じて、フォームフィールドの外観 (背景色およびテキストフォントを含む) を変更する。
- ・ ユーザがリンクまたはボタンをクリックした場合に、電子メールを自動送信する。
- ・ PDF ドキュメントあるいはフォーム内から応答ダイアログを表示し、ユーザ入力を求める。
- ・ その他

ここでは、JavaScript 言語の基本を説明し、コア言語を拡張した PDF 関連機能のいくつかを紹介します。その過程で、コードの信頼性を高めて管理しやすくするためのヒントも記載します。プログラミング経験が浅い場合は、記載されている内容をすべて理解できなくても構いません。プログラミングの習得には時間がかかるもので、その過程では間違いを犯すことも多分にあります。しかしここで示されるコード例を基にすることで、比較的少ない労力で基本的な独自の JavaScript をすぐに作成できるようになります。

JavaScript の使用場所

JavaScript は PDF ドキュメントの一部に含めることもできますし、ドキュメントの外に置くこともできます。

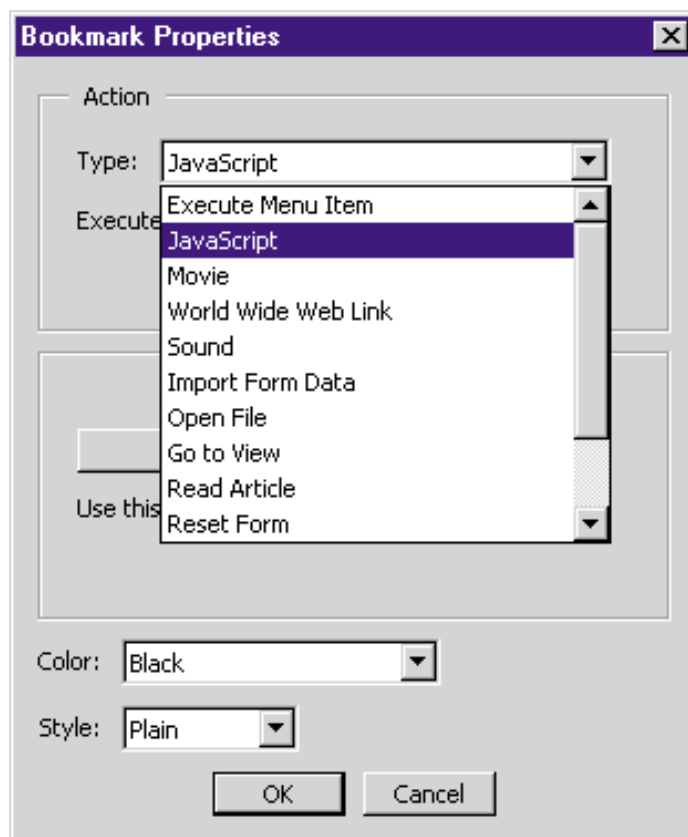
ドキュメント内の JavaScript

JavaScript を PDF ドキュメントに含めるには、6 通りの方法があります。

1. JavaScript を **ページを開く** または **ページを閉じる** アクションに追加できます (Acrobat の **文書／ページ開閉時の動作設定** を使用)。ここに追加したスクリプトは、ページが開く (または閉じる) たびに実行され、ドキュメントを初めて開く場合または閉じる直前にも実行されます。
2. JavaScript を **文書の動作設定** に追加できます (Acrobat の **ツール／JavaScript／文書の動作設定** を使用)。**文書を閉じる** (ドキュメントを閉じる場合)、**文書を保存する** (ドキュメン

トを保存する直前)、**文書を保存した** (ドキュメントを保存した直後)、**文書を印刷する** (ドキュメントを印刷する直前)、**文書を印刷した** (ドキュメントを印刷した直後) など、ドキュメントに関連するいくつかのアクションにスクリプトを追加できます。

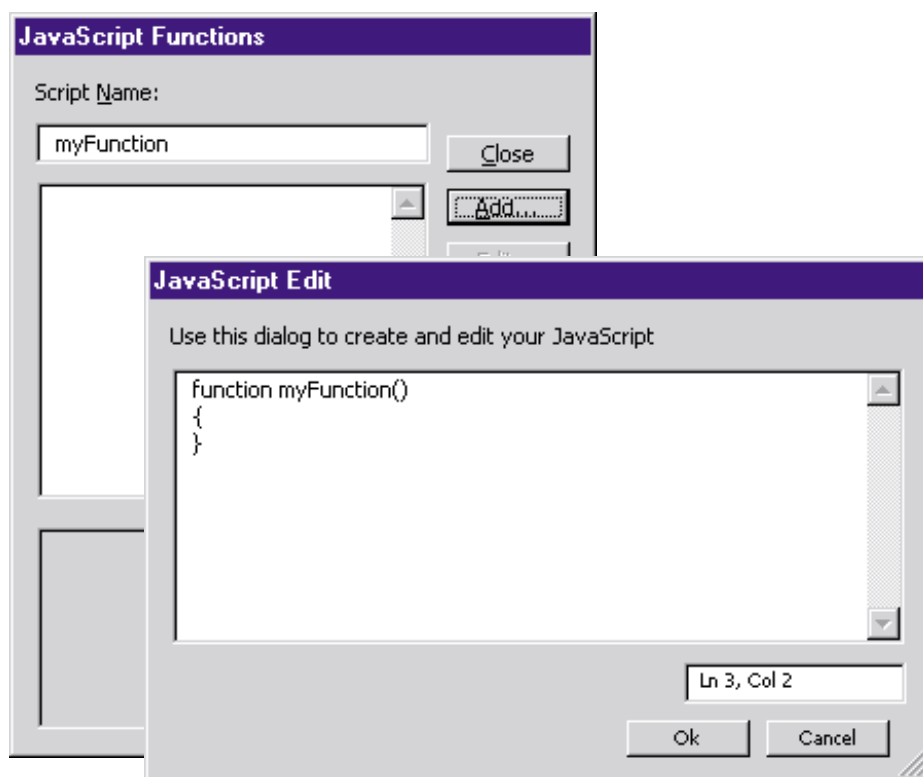
3. JavaScript をしおりに追加できます。Acrobat で Control-B (または Command-B) キーを押してしおりを作成した後、Control-I (または Command-I) キーでしおりのプロパティダイアログを表示します。最初にしおりを選択しておくのを忘れないでください。しおりのプロパティダイアログが表示されたら、次のように、「動作」のリストボックスで JavaScript を選択します。



JavaScript を選択した後、「編集」ボタンをクリックしてスクリプトをしおりに追加します。ここで追加したスクリプトは、しおりをクリックするたびに実行されます。

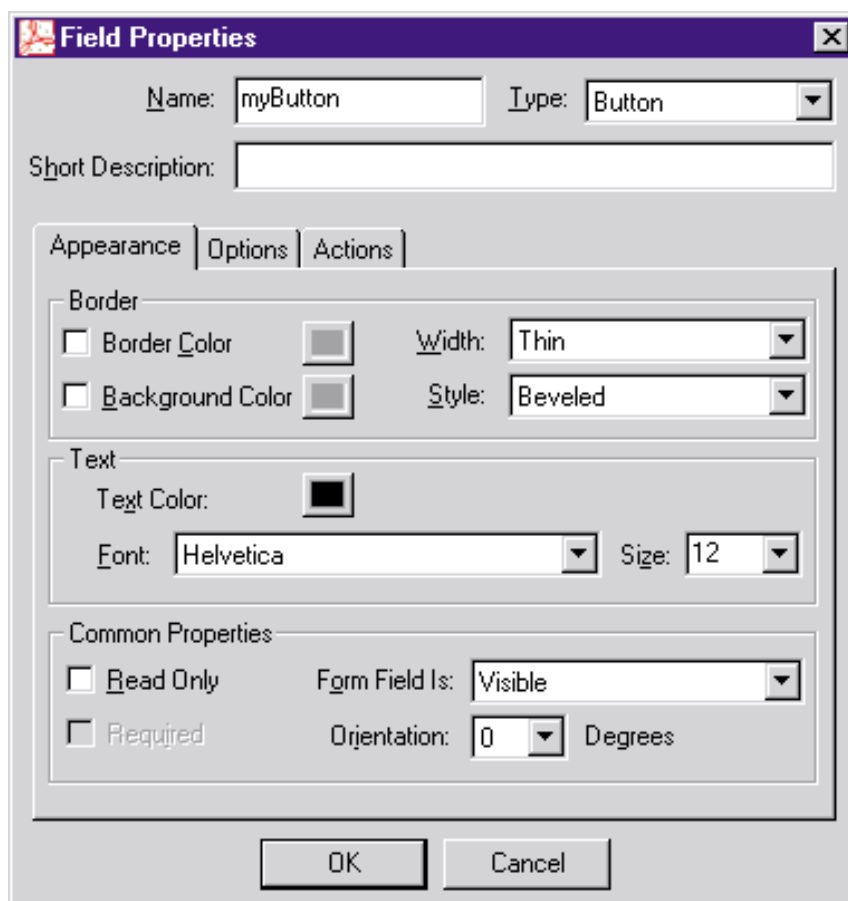
4. Acrobat のリンクツールで作成したリンクに JavaScript を追加することもできます。既存のリンクを選択するか、あるいは、リンクツールを使用して (キーボードの L キーを押す) ページ上でリンク範囲をドラッグし、新規のリンクを作成します。新規のリンクを作成した場合、「リンクのプロパティ」ダイアログが表示されます (既存のリンクを選択した場合は、Control-I / Command-I キーを押して「リンクのプロパティ」ダイアログを表示できます)。「動作」の「種類」のリストボックスで JavaScript を選択し、「編集」ボタンをクリックしてスクリプトを追加または編集します。

5. Acrobat で **ツール / JavaScript / 文書レベル JavaScript の編集** を使用すると、文書レベル（「トップレベル」とも言います）の JavaScript を追加できます。ダイアログが表示されたら、次に示すように、独自の JavaScript を追加できます。



このレベルで追加するコードはすべて、ドキュメント内の他の JavaScript から見える（つまり呼び出して使用できる）ので、これを「トップレベル」のコードと呼ぶこともあります。

6. フォームツールを使用して作成した個々のフォームフィールドに JavaScript を追加できます。フォームツールを選択してから（キーボードの F キーを押して使用可能にする）既存のフィールドをダブルクリックするか、あるいはフォームツールでフィールド範囲をドラッグし、「フィールドのプロパティ」ダイアログを表示します。



このダイアログでは、さまざまな方法で JavaScript をフィールドに追加できます。「アクション」タブには「追加」ボタンで JavaScript を追加できるボックスがあります（上の図を参照）。JavaScript をボタンフィールドに追加する場合（上の図を参照）、**フォーカスを合わせるおよびフォーカスをはずすイベントのほか、マウスボタンを放す、マウスボタンを押す、ポインタを範囲内に入れる、ポインタを範囲外に出すの各イベントに別々のスクリプトを割り当てることができます。また、1つのイベントに複数のスクリプトを追加することもできます（スクリプトは追加した順番で実行されます）。**他の種類のフィールドには、JavaScript を追加可能な別のエリアが存在します。例えば、テキストフィールドにはマウスイベントアクションのほかに、フォーマット、検証、計算用に JavaScript を追加することもできます（「フィールドのプロパティ」ダイアログの「種類」リストボックスで「テキスト」を選択すると、これらのタブが表示されます）。

ドキュメント外の JavaScript

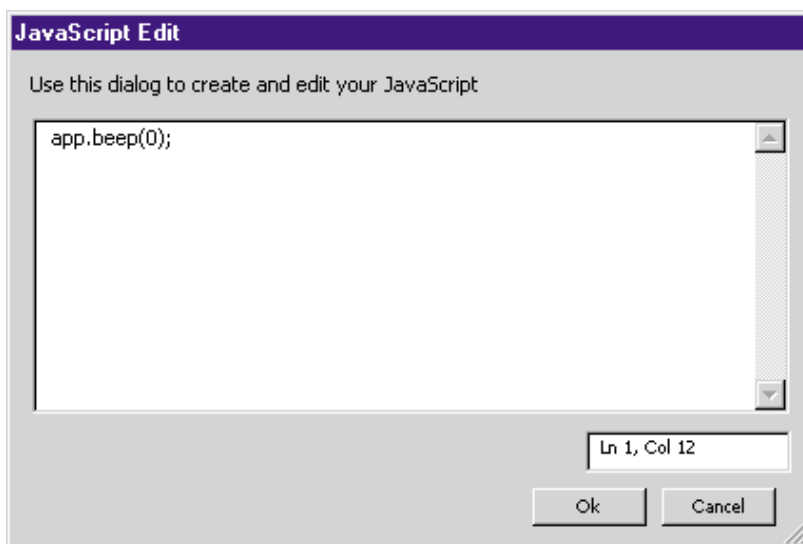
ドキュメント内に保存される JavaScript に加え、ドキュメント外の JavaScript を実行することもできます。外部 JavaScript の実行方法は、3 通りあります。

1. [フォルダレベルの JavaScript](#) : JavaScript を .js という拡張子のファイルに保存し、2 つの JavaScript フォルダ（アプリケーションフォルダとユーザフォルダ）のどちらかに配置できます。Acrobat アプリケーションを起動すると、これらのフォルダがアプリケーションでチェックされ、検出された JavaScript ファイルが実行されます。
2. コンソール JavaScript : JavaScript を Acrobat コンソールに入力して実行できます。この機能について詳しくは、[Console オブジェクト](#)を参照してください。この機能は、おもにテスト用に使用します。
3. バッチ JavaScript（バッチシーケンスとも言います） : Acrobat 5.0 以降では、選択したファイルごとに JavaScript を実行できる強力なバッチシステムが導入されました。詳しくは、Acrobat CD で提供される『Batch Sequences: Tips, Tricks and Examples』のマニュアルを参照してください。

簡単な例

JavaScript の最も一般的な使用例として、フォームフィールドでマウスまたはキーボードイベントが発生した際、計算あるいは他のアクションを実行するという処理があります。例えば、ユーザがフォーム上のボタンを押すたびにスクリプトを実行させたいとします。この設定をするには、「フィールドのプロパティ」ダイアログの「アクション」タブ（上の図を参照）をクリックして、JavaScript をマウスイベントに追加します（ほとんどの場合はマウスボタンを放すイベント）。

JavaScript をフォームフィールドに追加する際、スクリプトを編集するための次のようなダイアログボックスが表示されます。



このボックスの入力内容は、スクリプトを割り当てたフィールドで適切なイベントが発生したときに実行されます。この例では、短いスクリプト（1 行の JavaScript コード）をボタンフィールドの「マウスボタンを放す」イベントに割り当てています。このコードは [App オブジェクト](#)クラスの [beep](#) メソッドを使用して標準のシステムビープを鳴らします。つまり、ユーザがフォームのボタンをクリックするたびにボタン内部で「マウスボタンを放す」

イベントが発生し、システムビープが鳴ることになります（これはごく単純な例ですが、独自の JavaScript コードをフォームフィールドのマウスイベントに割り当てる方法を示しています）。

ヒント： スクリプトのアクションを（「マウスボタンを押す」ではなく）「マウスボタンを放す」イベントに関連付けるのは、適切なユーザインタフェース設計であると言えます。ユーザの指がマウスボタンから放れたときにスクリプトを実行させるのは、ユーザがボタンを押してから考えを変えてマウスポインタをボタンから放したい（アクションが発生する前にキャンセルしたい）場合も考えられるからです。ほとんどの商用ソフトウェアアプリケーションでは、これがボタンのデフォルト動作になっています。

コア言語の特性

PDF ドキュメントで使用する Acrobat™ JavaScript の機能を説明する前に、JavaScript 言語の基となる基本概念の一部を見直してみましょう。

データ型

JavaScript では、基本（コア）データ型（ブーリアン、数値、文字列など）および複合データ型（オブジェクトおよび関数など）をサポートしています。基本データ型は頻繁に使用するので、それが何を意味するのか、JavaScript でどのように解釈されるのかを理解しておくことが重要です。

ブーリアンには *true* と *false* という 2 つの値しかないので、最も単純なデータ型であると言えます。

ヒント： JavaScript では *true* および *false* が内部的にそれぞれ 1 および 0 で表され、JavaScript のブーリアンコンテキストにおいてゼロ以外の値はすべて *true* であると解釈されます。

数値は、3.14、2000、6.023e+23 など標準の科学的記数法で表現されます。他の一部の言語と異なり、JavaScript では整数と浮動小数点数は区別されません。JavaScript では（記述のしかたに関係なく）すべての数値が内部的には 64 ビットの IEEE 浮動小数点数で表されます。十進法で約 20 桁の精度を扱うことができるので、ほとんどのアプリケーションではこれで十分です（JavaScript の精度で不十分な場合、それ以上の精度を持つ言語を見つけるのは難しいでしょう）。

JavaScript で「021」は十進数 17 の 8 進表現であると解釈されるので、整数値の頭にゼロを付けしないでください（3 桁の 8 進数のほか、接頭辞「0x」を付けた 16 進数も使用できます。例えば、0xFF は 255 という値を表します）。

文字列とは、引用符で囲まれた一連の英字、数字、句読点、その他の文字のことです。引用符には一重引用符または二重引用符のどちらを使用してもかまいません（文字列の中に二重引用符が含まれる場合は、文字列全体を一重引用符で囲む必要があります。同様に、文字列の中に一重引用符が含まれる場合は、文字列全体を二重引用符で囲む必要があります）。以下はすべて有効な文字列値です。

```
'This will work.'  
"3.14"  
'The password is "zxcv"'
```

JavaScript には、通常は表現できない文字を文字列中で表すためのエスケープシーケンスがいくつかあります。次の表に、このようなエスケープシーケンスをまとめます。

表 1 JavaScript のエスケープシーケンス

シーケンス	文字
¥b	バックスペース
¥f	改ページ
¥n	改行
¥r	復帰
¥t	タブ
¥'	一重引用符
¥"	二重引用符
¥¥	円マーク（バックスラッシュ）
¥uXXXX	4 桁の 16 進数で示されるユニコード文字

配列とは、データ値の集合のことです。JavaScript では配列要素にどの基本データ型でも使用でき、必要であれば同じ配列に異なるデータ型を持たせることもできます。C 言語と同様に、配列のメンバは角かっこで囲まれた数値インデックスによって指定またはアクセスできます。したがって、*names* という配列の最初の要素に保存された値を示すには、*names[0]* と指定します。同様に、配列の 2 つ目の要素は *names[1]*、3 つ目の要素は *names[2]* となります。

ヒント： コア JavaScript の組み込みクラスには、String や Array 用に便利（かつ強力）なユーティリティメソッドが多数含まれており、もちろん Math、Date、RegExp などのコア言語のクラスもあります。これらのクラス（および関連メソッド）はすべて、Acrobat™ JavaScript で使用することができます（これらのクラスおよびメソッドについて詳しくは、JavaScript に関する書籍をご覧ください）。

変数

JavaScript では変数の宣言にキーワード「var」を使用します。JavaScript で変数を使用する場合は、以下の点に注意してください。

- JavaScript ではすべての識別子の大きい文字と小さい文字が区別されます。つまり、「MyVariable」という名前の変数は「myVariable」という変数とは異なります。
- JavaScript で作成した変数は、そのスコープ内において永続的です。一度宣言した変数を「宣言解除」したり破棄したりすることはできません（JavaScript のガベージコレクション機構により、変数が使用されなくなると自動的に割り当てが解除されます）。
- JavaScript の変数はすべてスコープを持っており、それによって変数の存続期間とアクセシビリティ（特定の期間に使用できるかなど）が決定されます。関数内で宣言した変数のスコープはローカルとなるので、変数を宣言した関数内でしか使用できません。

変数を宣言するには、次のように入力します。

```
var circumference;
```

あるいは、変数の宣言と初期化を次のように 1 つのステートメントにまとめることもできます。

```
var circumference = 6.28;
```

変数の宣言後は変数の前にキーワード「var」を付ける必要はなく、基本データ型と同様に式の中で使用できます。

ヒント： JavaScript では、キーワード「var」を使用せずに変数を宣言することもできます。しかし、上述のようにすべての JavaScript 変数はスコープを必要とするので、「var」のない変数についてはインタプリタがスコープを割り当てることになります。インタプリタはこの問題を解決するために、このような変数をグローバルオブジェクト（「すべてのランタイムオブジェクトの親」）に割り当てます。結果的に、このような変数はプログラムのどこからでも「使用できる」ようになります。意図的にこのようなプログラムを書くこともできますが、通常はしません。

他のほとんどの「高水準言語」と異なり、JavaScript では変数の型が厳密ではありません。つまりプログラマは、変数を数値あるいは文字列のどちらとして扱うべきかを JavaScript に指示する必要がないということです。JavaScript は変数を「適切に判断」し、数値コンテキストでは数値として、文字列コンテキストでは文字列として扱います。

JavaScript はオブジェクト指向言語ですが、Java や C++ と異なり、オブジェクトをプロトタイプ化する必要はありません。新規のオブジェクトを作成するには、次のようにするだけです。

```
var hue = new Object();
```

このステートメントによって JavaScript のコンストラクタ関数が呼び出され、プロトタイプのない汎用的なオブジェクトが作成されます。オブジェクトを作成したら、次のようにしてフィールド（オブジェクトで基本データ型または関数のどちらを参照するかによって、プロパティまたはメソッドのいずれかになります）をオブジェクトに割り当てることができます。

```
hue.favorite = "taupe";
hue.worst = "teal";
hue.maximum = 15;
hue.sum = function (a,b) { return a + b; }
```

以上の 4 つのステートメントはすべて有効です。最初の 3 つのステートメントは、プロパティを作成し、それに値を割り当てています（ステートメントの 1 つは値が数値で、他の 2 つでは値が文字列になっています）。4 つ目のステートメントは、オブジェクトに関数を割り当てることによって、新規のメソッドを作成しています。このメソッドは、他の関数と同じように呼び出して使用することができます。

```
var x = hue.sum( 2, 2 ); // 'x' now has the value '4'
```

配列とオブジェクトはどちらも任意のデータ型の集合ですから、ある意味では似ています。配列の場合は、数値インデックスと角かっこを使用してコンポーネントにアクセスします。オブジェクトの場合は、オブジェクト名とコンポーネント名をピリオドでつなげたものを識別子としてコンポーネントにアクセスします。

未定義の変数

JavaScript では、他の言語と同様に、変数を宣言せずに式で使用するとランタイムエラーになります。その結果、コンソールウィンドウ（Acrobat Business Tools または標準の Acrobat の場合）またはメッセージボックス（Acrobat Reader の場合）がポップアップされ、「～変数が未定義」というエラーが表示されます。次に例を示します。

```
function memberSum() {
    var member = new Object();           // create a new, blank object
    member.john = 1;                     // create a new property, 'john'
    return (totalmembers + member.john); // error!
}
```

この場合、*totalmembers* は新しい変数ですが、どこにも宣言されていません。この変数を *return* 式で使用すると、ランタイムエラーになります。JavaScript では、上記のコードに示すように、オブジェクトの新規のプロパティをその場で作成できるので、*member.john* を作成して「1」の値を割り当てるコードはエラーになりません（上の「オブジェクト」の説明を参照）。

JavaScript において型は厳密ではないと前述しましたが、これは、型を持たない言語であるということではありません。JavaScript にはデータ型があり、実際、このデータ型によって、変数が定義されているかどうかを実行時に調べるができるわけです。変数が定義されているかどうかをテストするには、*typeof* 演算子を使用します。


```
if (typeof aVariable == "undefined") // undefined variable?
    app.alert("Undefined variable.");
else {
    // the variable is safe to use
}
```

ここでは、[App オブジェクト](#)に事前定義されている [alert](#) メソッドを使用して、テスト対象となる変数が未定義の場合に警告ダイアログを表示しています。JavaScript では C 言語に似た `==` 演算子を使用して、等しいかどうかをテストしていることに注目してください。文字列は値により比較されるので、`==` 演算子を使用して2つの文字列を直接比較することができます。

コメント

C++ または C のスタイルで JavaScript コードにコメントを挿入できます。つまり、ダブルスラッシュ (`//`) から改行までのテキストはすべて、JavaScript では無視されます。また、`/*` と `*/` の間にあるテキストは、実行の対象外（またはコメント）として処理されます。以下は有効なコメントの形式です。

```
// This is a single-line comment.
/* This is also a comment. */    // as is this
/*
 *
 * You can, if you want, also do this.
 *
 */
```

コメントを付けるとプログラムが読みやすくなり管理も楽になるので、コメントは非常に重要なツールであると言えます。JavaScript の構文は C 言語のように簡潔になっているので、わずか数行のコードで多くの処理を行うことができます。このため、プログラムを非常にコンパクトにまとめることができますが、解読しづらくデバッグが困難になる傾向もあります。したがって、プログラムを作成するときは「コードに十分なコメントを入れる」よう特に注意を払ってください。そうすることでコードの信頼性が増し、しばらくしてからコードを見直すような場合でもすぐにコードの内容を思い出すことができます。

句読点

JavaScript ステートメントの末尾には、厳密に言えば（C または Java ステートメントと異なり）セミコロンを付けなくても構いません。ステートメントにセミコロンがなくても、プログラムは問題なく実行されます。ただし、適切な場所にはすべてセミコロンを挿入するのが良いでしょう。同様に、次のような特定のコンストラクタ関数ではかっこを省略することもできます。

```
var n = new Object;
```

しかし、これも良い書き方とは言えません。ほとんどの関数にはかっこが必要ですから、常にかっこを使用するようにしておけばプログラムの一貫性が保たれ、読みやすくなります。

メソッドのパラメータ指定

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

Acrobat JavaScript メソッドのパラメータ（または引数）リストは、次の 2 通りの方法で指定できます。

1. パラメータをコンマで区切って並べます。これは標準コア JavaScript の手法です。オプションのパラメータは指定してもかまいませんし、空文字列によってスキップしたり、リストから切り捨てることもできます。パラメータは正しい順序で指定する必要があります。次に例を示します。

```
app.alert("Hello World!", 1, 1);
app.alert("Hello World!", "", 1); // Take default for second parameter
app.response("How are you today?");
app.response("How are you today?", "", "Fine"); // Specify cDefault only
```

2. オブジェクトリテラルを使用して、引数の情報をメソッドに渡します。オブジェクトリテラルとは、「オブジェクトのプロパティ名と値のペアを 0 個以上、中かっこ ({}) で囲んだリスト」のことです。¹ 以下のコードは、上記の例と同じ意味になります。

```
app.alert( {cMsg: "Hello World!", nIcon: 1, nType: 1} );
app.alert( {cMsg: "Hello World!", nType: 1} );
app.response( {cQuestion: "How are you today?"} );
app.response( {cQuestion: "How are you today?", cDefault: "Fine"} );
```

オブジェクトリテラルに含まれるプロパティ名は、メソッドの仕様で定義されているパラメータ名で、例えば、[app.response](#) メソッドでは次のように定義されています。

response

パラメータ : *cQuestion*、*[cTitle]*、*[cDefault]*、*[bPassword]*

戻り値 : *cResponse*、キャンセルの場合は *null*

- 1 つの必須パラメータと、3 つのオプションパラメータがあります。パラメータ名は、`App.response` の引数としてオブジェクトリテラルを作成するために使用されます。上の例を参照してください（戻り値の名前 *cResponse* は、メソッドから文字列が返されることを示しており、それ以外に特に意味はありません）。

注意： オブジェクトリテラルを使用してパラメータを指定する方法は、すべての Acrobat JavaScript 関数で利用できるわけではありません。

¹. 『Core JavaScript Guide』、バージョン 1.4、Netscape Communications Corporation、1998 年、10 月 30 日、31 ページ。

メソッドのクイックヘルプ

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

Acrobat メソッドで *acrohelp* という引数を指定してメソッドを（コンソールエディタなどで）実行すると、引数名と型のリストがメソッドから返されるので、非常に便利です。

例：コンソールで次のように入力します。

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

カーソルがまだ入力行にある状態で Ctrl+Enter または数値キーパッドの Enter キーを押すと、コンソールに次のように出力されます。

```
uncaught exception: Console:Exec:1: Help for App.response
====> [cQuestion: string]
====> [cTitle: string]
====> [cDefault: string]
====> [bPassword: boolean]
```

オプションパラメータは角かっこで囲まれて表示され、必須パラメータはかっこなしで表示されます。

例外処理

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

Acrobat JavaScript メソッドでは、エラー時に例外が発生します。スクリプト作成者は *try/catch/finally*（JavaScript 1.4）や *throw* ステートメントを使用して、これらの例外を制御することができます。例えば、次のコードは ADBC プラグインを使用してデータベースへの接続を試みます。

```
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";
        statement.nextRow() // Statement.nextRow() throws an exception...
        return true;        // if there is no next row
    } catch(e) {
        app.alert(e);
        return false;
    }
}
```

開発者およびスクリプト作成者は、次のスクリプトを実行することで、例外オブジェクトの構造情報を取得できます。

```
try {
    app.alert();    // force an exception, one argument required for alert
} catch(e) {
    for ( var i in e )
        console.println( i + ": " + e[i])
    // now display the error message, which is e.toString()
    console.println("e: " + e );
}
```

エラーハンドラを作成して、Acrobat プラグインで発生した例外や、上記の（非常に単純な）最初の例のような独自の JavaScript コードで発生した例外を処理することができます。

例外が発生する一般的な例としては、基になるオブジェクトが存在しない場合などがあります（オブジェクトが存在しないにも関わらず、オブジェクトへの参照が行われるなど）。次に例を示します。

```
var myDoc = app.newDoc();
myDoc.close();
myDoc.pageNum = 3;
```

この場合、3 行目が実行された時点で例外が発生します。ドキュメントは閉じられた後なので存在しませんが、ドキュメントへの参照は `myDoc` にまだ存在しており、参照を使おうとしたとき例外が発生します。

```
uncaught exception:Console :Exec:1: Doc.pageNum object is dead.
```

Acrobat 5.0 での JavaScript の編集



ドキュメント内のすべての JavaScript の編集

Acrobat メニューの「すべての JavaScript を編集」(ツール／JavaScript／すべての JavaScript を編集) を使用して、PDF ドキュメントに含まれるすべての JavaScript を表示および編集できるようになりました。すべてのスクリプトは XML タグで整理されて表示されるので、XML タグのコンテンツにより、多くのスクリプトの中から目的のスクリプトを簡単に見つけることができます。XML タグはスクリプトを識別するために Acrobat によって追加されたものですから、絶対に編集しないでください。XML タグを編集すると、変更内容が受け付けられなくなる可能性があります。編集は、適所に配置されたダイアログ上のボタンから行うこともできます（フォームの「フィールドのプロパティ」ダイアログの「アクション」タブなど）。

外部エディタ

Acrobat で外部エディタを指定して JavaScript を編集できるようになりました（**編集／環境設定／一般／JavaScript**）。外部エディタを指定すると、Acrobat で JavaScript を編集する時に毎回そのエディタが使用されるようになります。編集の際には Acrobat で一時ファイルが生成され、それが外部エディタで開かれます。変更内容を Acrobat に受け付けてもらうには、このファイルを保存する必要があります。外部エディタでファイルを編集している間は Acrobat にアクセスできません。エディタを閉じると再び Acrobat にアクセスできます。

注意： この機能は、Macintosh では使用できません。

内部エディタ内での Tab キーの使用

Tab キーを押すと、カーソル位置に 4 つのスペースが挿入されます。Shift キーを押しながら Tab キーを押すと、4 スペース分（またはそれ以下）カーソルが左に移動します。

行またはその一部を選択してから Tab（または Shift+Tab）キーを押すと、4 スペース分（Shift+Tab の場合はそれ以下）だけ行全体が右（Shift+Tab の場合は左）に移動します。複数の行についても、複数行を選択してから Tab または Shift+Tab キーを押すことで、同じように移動することができます。

対話型 JavaScript コンソール

5.0			⊗
-----	--	--	---

Acrobat JavaScript コンソールでは、対話形式の編集が可能になりました。コンソールに記述した JavaScript コードは直ちに評価することができます。評価には数値キーボード上の <Enter> キーを使用するか、Windows または Macintosh の場合は Ctrl+Enter キーを使用することもできます。以下に記載している JavaScript の実行に関する説明、およびコンソール内でのタブキーの使用に関する説明は、コンソール、文書レベルの JavaScript エディタ、フォームアクション JavaScript エディタなど、すべての内部 JavaScript エディタに共通します。

JavaScript の実行

JavaScript コードをコンソールで評価するには、基本的に 2 通りの方法があります。コードブロックを評価するには、コードを入力してからコード全体を選択し、<Enter> (Ctrl+Enter) キーを押します。1 行のコードを評価するには、評価したい行にカーソルを置き、<Enter> (Ctrl+Enter) キーを押してください。

コードの評価結果は、コンソールに直ちに表示されます。コードブロックを評価する場合、ブロックで最後の式の結果しかコンソールに表示されないのに注意してください。また、評

価の結果は、式で設定した変数の値と同じではありません。例えば、変数「x」を 0 の値に設定する次の式を評価するとします。

```
var x = 0;
```

コンソールには、次のように表示されます。

```
undefined
```

これは、「x」の値が「定義されていない」ということではなく、式全体が「undefined」を返しているということです。「x」の値を知りたい場合、式の中の「x」のみを選択して <Enter> キーを押せば、「x」の値 0 が表示されます。コンソールで結果を即座に知りたい場合、`console.println()` メソッドも便利です。

JavaScript コードの評価には、Acrobat JavaScript 固有の編集ダイアログも使用できます。この場合も出力はコンソールに表示されますが、それ以外はコンソールと同じように動作します。出力をコンソールに表示するのは、編集ダイアログに入力したコードと評価結果の表示が混在しないようにするためです。

信頼性の高いコードを作成するためのヒント

プログラミングにはバグがつきものです。しかし、バグに対処しなくて済むのであれば、それに越したことはありません。正しいコーディングを行い、コードの信頼性を高める努力を常に怠らないようにしてください。ここでは、より良いコードを書くためのヒントを記載します。

1. 初めてプログラムを学ぶのであれば、他人が作成したデバッグ済のコードサンプルを変更していくのが最も簡単です。変更の結果を完全に理解できるようになるまで、一度に少しずつ変更を加えていくようにします。
2. 変数には意味のある名前を使用します。numberOfAvailableColors などの説明的な名前は長くなりますが、num や n などの短く暗号的な名前よりは、一般的に良いとされます。
3. 変数名には一貫性を持たせます。プログラマによっては、変数名の先頭に説明的な接頭辞を付けることもあります。例えば、数値の変数名の先頭には「n」（nTotal、nMaximum など）、ブーリアンの先頭には「b」、文字列には「s」、グローバルスコープを持つ変数には「g」などの接頭辞を付けます。どのような方法を採用にしろ、コードに一貫性を持たせると後で非常に役立ちます。
4. 初期化されていない変数に注意します。変数を宣言することは、初期化する（初期値を割り当てる）こととは異なります。値のない変数を（比較ステートメントなどで）使おうとするとエラーになります。変数の作成時には安全な「ダミー値」を常に割り当てるようにすると、この種のエラーを防ぐことができます。

5. 簡潔さではなく読みやすさを重視します。1 つの JavaScript ステートメントを 2 つに分けたほうが明確に処理内容を表現できる場合、2 つのステートメントを使用してください。コードの短さを誰かと競うわけではありませんし、多くの場合、実行速度はどちらも同じか、ほとんど変わりません。
6. コードの編集時には「切り取り」および「貼り付け」コマンドを使用して、キー入力を少なくします。キー入力が少ないければ、コード中でタイプミスをする可能性も低くなります。
7. かっこの対応に注意します。ネストしたステートメントに特に注意し（あるいは複数行に書く）、すべての開くかっこが閉じるかっこと対になっていることを確認します。
8. コメントを十分に使用します。コメントが少ないコードは管理が難しくなります。
9. 複雑な処理を短い関数に「分解」します。一般に、2 行から成る 6 つの関数は、12 行から成る 1 つの関数に比べると、分かりやすくデバッグも楽です。また、コードを分解すると、再利用しやすくなるという利点もあります。
10. 常に作業用のコードを用意します。作業時には頻繁にバックアップを取ってください。「問題のない」既存のコードを編集する前に、そのコピーを取っておくようにします。そうすれば変更を加えた新しいコードで修正不可能なバグが生じても、元に戻すことができます。
11. 可能な限り、他のプログラマのコードを研究します。すでにあるものを一から再開発する必要はありません。他のプログラマのコードから学び、最良のアイデアを自分のスクリプトに取り込んでいくうちに、プロのようなコーディングができるようになります。

PDF フォームでの JavaScript の使用

JavaScript を AcroForms と併用すると、特に効果的に使用できます。ボタン、テキスト、コンボボックス、その他の「フォームウィジェット」にスクリプトを割り当てることにより、フォームの対話性、外観、信頼性を大幅に高めることができます。例えば、JavaScript を使用してユーザ入力のフォーマットや検証をしたり、サーバ上の CGI スクリプトや電子メールアドレスへフォームデータを送信したり、さらにボタンのカスタムアクションを作成したりすることができます。ここでは、一般的なスクリプトアクションを取り上げ、フォームによる典型的なやり取りを JavaScript で設定する方法について説明します。

コードの作成

[JavaScript の使用場所](#)で記載したように、JavaScript を PDF ドキュメントに割り当てる場所としては、フィールドレベル（アクションに割り当てる場合、またはカスタムフォーマット、検証、計算などのスクリプトとして割り当てる場合）、「ページを開く」または「ページを閉じる」アクションのレベル、リンクまたはしおり、そして文書レベルがあります。コードをどこに配置するかを決めるのは簡単で、通常、コードで制御するアクションまたはフォームフィールドにできるだけ近いところにコードをリンクさせます。しかし大規模なプロジェクトでは、同じような処理を多くの異なる場所で行う必要もあります。例えば、一連の数値の平均値を繰り返し計算しなければならないことがあります。この場合、数値の平均を計算するコードをいくつも作成せずに、この処理を行うユーティリティ関数を 1 つだけ作成します。そしてそれを文書レベルに保存すれば、ドキュメント内のどのスクリプトからでもアクセスできるので効率的です。例えば、**ツール / JavaScript / 文書レベル JavaScript の編集**で次のように入力します。

```
function average() {  
    if (arguments.length == 0) // nothing to do  
        return 0;  
    var sum = 0.0;  
    for (var i = 0; i < arguments.length; i++)  
        sum += arguments[i] - 0;  
    return sum/arguments.length;  
}
```

そして、ドキュメント内の任意の場所にある他のコードからこの関数を呼び出せば、数値の平均を計算できます。

ヒント： 上記の関数では、コア JavaScript の arguments 配列を使用しています。これは全ての JavaScript 関数内で使用することができます。average() 関数には任意の個数の引数を渡すことができ、一部または全部の引数が文字列型であっても、渡した引数の平均が計算されます。arguments[i] からゼロを引いているのは、arguments[i] が文字列型の場合、インタプリタで強制的に数値型に変換するためであり、これは JavaScript での標準的な手法です。したがって、average(1,5,9) と average("1","5","9") は、同じ値を返します。

頻繁に使用する関数を文書レベルに保存すると、いろいろな意味で利点があります。フィールドレベルのコードをすっきりとさせて合理化できるだけでなく、コードの再利用が促進されて読みやすくなります（したがって管理しやすくなります）。ユーティリティ関数を文書レベルのスクリプトに「分解」すると、プロジェクトの管理やデバッグが楽になります。

フィールドの操作

JavaScript の非常に一般的な使用法として、ユーザ入力値の計算があります。これを行うためには、フォームフィールドの値を取得する必要があります。このために必要なプロセスは 2 つで、JavaScript では次のように非常に簡単です。

```
var theField = this.getField("City");
var theValue = theField.value;
```

ここで、コードの最初の行でどのような処理が行われているかに注目してみましょう。まず、新規の変数 *theField* を宣言し、それに値を割り当てています。代入演算子（等号）の右辺は、*this.getField("City")* です。「this」という参照は JavaScript で使われるショートカットで、文書レベルまたはフィールドレベルのスクリプトでは現在のドキュメントオブジェクトを意味します。*getField()* はドキュメントオブジェクト（[Doc オブジェクト](#)を参照）に含まれる多数の事前定義メソッドの 1 つで、このメソッドの引数には、プロパティを操作または取得したいフィールド名を与えます。上の例では、「City」という名前のフィールドがフォームのどこかに含まれていることを前提としています。

コードの 2 行目では、2 つ目の変数 *theValue* を宣言しています。この変数には、フィールドの実際の値（つまり、ユーザ入力値）が保持されます。Field オブジェクトには独自のプロパティとメソッドがあり（[Field オブジェクト](#)を参照）、「value」プロパティにはユーザが設定したフィールド値が含まれています。テキストフィールドの場合、この値は文字列になります。

フィールド値は JavaScript を使用して実行時に変更することができます。それには、該当する Field オブジェクトの「value」プロパティを目的の値に設定します。次に、フィールド値の正しい設定方法と誤った設定方法を示します。

```
var theValue = "San Jose";           // Wrong. Does not alter field.
theField.value = "San Jose";         // Correct. Will alter field.
```

最初の行では、ローカル変数 *theValue* に新しい値を設定していますが、これではフォームの「City」フィールドの値を変更することはできません。これはフィールドの値ではなくローカル変数の値を変更しているだけです。フィールドを操作するには、[Field オブジェクト](#)自体にアクセスする必要があります。このオブジェクトへの参照を *theField* に取得済であれば、上記の 2 つ目の方法でフォームに表示される値を変更できます（フォームの外観は直ちに更新されて、新しい値が表示されます）。

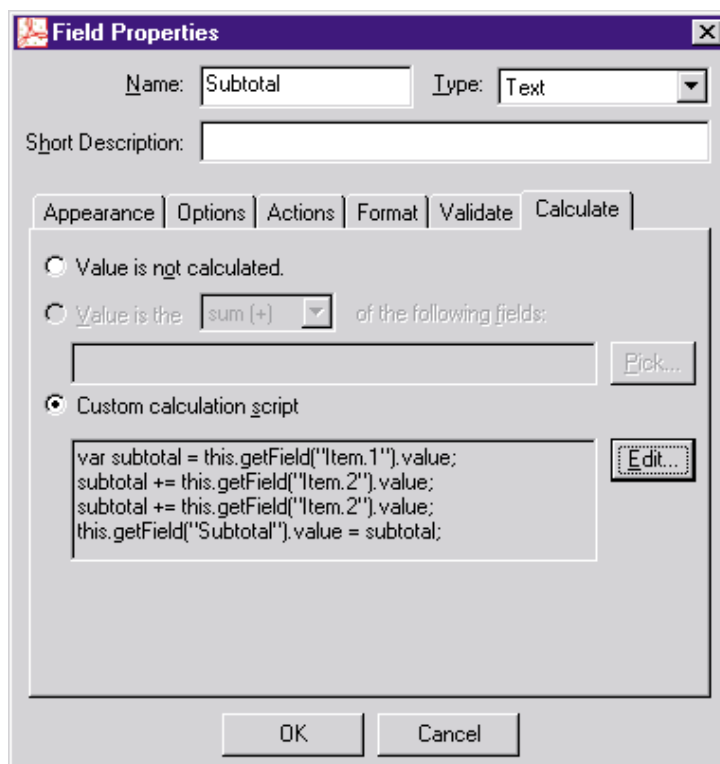
このフィールドの値は、次のようにして変更することもできます。

```
this.getField("City").value = "San Jose";
```

1 行のコードで、フォームの「City」フィールドの値（存在する場合）を新しい値で上書きできたことになります。

相互依存するフィールドの構成

JavaScript の一般的な別の使用法として、フォーム内の各フィールドデータの結合があります。例えば、配送料をフォームの "Zip Code" フィールドの値に応じて変更したり、ユーザによる現在の購入額の小計を "Subtotal" フィールドに保持したいことがあります。通常、このようなフィールド固有のデータを処理するのに最も適した方法は、計算スクリプトを使用することです。新規のフォームフィールドを作成（または既存のフォームフィールドをダブルクリック）して、「フィールドのプロパティ」ダイアログを表示してください。フィールドの種類がテキストまたはコンボボックスのどちらかであることを確認してから、「フィールドのプロパティ」ダイアログの「計算」タブを選択します。



フォームが更新されるたびに実行される計算スクリプトを入力するには、「カスタムの演算スクリプト」ラジオボタンに切り替えて、右にある「編集」ボタンをクリックします。

ヒント： Acrobat では、ツール／フォーム／フィールドの計算順序の設定を使用すると、フィールドの計算順序を手動で変更することができます。また、[Field オブジェクトの calcOrderIndex プロパティ](#)を使用して、計算順序をプログラムで設定することもできます。デフォルトのフィールドの順序は、必ずしも希望の順序であるとは限りません。

フォーマットスクリプトと検証スクリプト

JavaScript を使用して、ユーザの入力時に入力内容を自動的にフォーマットしたり検証したりすることができます。「フィールドのプロパティ」ダイアログ（テキストおよびコンボボックスフィールドの場合のみ）には、「フォーマット」および「検証」タブが表示されま

す。これらのタブを選択すると専用のペインが表示されるので、そこで独自のスクリプトをフィールドに割り当て、フォーマットや検証を行うことができます。フォーマットスクリプトと検証スクリプトの違いは何でしょうか。フォーマットスクリプトはユーザが入力したデータの外観に関係しますが（例えば、電話番号の市外局番をカッコで囲むなど）、検証スクリプトはデータの基となる値（電話番号そのもの）に関係します。

ユーザのキー入力をフィルタするには、「フィールドのプロパティ」ダイアログの「フォーマット」タブを選択し、カスタムキーストロークスクリプトを入力します。カスタムキーストロークスクリプトでは [Event オブジェクト](#) の [change](#) プロパティの値を調べ、ユーザが入力した値を受け付けるか拒否するのが一般的です。次の簡単なスクリプトでは、ユーザが数値のみを入力したかをチェックし、そうでない場合はすべて拒否します。

```
if (event.change.match(/¥D+/g)) {  
    app.alert("Enter numeric characters only.");  
    event.rc = false;  
}
```

条件チェック（このコードの一番上の行）では JavaScript の String クラスに組み込まれている `match()` メソッドを使用し、数値以外の文字を表す正規表現と照合して、ユーザの入力をチェックしています（¥D は数値以外の文字を意味します）。

ヒント： 正規表現では特殊な記号による表記を使用し、明確に定義された規則に基づいてパターンマッチが行われます。効果的な文字列検索、置換、マッチ操作を行うための正規表現の使用方法については、[JavaScript に関する本をご覧ください](#)。

数値以外の文字を含む値をユーザが入力すると、2 つの処理が行われます。まず警告ダイアログを画面に表示して数値のみを入力するようユーザに指示し、次に [Event オブジェクト](#) の「`rc`」プロパティを `false` に設定することで入力された文字が表示されないようにします。

`event.change` は文字列を扱えることに注意してください。つまり、キー入力される 1 文字に限らず、貼り付け操作で入力される長い文字列を参照することも可能です。このため、入力されるテキストを文字列として扱うことができ、そこに含まれるすべての文字をチェックすることが可能です。

拡張フォーマット

ユーザが入力したフィールドデータに、厳密なフォーマット条件を適用しなければならない場合があります。例えば、数値の精度を小数点以下 2 桁までに制限したり、ユーザが入力した日付や通貨の値に特定のフォーマット条件を適用しなければならない場合がありますが、このような処理は意外に面倒なことがあります。日付、時刻、通貨などの値については、Acrobat に組み込まれている「フォーマット」オプションで処理することができます（「フィールドのプロパティ」ダイアログの「フォーマット」タブを参照）。しかし、独自に作成したカスタムスクリプトで Acrobat の組み込みのフィルタを補わなければならないこともあります。ここで、役立つ情報が 2 つあります。まず、[Util オブジェクト](#) クラスには、

フォーマットを行うのに便利なくつかの組み込みメソッドが用意されています。例えば、数値の文字列をフォーマットするには [printx](#) メソッドを使用することができます。

```
var v = "aaa14159697489zzz";  
v = util.printx("9 (999) 999-9999", v);
```

この場合、文字列 "aaa14159697489zzz" は "1 (415) 969-7489" としてフォーマットされます（詳しくは、[printx](#) を参照）。同様に、[printd](#) および [scand](#) メソッドは、さまざまな条件に応じて日付をフォーマットするのに便利です。また、C 言語でお馴染みの [printf](#) も使用できます。

下位レベルのフォーマットに役立つ 2 つ目の情報として、*AForm.js* ファイルのユーティリティ関数があります。このファイルは JavaScript ディレクトリ (*Acrobat / JavaScripts*) に入っています。弊社が提供するこのファイルには、日付文字列や数値の操作に便利な数多くの関数とユーティリティが含まれています。例えば、関数 *AFMakeNumber()* は、セパレータとしてピリオドなどを使用している文字列から数値を作成します（多くの場合、ピリオドではなくコンマですが）。この関数および他のユーティリティ関数のソースコードは、*AForm.js* に含まれています。

ヒント： *AForm.js* は、*Acrobat*（または *Reader*）が起動するときに自動的に読み込まれます。*AForm.js* に含まれている全てのグローバル変数と関数は、*Acrobat* の実行時に独自のスクリプトから利用することができます。

PDF と HTML の違い

HTML Web ページ用の JavaScript コードの作成に慣れている場合、*window.open()* や *document.write()* といったメソッドを呼び出したくなるかもしれませんが、しかし、ブラウザ環境で扱い慣れているオブジェクト、メソッド、プロパティの多くは PDF JavaScript では動作せず、存在すらしません。なぜなら PDF 用の JavaScript は、*Acrobat* または *Acrobat Reader* 内で動作するからです。この場合のランタイムインタプリタはブラウザ内にあるものとは異なります。PDF JavaScript で使用されるオブジェクトやメソッドのスコープは PDF ドキュメント自体にあり、HTML ページはスコープの外にあります。

既にご存知かもしれませんが、HTML ベースの JavaScript で見慣れている多くのオブジェクトやメソッドはコア言語の一部ではなく、コア言語に対するクライアントサイドの拡張機能の一部です。サーバサイドの JavaScript では、サーバ環境に関連する別の「アドイン」メソッドと「アドイン」オブジェクトを使用しています。これらはサーバサイドの拡張機能です。同様に PDF も、独自のオブジェクトとメソッドを持っています。HTML ページ用に書いたコードはブラウザで実行され、サーバサイドの JavaScript コードはサーバで、そして PDF ドキュメント用の JavaScript は *Acrobat*（または *Acrobat Reader*）で動作するということを常に念頭に置いてください。それぞれで使用可能なメソッドは異なります。

HTML ページ用に書いた JavaScript で PDF ドキュメントの「内部を見る」ことができないのは言うまでもありません。また、PDF ドキュメント用のコードでも HTML ページの内部を見ることはできません（ただし、どちらのコードもサーバとやり取りすることはできます）。

5.0 の新機能

オブジェクト	プロパティ	メソッド
ADBC オブジェクト		getDataSourceList() , newConnection()
Annot オブジェクト	alignment , AP , arrowBegin , arrowEnd , attachIcon , author , contents.doc , fillColor , hidden , modDate , name , notelcon , noView , page , point , popupRect , print , points , print , rect , readOnly , rotate , strokeColor , textFont , type , soundIcon , width	destroy() , setProps() , setProps()
App オブジェクト	activeDocs , fs , plugIns , viewerVariation	addMenuItem() , addSubMenu() , clearInterval() , clearTimeout() , listMenuItems() , listToolbarButtons() , newDoc() , openDoc() , popupMenu() , setInterval() , setTimeout()
Bookmark オブジェクト	children , doc , name , open , parent , style	createChild() , execute() , insertChild() , remove()
Color オブジェクト		convert() , equal()
Connection オブジェクト		newStatement() , getTableList() , getColumnList()
Data オブジェクト	creationDate	creationDate , modDate , MIMEType , name , path , size

オブジェクト	プロパティ	メソッド
Doc オブジェクト	bookmarkRoot , icons , info , layout , securityHandler , selectedAnnots , sounds , templates , URL , disclosed(5.0.5)	addAnnot() , addField() , addIcon() , addThumbnails() , addWeblinks() , bringToFront() , closeDoc() , createDataObject() , createTemplate() , deletePages() , deleteSound() , exportAsXFDF() , exportDataObject() , extractPages() , flattenPages() , getAnnot() , getAnnots() , getDataObject() , getIcon() , getPageBox() , getPageLabel() , getPageNthWord() , getPageNthWordQuads() , getPageNumWords() , 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 オブジェクト	changeEx , keyDown , targetName	
Field オブジェクト	buttonAlignX , buttonAlignY , buttonPosition , buttonScaleHow , buttonScaleWhen , currentValueIndices , doNotScroll , doNotSpellCheck , exportValues , fileSelect , multipleSelection , rect , strokeColor , submitName , valueAsString	browseForFileToSubmit() , buttonGetCaption() , buttonGetIcon() , buttonSetCaption() , buttonSetIcon() , checkThisBox() , defaultIsChecked() , isBoxChecked() , isDefaultChecked() , setAction() , signatureInfo() , signatureSign() , signatureValidate()

オブジェクト	プロパティ	メソッド
FullScreen オブジェクト	backgroundColor , clickAdvances , cursor , defaultTransition , escapeExits , isFullScreen , loop , timeDelay , transitions , usePageTiming , useTimer	
Global オブジェクト		subscribe()
Identity オブジェクト	corporation , email , loginName , name	
Index オブジェクト	available , name , path , selected	
PlugIn オブジェクト	certified , loaded , name , path , version	
PPKLite 署名ハンドラオブジェクト	appearances , isLoggedIn , loginName , loginPath , name , signInvisible , signVisible , uiName	login() , logout() , newUser() , setPasswordTimeout()
Report オブジェクト	size , absIndent , color	Report() , writeText() , breakPage() , divide() , indent() , outdent() , open() , save() , mail()
Search オブジェクト	available , indexes , matchCase , maxDocs , maxDocs , proximity , proximity , refine , soundex , stem	addIndex() , getIndexForPath() , query() , removeIndex()
Security オブジェクト	handlers , validateSignaturesOnOpen	getHandler()

オブジェクト	プロパティ	メソッド
Spell オブジェクト	available , dictionaryNames , dictionaryOrder , domainNames	addDictionary() , addWord() , check() , checkText() , checkWord() , removeDictionary() , removeWord() , userWords()
Statement オブジェクト	columnCount , rowCount	execute() , getColumn() , getColumnArray() , getRow() , nextRow()
Template オブジェクト	hidden	spawn()

5.0 のその他の変更

このマニュアルには [Acrobat JavaScript に関する簡単な FAQ](#) という詳しい付録が含まれています。「[どうすればフォームフィールドをプログラムで作成できますか？](#)」、「[クイックリファレンス：フォーム](#)」、および「[どうすれば注釈をプログラムで作成できますか？](#)」の節では、フォームフィールドおよび注釈の作成に関連する数多くのプロパティやメソッド（新旧両バージョン）の使用法をまとめており、特に重要です。これらの節には、多くの例やプログラミングのヒントも網羅されています。

上記の表に示した新規のオブジェクト、プロパティおよびメソッドの他にも、さまざまな変更や拡張が加えられているので注意してください。

新機能：コンソールをエディタのように使用して、JavaScript コードを実行できるようになりました。「[対話型 JavaScript コンソール](#)」の節を参照してください。

変更／拡張：次のプロパティおよびメソッドが拡張されました。App.[language](#)、App.[execMenuItem](#)、Event.[type](#)、Doc.[exportAsFDF](#)、Doc.[print](#)、Doc.[submitForm](#)、Field.[buttonImportIcon](#)、Field.[textFont](#)、Field.[getItemAt](#)、Field.[value](#)、Util.[printd](#)。[Event オブジェクト](#)に関する節が大幅に変更され、Acrobat JavaScript のイベントモデルが分かりやすくなりました。

非推奨：次のプロパティおよびメソッドの使用は推奨されません。App.[fullscreen](#)、App.[numPlugins](#)、App.[getNthPluginName](#)、Doc.[author](#)、Doc.[creationDate](#)、Doc.[creator](#)、Doc.[getNthTemplate](#)、Doc.[keywords](#)、Doc.[modDate](#)、Doc.[numTemplates](#)、Doc.[producer](#)、Doc.[spawnPageFromTemplate](#)、Doc.[title](#)、Field.[hidden](#)、Tts.[soundCues](#)、Tts.[speechCues](#)。

5.05 の変更

Acrobat™ Approval™ で使用できないメソッドを示す新しいマークが、クイックバーに追加されました。


新しいドキュメントプロパティである [disclosed](#) の説明が追加されました。

ADBC オブジェクト

5.0			⊗
-----	--	--	---

Acrobat Database Connectivity (ADBC) プラグインでは、異なるプラットフォーム間で同一のオブジェクトモデルを使用し、PDF ドキュメント内の JavaScript からデータベースにアクセスすることができます。ADBC オブジェクトモデルは、ODBC および JDBC API のオブジェクトモデルで使用されている一般原則を基にしており、ODBC や JDBC と同様、SQL (Structured Query Language : 構造化照会言語) を介してデータベースとやり取りをするための手段です。

ADBCはWindows 専用の機能で、クライアントマシンにODBC(Microsoft社のOpen Database Connectivity) がインストールされている必要があります。

セキュリティ  : ADBC では、データベースアクセスのセキュリティ機能が提供されないので注意してください。データのセキュリティについては、データベース管理者が責任を持つ必要があります。

ADBC オブジェクトはグローバルオブジェクトであり、JavaScript でこのオブジェクトのメソッドを使用して、データベースへの接続を確立することができます。以下に説明する ADBC オブジェクトの他にも、関連するオブジェクトがいくつかあります。

オブジェクト	簡単な説明
ADBC オブジェクト	このオブジェクトでは、アクセス可能なデータベースのリストを取得して、そのいずれかのデータベースに接続することができます。
Connection オブジェクト	このオブジェクトでは、接続済のデータベースに含まれるテーブルのリストを取得することができます。
Statement オブジェクト	このオブジェクトでは、SQL ステートメントを実行して、クエリに基づいたレコードの抽出をすることができます。

ADBC プロパティ

SQL 型

5.0			⊗
-----	--	--	---

ADBCオブジェクトには、さまざまなSQL型を表すいくつかの定数のプロパティがあります。

名前	値	名前	値
SQLT_BIGINT	0	SQLT_LONGVARCHAR	10
SQLT_BINARY	1	SQLT_NUMERIC	11
SQLT_BIT	2	SQLT_REAL	12
SQLT_CHAR	3	SQLT_SMALLINT	13
SQLT_DATE	4	SQLT_TIME	14
SQLT_DECIMAL	5	SQLT_TIMESTAMP	15
SQLT_DOUBLE	6	SQLT_TINYINT	16
SQLT_FLOAT	7	SQLT_VARBINARY	17
SQLT_INTEGER	8	SQLT_VARCHAR	18
SQLT_LONGVARBINARY	9		

[Column オブジェクト](#)の `type` プロパティと [ColumnInfo オブジェクト](#)の `type` プロパティは、どちらも上記の SQL 型のプロパティを返します。

JavaScript 型

5.0			⊗
-----	--	--	---

ADBC オブジェクトには、JavaScript のデータ型を表す定数プロパティがいくつかあります。

名前	値	名前	値
Numeric	0	Time	4
String	1	Date	5
Binary	2	TimeStamp	6
Boolean	3		

[Statement オブジェクト](#)の `getColumn` および `getColumnArray` メソッドは、どちらも上記のデータ型を使用します。

ADBC メソッド

getDataSourceList

5.0			⊗
-----	--	--	---

パラメータ：なし

戻り値：配列

`getDataSourceList` メソッドは、システムでアクセス可能なデータベースに関する情報を取得します。パラメータはなく、各データベースの [DataSourceInfo オブジェクト](#) を含む配列を返します（配列が空の場合もあります）。このメソッドが失敗することはありませんが、配列長がゼロの場合もあります。

次の表に、[DataSourceInfo オブジェクト](#) のプロパティを示します。

DataSourceInfo オブジェクト			
DataSourceInfo オブジェクトは、特定のデータベースに関する基本情報を含むオブジェクトです。			
プロパティ	型	アクセス	説明
name	文字列	R	データベースの識別名を表す文字列です。この文字列を newConnection に渡すと、DataSourceInfo オブジェクトに関連付けられているデータベースに接続することができます。
description	文字列	R	データベース固有の情報を含みます。

例については、[newConnection](#) を参照してください。

newConnection

5.0			⊗
-----	--	--	---

パラメータ：`cDSN`、`[cUID]`、`[cPWD]`

戻り値：Connection オブジェクトまたは `null`

`newConnection` メソッドは、`cDSN`（データソース名）パラメータで識別されるデータベースの [Connection オブジェクト](#) を作成します。オプションとして、`cUID` パラメータにユーザ ID、`cPWD` にパスワードを指定することもできます。このメソッドは成功すると Connection オブジェクトを返し、失敗すると `null` を返します。

例：

```
// 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");

```

Annot オブジェクト

Acrobat 注釈プラグインの機能は、Annot オブジェクトを介して JavaScript メソッドに公開されます。

Annot オブジェクトは、特定の Acrobat 注釈（Acrobat の注釈ツールあるいは [Doc オブジェクト](#) の [addAnnot](#) メソッドを使用して作成した注釈）を表します。有効な注釈タイプは、*Circle*、*FileAttachment*、*FreeText*、*Highlight*、*Ink*、*Line*、*Sound*、*Square*、*Squiggly*、*Stamp*、*StrikeOut*、*Text*、および *Underline* です（注釈タイプ *Squiggly* には JavaScript からしかアクセスできないので、ユーザインタフェイスはありません）。

注釈の作成に使用する [addAnnot](#) に加え、ドキュメントから注釈を取得する [getAnnot](#) および [getAnnots](#) という他の [Doc オブジェクト](#) メソッドもあります。

「[どうすれば注釈をプログラムで作成できますか？](#)」の節では、ほぼすべてのタイプの注釈を JavaScript で作成する方法を、例とともに詳しく説明しています。

注意： Acrobat のユーザインタフェイスでは、さまざまなタイプの注釈 (*annotations*) を総称して注釈 (*comments*) と呼びます。

JavaScript による注釈へのアクセス

注釈にアクセスするには、[Doc オブジェクト](#) メソッドのインタフェイスに用意されているメソッドを使用し、注釈を JavaScript 変数に結び付ける必要があります。

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

この例では、1 ページ目（0 ベースのページ番号システムなので）にある「Important」という名前の注釈を、変数 *a* を介して、スクリプトで操作できるようにしています。したがって、たとえば次のような処理が可能です。

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

このコードでは、まず注釈タイプを変数 *thetype* に保存してから、作成者を「John Q. Public」に変更しています。

Annot オブジェクト：クイックリファレンス	
注釈タイプ	プロパティ
全てのタイプ	type , rect , page , author , name , contents , modDate
Circle	point , popupRect , fillColor , strokeColor , width

Annot オブジェクト : クイックリファレンス	
注釈タイプ	プロパティ
FileAttachment	print , attachIcon
FreeText	alignment , fillColor , rotate , strokeColor , textFont , textSize , width
Highlight	quads , strokeColor , point , popupRect
Ink	gestures , strokeColor , point , popupRect , width
Line	points , arrowBegin , arrowEnd , point , popupRect , fillColor , strokeColor , width
Sound	print , soundIcon
Square	point , popupRect , fillColor , strokeColor , width
Squiggly	quads , strokeColor , point , popupRect
Stamp	point , popupRect , AP
StrikeOut	quads , strokeColor , point , popupRect
Text	print , notelcon , point , popupRect
Underline	quads , strokeColor , point , popupRect

Annot プロパティ

alignment

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

型 : 数値

注釈 : FreeText

アクセス : R/W

このプロパティは、FreeText 注釈のテキストの整列を制御します。

整列	値
左揃え	0
中央揃え	1
右揃え	2

alignment プロパティの使用例については、「[FreeText 注釈](#)」の節を参照してください。

AP

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

型 : 文字列

注釈 : Stamp

アクセス : R/W

このプロパティの値は、Stamp 注釈のスタンプ表示に使用される外観の内部名です。Stamp 注釈の標準の名前には、「Approved」、「AsIs」、「Confidential」、「Departmental」、「Draft」、「Experimental」、「Expired」、「Final」、「ForComment」、「ForPublicRelease」、「NotApproved」、「NotForPublicRelease」、「Sold」、「TopSecret」があります。

例 :

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

注意 : 特定のスタンプ名を知りたい場合は、Stamps フォルダにある目的のスタンプを含む PDF ファイルを開きます。そして、ツール/フォーム/ページのテンプレートを選択すると、すべてのスタンプの外観名と内部名が表示されます。ドキュメントで現在使用されているスタンプ名のリストについては、[Doc オブジェクト](#)の [icons](#) プロパティを参照してください。

arrowBegin

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

型 : 文字列

注釈 : Line

アクセス : R/W

arrowBegin の値は、*Line* 注釈の始点のスタイルを示します。有効な値は、「Circle」、「ClosedArrow」、「Diamond」、「None」（デフォルト）、「OpenArrow」、「Square」です。「[Line 注釈](#)」の節を参照してください。また、[arrowEnd](#)、および例については [setProps](#) を参照してください。

arrowEnd

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

型：文字列

注釈：Line

アクセス：R/W

arrowEnd の値は、*Line* 注釈の終点のスタイルを示します。有効な値は、「Circle」、「ClosedArrow」、「Diamond」、「None」（デフォルト）、「OpenArrow」、「Square」です。「[Line 注釈](#)」の節を参照してください。また、[arrowBegin](#)、および例については [setProps](#) を参照してください。

attachIcon

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


型：文字列

注釈：FileAttachment

アクセス：R/W

このプロパティの値は、注釈の表示に使用するアイコン名です。認識される値には、「Paperclip」、「PushPin」（デフォルト）、「Graph」、「Tag」があります。

author

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

型：文字列


注釈：すべて

アクセス：R/W

注釈の作成者です。

使用例については、[contents](#) を参照してください。

contents

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

型：文字列

注釈：すべて

アクセス：R/W

ポップアップ表示可能な注釈の内容には、このプロパティからアクセスできます。*Sound* および *FileAttachment* 注釈の場合は、サウンドまたは添付ファイルの説明として表示されるテキストにアクセスします。

例：

```
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"
});
```

[addAnnot](#) メソッドも参照してください。

doc

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

型：Doc オブジェクト

注釈：すべて

アクセス：R

このプロパティは、注釈が存在するドキュメントの [Doc オブジェクト](#) を返します。

例：

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

fillColor

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

型：色



注釈：Circle、Square、Line、FreeText

アクセス：R/W

このプロパティは、Circle、Square、Line、および FreeText 注釈の背景色を示します。値は透明、グレー、RGB、または CMYK カラーのいずれかを使用して定義することができます。カラー配列の定義およびこのプロパティでの値の使用方法については、[カラー配列](#)の節を参照してください。

例については、[Circle および Square 注釈](#)を参照してください。

gestures

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


型：配列

注釈：Ink

アクセス：R/W

ストロークのパスを表す配列の配列です。各配列には、パスのポイントを示す[デフォルトユーザスペース](#)の x および y 座標が交互に格納されます。描画の際には、実装に依存した方法で、これらの点が直線または曲線で結ばれます。詳しくは、『[PDF Reference](#)』の 415 ページにある「Ink Annotations」を参照してください。
詳しい例については、[Ink 注釈](#)を参照してください。

hidden

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



型：ブーリアン

注釈：すべて

アクセス：R/W

`hidden` が `true` に設定されると注釈は表示されなくなり、ユーザは注釈を操作、表示、印刷することができなくなります。

modDate

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

型：日付

注釈：すべて

アクセス：R

このプロパティは、注釈の最終変更日を返します。

例：

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

name

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

型：文字列

注釈：すべて



アクセス：R/W

注釈のプロパティおよびメソッドを検索してアクセスするには、[getAnnot](#) に注釈の名前を指定します。

例：

```
// 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		
-----	-----------------------------------------------------------------------------------	-----------------------------------------------------------------------------------

型：文字列



注釈：Text

アクセス：R/W

このプロパティの値は、注釈の表示に使用するアイコン名です。*notelcon* で認識される値には、「Comment」、「Help」、「Insert」、「Key」、「Note」（デフォルト）、「NewParagraph」、「Paragraph」があります。

「[Text 注釈](#)」の節を参照してください。例については、[contents](#) プロパティを参照してください。

noView

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

型：ブーリアン

注釈：すべて

アクセス：R/W

noView が *true* に設定されると注釈は非表示になりますが、注釈が外観を持っている場合、その外観を印刷することはできます。

page

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

型：整数

注釈：すべて


アクセス：R/W

このプロパティは、注釈が存在するページを示します。例えば、次のコードを実行したとします。

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

この場合、Annot オブジェクトである *annot* は、現在のページからページ 3（0 ベースのページ番号システムなので）に移動します。

point

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

型：配列

注釈：Text、Sound、FileAttachment

アクセス：R/W

2つの数値の配列 $[x_{ul}, y_{ul}]$ で、*Text*、*Sound*、または *FileAttachment* 注釈のアイコンの左上隅を、デフォルトユーザスペースで示します。

例：

```
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"
});
```

[addAnnot](#) および [noteIcon](#) も参照してください。

points

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

型：配列

注釈：Line

アクセス：R/W

2つの点の配列 $[[x_1, y_1], [x_2, y_2]]$ で、線の開始点と終了点を [デフォルトユーザスペース](#) の座標で示します。

例：

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

[addAnnot](#)、[arrowBegin](#)、[arrowEnd](#)、[setProps](#)、および [Line 注釈](#) を参照してください。



popupOpen

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

型：ブーリアン注釈：Sound、FreeText、FileAttachment を除くすべて アクセス：R/W

popupOpen が *true* に設定されると、注釈が存在するページ上にテキストノートがポップアップして開きます。

popupRect

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

型 : 配列 注釈 : FreeText、FileAttachment、Sound を除くすべて アクセス : R/W

$[x_{ll}, y_{ll}, x_{ur}, y_{ur}]$ という 4 つの数値で構成される配列です。注釈に関連付けられているポップアップ注釈の矩形を表し、ページ上の位置を決定します。値には、左下の x 、左下の y 、右上の x 、右上の y を[デフォルトユーザスペース](#)で指定します。


print

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

型 : ブーリアン 注釈 : すべて アクセス : R/W

`print` プロパティは、注釈を印刷するかどうかを示します。`true` の場合、注釈は印刷されます。

quads

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

型 : 配列 注釈 : Highlight、StrikeOut、Underline、Squiggly アクセス : R/W

$8 \times n$ の数値の配列で、 n 個の四辺形の[デフォルトユーザスペース](#)における座標を示します。各四辺形は、注釈を構成する 1 つ以上の連続する単語を囲みます。詳しくは、『[PDF Reference](#)』の 414 ページにある表 7.19 を参照してください。1 つの単語の四辺形は、[getNthTemplate](#) を呼び出して取得することができます。

例については、[getNthTemplate](#) を参照してください。

rect

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

型 : 配列 注釈 : すべて アクセス : RW

$[x_{ll}, y_{ll}, x_{ur}, y_{ur}]$ という 4 つの数値で構成される配列で、注釈の矩形を表し、ページ上の位置を決定します。値には、左下の x 、左下の y 、右上の x 、右上の y を[デフォルトユーザスペース](#)で指定します。[popupRect](#) も参照してください。

readOnly

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

型：ブーリアン

注釈：すべて

アクセス：R/W

`readOnly` が `true` に設定されると、注釈は表示専用となり、変更などはできなくなります。

rotate

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

型：整数

注釈：FreeText

アクセス：R/W

`rotate` プロパティは、ページ上の注釈を反時計回りに回転する角度（0、90、180、270）を示します。アイコンベースの注釈は回転しません。このプロパティは `FreeText` 注釈にのみ意味があります。

strokeColor

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

型：色

注釈：すべて

アクセス：R/W

このプロパティは、注釈の表示色を設定します。透明、グレー、RGB、または CMYK カラーを使用して、値を定義します。`FreeText` 注釈の場合、`strokeColor` は境界線およびテキストの色を示します。カラー配列の定義およびこのプロパティでの値の使用方法について詳しくは、「[カラー配列](#)」の節を参照してください。

例：

```
// Make a text note red
var annot = this.addAnnot({type: "Text", page:0});
annot.strokeColor = color.red;
```

textFont

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

型：文字列

注釈：FreeText

アクセス：R/W



`textFont` プロパティは、*FreeText* 注釈に入力されるテキストのフォントを設定します。有効なフォントは、Field オブジェクトの [textFont](#) プロパティのリストに示すように、「フォント」オブジェクトのプロパティとして定義されています。

FreeText 注釈で任意のフォントを使用したい場合は、`textFont` にフォントの PostScript 名を指定してください。

次の例に、このプロパティとフォントオブジェクトの使用法を示します。

```
// 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		
-----	-------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------



型：整数

注釈：FreeText

アクセス：R/W

このプロパティは、*FreeText* 注釈で使用するテキストのサイズ（ポイント）を示します。有効なサイズはゼロと 4 ～ 144 の範囲です。ゼロのサイズは、すべてのテキストデータが注釈の矩形に収まる最大のポイントサイズになることを意味します。例については、[textFont](#) を参照してください。

type

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

型：文字列



注釈：すべて

アクセス：R

このプロパティは、注釈のタイプを示します。有効なタイプは、「Circle」、「FileAttachment」、「FreeText」、「Highlight」、「Ink」、「Line」、「Sound」、「Square」、「Squiggly」、「Stamp」、「StrikeOut」、「Text」、「Underline」です。

注意： 注釈の「タイプ」は、[addAnnot](#) メソッドのオブジェクトリテラル引数内でのみ設定することができます。

soundIcon

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

型：文字列

注釈：Sound

アクセス：R/W

このプロパティの値は、注釈の表示に使用するアイコン名です。「Speaker」という値を使用することができます。

width

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

型：数値

注釈：Square、Circle、Line、Ink、FreeText

アクセス：R/W

境界線の幅をポイント単位で示します。この値が 0 の場合、境界線は描かれませんが、デフォルト値は 1 です。

Annot メソッド

destroy

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

パラメータ：なし

戻り値：なし

このメソッドは、注釈を破棄してページから削除します。オブジェクトは無効になります。

例：

```
// 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		
-----	-----------------------------------------------------------------------------------	-----------------------------------------------------------------------------------

パラメータ：なし

戻り値：オブジェクトリテラル

このメソッドは、注釈のプロパティのオブジェクトリテラルを返します。オブジェクトリテラルは、[addAnnot](#) に渡されるものと同じです。このメソッドは、注釈をコピーするために使用できます。

例：

```
var annot = this.addAnnot({
    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;
}
```

setProps

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

パラメータ：オブジェクトリテラル

戻り値：Annot オブジェクト

注釈の複数のプロパティを同時に設定します。オブジェクトリテラルは、[addAnnot](#) に渡されるものと同じです。

例：

```
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"
})
```

App オブジェクト

App オブジェクトは静的な JavaScript オブジェクトで、Acrobat 固有の数多くの関数に加え、さまざまなユーティリティ関数、および便利な関数を定義しています。

App オブジェクトのプロパティ

activeDocs

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

型 : 配列

アクセス : R

このプロパティは、ビューアで開いているアクティブなドキュメントのうち、[disclosed](#) プロパティが *true* に設定されているドキュメントの [Doc オブジェクト](#) の配列を返します。アクティブなドキュメントが存在しない場合は *activeDocs* から何も返されないか、コア JavaScript の `d=new Array(0)` と同じ動作になります。

このプロパティを使用すると、ドキュメントをまたがった操作が可能になります。

例 :

```
/* 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 */
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

☹			
---	--	--	--

型 : ブーリアン

アクセス : R/W

このプロパティが *true* の場合は計算を実行できます。*false* の場合、すべてのドキュメントで計算がまったく実行されなくなります。デフォルト値は *true* です。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [calculate](#) プロパティも参照してください。

focusRect

4.05			
------	-----------------------------------------------------------------------------------	--	--

型：ブーリアン

アクセス：R/W

このプロパティは、フォーカスを示す矩形のオン／オフを切り替えます。フォーカスを示す矩形とは、ボタン、チェックボックス、ラジオボタン、署名を囲む薄い点線のこと、フォームフィールドにキーボードフォーカスがあることを示すものです。

formsVersion

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

型：数値

アクセス：R

このプロパティは、ビューア内部で動作しているフォームソフトウェアのバージョン番号を示します。スクリプトで下位互換性を維持したい場合、新しいバージョンのオブジェクト、プロパティ、またはメソッドが使用できるかどうかを判断するために、このメソッドを使用することができます。詳しくは、「[このマニュアルで使用する表記](#)」を参照してください。

例：

```
if (typeof app.formsVersion != "undefined" && app.formsVersion >= 4.0) {  
    // Perform version specific operations here.  
}
```

fs

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

型：オブジェクト

アクセス：R

[FullScreen オブジェクト](#)が返されます。これを使用して全画面表示に関連するプロパティにアクセスすることができます。

例：

```
// This code puts the viewer into fullscreen (presentation) mode.  
app.fs.isFullScreen = true;
```

[FullScreen オブジェクト](#)の [isFullScreen](#) も参照してください。

fullscreen

			
-------------------------------------------------------------------------------------	--	--	--

型：ブーリアン

アクセス：R/W

このプロパティは、Acrobat ビューアの通常の表示モードと全画面表示モードを切り替えます。

例：

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

上の例で、`app.fullscreen` が `true` に設定されると、Adobe Acrobat ビューアが全画面表示モードになります。`app.fullscreen` が `false` の場合、通常の表示モードになります。この場合の通常の表示モードとは、Acrobat アプリケーションが全画面表示モードになる前の状態を指します。

注意： Web ブラウザで表示されている PDF ドキュメントを全画面表示モードにすることはできません。ブラウザの中から全画面表示モードを実行することはできますが、Acrobat ビューアアプリケーションでドキュメントが開かれていなければ何も起こりません。Acrobat ビューアアプリケーションでドキュメントが開かれている場合、ビューアで開かれているドキュメントが全画面表示になりますが、Web ブラウザで開いている PDF ドキュメントはそのままです。

[FullScreen オブジェクト](#)の `isFullScreen` プロパティを参照してください。今後のバージョンでは `App.fullscreen` に代わってこのプロパティが使用されます。`App.fs` も参照してください。`App.fs` は [FullScreen オブジェクト](#)を返しますが、このオブジェクトを使用して全画面表示に関連するプロパティにアクセスすることができます。

language

型：文字列

アクセス：R

このプロパティは、動作中の Acrobat ビューアで使用されている言語を取得します。次の文字列が返されます。

文字列	言語	文字列	言語
CHS	簡体字中国語	KOR	韓国語
CHT	繁体字中国語	JPN	日本語
DAN	デンマーク語	NLD	オランダ語
DEU	ドイツ語	NOR	ノルウェー語
ENU	英語	PTB	ポルトガル語（ブラジル）
ESP	スペイン語	SUO	フィンランド語
FRA	フランス語	SVE	スウェーデン語
ITA	イタリア語		

numPlugins

☹			
---	--	--	--

型 : 数値

アクセス : R

このプロパティは、Acrobat にロードされているプラグインの数を示します。今後のバージョンでこのプロパティは使用されなくなります。このプロパティに代わって使用される [plugins](#) プロパティを参照してください。

openInPlace

5.0	👤		
-----	---	--	--

型 : ブーリアン

アクセス : R/W

このプロパティは、ドキュメント間のリンクが同じウィンドウで開かれるのか新規のウィンドウで開かれるのかを示します。

platform

型 : 文字列

アクセス : R

このプロパティは、スクリプトを実行中のプラットフォームを返します。値は「WIN」、「MAC」、「UNIX」のいずれかになります。

plugins

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

型 : 配列

アクセス : R

ビューアに現在インストールされているプラグインを判断するのに使用できます。 [PlugIn オブジェクト](#) の配列を返します。

例 :

```
// 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);
```

toolbar



型：ブーリアン

アクセス：R/W

このプロパティでは、Acrobat の水平および垂直ツールバーの両方を、スクリプトで表示したり隠したりできます。外部ウィンドウ（Web ブラウザ内の Acrobat ウィンドウなど）では、ツールバーを隠しません。

例：

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

toolbarHorizontal



型：ブーリアン

アクセス：R/W

このプロパティでは、Acrobat の水平方向のツールバーをスクリプトで表示したり隠したりできます。外部ウィンドウ（Web ブラウザ内の Acrobat ウィンドウなど）では、ツールバーを隠しません。

注意： Acrobat 5.0 では、ツールバーの概念およびアプリケーションフレーム内で配置可能な場所が大幅に変更されました。したがって、このプロパティの使用は推奨されなくなりました。このプロパティを使用すると、[toolbar](#) プロパティのように動作します。

toolbarVertical



型：ブーリアン

アクセス：R/W

このプロパティでは、Acrobat の垂直方向のツールバーをスクリプトで表示したり隠したりできます。外部ウィンドウ（Web ブラウザ内の Acrobat ウィンドウなど）では、ツールバーを隠しません。

注意： Acrobat 5.0 では、ツールバーの概念およびアプリケーションフレーム内で配置可能な場所が大幅に変更されました。したがって、このプロパティの使用は推奨されなくなりました。このプロパティを使用すると、[toolbar](#) プロパティのように動作します。

viewerType

型：文字列

アクセス：R

このプロパティは、動作中の Acrobat ビューアが Reader であるか標準の製品であるかを示します。値はそれぞれ「Reader」または「Exchange」です。

viewerVariation

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

型：文字列

アクセス：R

このプロパティは、動作中の Acrobat ビューアのパッケージを示します。値は、「Reader」、「Fill-In」、「Business Tools」、または「Full」のいずれかになります。

viewerVersion

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

型：数値

アクセス：R

このプロパティは、現在のビューアのバージョン番号を示します。

App オブジェクトのメソッド

addMenuItem

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

パラメータ：cName、[cUser]、cParent、[nPos]、cExec、[cEnable]、[cMarked]

戻り値：なし

アプリケーションにメニュー項目を追加します。

cName には、言語に依存しないメニュー項目名を指定します。この言語に依存しない名前は、他のメソッド ([hideMenuItem](#) など) でメニュー項目にアクセスする際に使用されます。

cUser は、ユーザにメニュー項目の名前として表示される文字列（言語に依存しない名前）です。cUser が指定されていない場合は、cName が使用されます。

cParent は、親メニュー項目の名前です。新しいメニュー項目は、このサブメニューに追加されます。cParent にサブメニューが存在しない場合は、例外が発生します。

メニュー項目名は、[listMenuItems](#) メソッドを使用して知ることができます。また、言語に依存しないメニュー項目名は、『Acrobat Viewer plug-in API On-Line Reference』にも記載されています ([参考資料](#)を参照)。

nPos は、メニュー項目を追加するサブメニュー内の位置を示します。デフォルトではサブメニューの最後に追加され、*nPos* に 0 を指定するとサブメニューの先頭に追加されます。

*cExec*には、ユーザがメニュー項目を選択したときに評価する式を文字列として指定します。


cEnable には、メニュー項目を使用可能にするかどうかを指定する式を文字列として指定します。デフォルトでは、メニュー項目は常に使用可能になります。メニュー項目を使用不可にするには、この式で *event.rc* を *false* に設定してください。

cMarked には、メニュー項目の横にチェックマークを付けるかどうかを指定する式を文字列として指定します。デフォルトでは、メニュー項目にマークは付きません。メニュー項目のチェックマークを解除するには、この式で *event.rc* を *false* に設定し、チェックマークを付けるには *true* に設定してください。

例：

```
// 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});
```

[addSubMenu](#)、[execMenuItem](#)、[hideMenuItem](#)、および [listMenuItems](#) メソッドも参照してください。

セキュリティ ：このメソッドは、アプリケーションの初期化時またはコンソールイベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

addSubMenu

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

パラメータ：*cName*、*[cUser]*、*cParent*、*[nPos]*
戻り値：なし

サブメニューを持つメニュー項目をアプリケーションに追加します。

cName には、言語に依存しないメニュー項目名を指定します。この言語に依存しない名前は、[hideMenuItem](#) など、メニュー項目にアクセスする際に使用されます。

cUser は、ユーザにメニュー項目の名前として表示される文字列（言語に依存しない名前）です。*cUser* が指定されていない場合は、*cName* が使用されます。

cParent は、サブメニューを追加する親メニュー項目の名前です。


メニュー項目名は、[listMenuItems](#) メソッドを使用して知ることができます。また、言語に依存しないメニュー項目名は、『*Acrobat Viewer plug-in API On-Line Reference*』にも記載されています（[参考資料](#)を参照）。

nPos は、サブメニューを追加する親のサブメニュー内の位置を示します。デフォルトでは親のサブメニューの最後に追加され、*nPos* に 0 を指定すると親のサブメニューの先頭に追加されます。

例：

```
// 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});
```

[addMenuItem](#)、[execMenuItem](#)、[hideMenuItem](#)、および [listMenuItems](#) メソッドも参照してください。

セキュリティ ：このメソッドは、アプリケーションの初期化時またはコンソールイベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

alert

パラメータ：*cMsg*、*[nIcon]*、*[nType]*

戻り値：*nButton*

このメソッドは、警告ダイアログを画面に表示します。パラメータとして *cMsg* は最低限必要で、ここに表示するメッセージの文字列を指定します。オプションの *nIcon* パラメータを使用すると、アイコンタイプも指定できます。以下は、アイコンおよびその関連値のリストです。

アイコン	値
警告（デフォルト）	0
注意	1
問い合わせ	2
情報	3

注意： *Macintosh OS* では注意と問い合わせが区別されないので、3 種類のアイコンのみとなります。

また、*nType* を使用すると、ボタンの組み合わせも指定できます。

ボタンの組み合わせ	値
OK (デフォルト)	0
OK、キャンセル	1
はい、いいえ	2
はい、いいえ、キャンセル	3

このメソッドは、ユーザがクリックしたボタンの種類を返します。

ボタンの種類	値
OK	1
キャンセル	2
いいえ	3
はい	4

beep

パラメータ : *[nType]*

戻り値 : なし

このメソッドは、システムの音を鳴らします。音の種類および値は、次の通りです。

メッセージの種類	値
警告	0
注意	1
問い合わせ	2
情報	3
デフォルト (デフォルト)	4

注意 : *Apple Macintosh* および *UNIX* システムでは、音の種類は無視されます。

clearInterval

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

パラメータ : *oInterval*

戻り値 : なし

このメソッドは、登録されているインターバル *oInterval* をキャンセルします。インターバルは [setInterval](#) メソッドで最初の設定をすることができます。

[setTimeout](#) および [clearTimeout](#) メソッドも参照してください。使用例は、[setTimeout](#) メソッドの説明の後にあります。

clearTimeout

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

パラメータ : *oTime*

戻り値 : なし

このメソッドは、登録されているタイムアウトインターバル *oTime* をキャンセルします。インターバルは [setTimeout](#) メソッドで最初の設定をすることができます。

[setInterval](#) および [clearInterval](#) メソッドも参照してください。使用例は、[setTimeout](#) メソッドの説明の後にあります。

execMenuItem

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

パラメータ : *cMenuItem*

戻り値 : なし

このメソッドは、指定のメニュー項目を実行します。

メニュー項目名は、[listMenuItems](#) メソッドを使用して知ることができます。言語に依存しないメニュー項目名は、『*Acrobat Viewer plug-in API On-Line Reference*』にも記載されています（[参考資料](#)を参照）。


例 :

```
/* This example executes File->Open menu item. It will display a dialog to the
** user asking for the file to be opened. */
app.execMenuItem("Open");
```

[addMenuItem](#)、[addSubMenu](#)、および [hideMenuItem](#) メソッドも参照してください。[listMenuItems](#) メソッドを使用すると、メニュー項目がすべてコンソールに表示されるので便利です。

5.0	追記事項
-----	------

App.execMenuItem("SaveAs") を実行すると「名前を付けて保存」ダイアログが開くので、ここでフォルダ名とファイル名を指定し、現在のファイルをユーザのハードドライブに保存することができます。編集／環境設定／一般／オプションダイアログの「Web 表示用に最適化して保存」にチェックマークが付いている場合は、保存時にファイルが最適化されます。

セキュリティ  : App.execMenuItem("SaveAs") は、バッチ、コンソール、またはメニューイベントでのみ実行できます。.Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

注意： 「Web 表示用に最適化して保存」がチェックされている場合、「SaveAs」の実行後にフォームオブジェクトは消えてしまいます。Field オブジェクトも無効になりますから、「SaveAs」の直後に Field オブジェクトにアクセスしようとすると、例外が発生します。以下の例を参照してください。

例：

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

例：

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

注意： セキュリティ上の理由で、スクリプトから「終了」メニュー項目を実行することはできません。

getNthPluginName



パラメータ : *nIndex*

戻り値 : *cName*

このメソッドは、ビューアにロードされている *n* 番目のプラグイン名を返します。[numPlugins](#) プロパティも参照してください。

今後のバージョンでこのプロパティに代わって使用される [plugins](#) プロパティも参照してください。

goBack

パラメータ：なし
戻り値：なし

この関数は、ビュースタックにある前のビューに戻ります。Acrobat ツールバーの「前の画面」ボタンを押す操作と同じです。

goForward

パラメータ：なし
戻り値：なし

この関数は、ビュースタックにある次のビューに進みます。Acrobat ツールバーの「次の画面」ボタンを押す操作と同じです。

hideMenuItem


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

パラメータ：cName
戻り値：なし

このメソッドでは、cName に指定したメニュー項目を削除し、Acrobat ビューアの外観をカスタマイズすることができます。

メニュー項目名は、[listMenuItems](#) メソッドを使用して知ることができます。言語に依存しないメニュー項目名は、『*Acrobat Viewer plug-in API On-Line Reference*』（テクニカルノート #5191）にも記載されています。[参考資料](#)を参照してください。

[addMenuItem](#)、[addSubMenu](#)、[execMenuItem](#)、および [listMenuItems](#) メソッドも参照してください。

セキュリティ ：このメソッドは、アプリケーションの初期化時またはコンソールイベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

hideToolbarButton

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

パラメータ：cName
戻り値：なし


このメソッドでは、*cName* に指定したツールバーボタンを削除し、Acrobat ビューアの外観をカスタマイズすることができます。

ツールバーボタン名は、[listToolBarButtons](#) メソッドを使用して知ることができます。言語に依存しないツールバーボタン名は、『*Acrobat Viewer plug-in API On-Line Reference*』（テクニカルノート #5191）にも記載されています。[参考資料](#)を参照してください。

例：以下のスクリプトを含む *myConfig.js* という名前のファイルを作成します。

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

そしてこのファイルを[フォルダレベルの JavaScript](#) フォルダに入れて Acrobat ビューアを起動すると、「手のひら」アイコンが表示されなくなります。

セキュリティ ：このメソッドは、アプリケーションの初期化時またはコンソールイベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

listMenuItems

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

パラメータ：なし
戻り値：なし

アプリケーションのメニュー項目名をすべてコンソールに表示します。スクリプトの作成やデバッグの役に立ちます。

言語に依存しないメニュー項目名は、『*Acrobat Viewer plug-in API On-Line Reference*』（テクニカルノート #5191）にも記載されています。[参考資料](#)を参照してください。

例：すべてのメニュー項目の名前をコンソールに表示します。

```
console.show();  
app.listMenuItems();
```

[addItem](#)、[addSubMenu](#)、[execMenuItem](#)、および [hideMenuItem](#) メソッドも参照してください。

listToolBarButtons

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

パラメータ：なし
戻り値：なし

アプリケーションのツールバーボタン名をすべてコンソールに表示します。スクリプトの作成やデバッグの役に立ちます。

言語に依存しないメニュー項目名は、『*Acrobat Viewer plug-in API On-Line Reference*』（テクニカルノート #5191）にも記載されています。[参考資料](#)を参照してください。

[hideToolBarButton](#) メソッドも参照してください。

mailMsg

4.0			ⓧ
-----	--	--	---

パラメータ : *bUI*、*cTo*、*[cCc]*、*[cBcc]*、*[cSubject]*、*[cMsg]*
戻り値 : なし

このメソッドは、電子メールメッセージを送信します。ユーザとのやり取りの有無は、*bUI* の値によって決まります。このパラメータが *true* に設定されていると、メーラーの新規メッセージウィンドウがユーザに表示され、他のパラメータ値がセットされます。

bUI が *false* の場合、必須パラメータは *cTo* のみで、他のパラメータはオプションになります。*cTo*、*cCc*、*cBcc* パラメータに複数の宛先を指定する場合は、セミコロン「;」で区切ってください。*cSubject* および *cMsg* の長さは 64K バイトまで有効です。

例 :

```
/* 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.");
/* Or the same message can be sent as follows: */
app.mailMsg( {bUI: false,
               cTo: "fun1@fun.com; fun2@fun.com",
               cSubject: "This is the subject",
               cMsg: "This is the body of the mail."} );
```

注意 : Windows の場合 : このメソッドを使用するには、クライアントマシンでのデフォルトのメールプログラムを MAPI 対応に設定しておく必要があります。

newDoc

5.0		🔑	ⓧ
-----	--	---	---


パラメータ : *[nWidth]*、*[nHeight]*
戻り値 : *Doc* オブジェクト

このメソッドは、Acrobat ビューアで新規のドキュメントを作成し、その [Doc オブジェクト](#) を返します。オプションのパラメータである *nWidth* と *nHeight* には、ドキュメントのメディアボックスの寸法をポイント単位で指定します。デフォルト値は、*nWidth* = 612 および *nHeight* = 792 です。

例：Acrobat のファイルメニューに「New」というメニュー項目を追加し、そこに「Letter」、「A4」、「Custom」という 3 つのサブメニュー項目を含めます。このスクリプトを [フォルダレベルの JavaScript](#) フォルダに入れてください。

```
app.addSubMenu({ cName: "New", cParent: "File", nPos: 0 })
app.addMenuItem({ cName: "Letter", cParent: "New", cExec:
  "var d = 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 })" });
```

このコードはやや不完全です。「Custom」というメニュー項目の場合、コードを追加して、空の文字列や有効範囲外の値を入力できないようにしたほうが良いでしょう。現時点における PDF の制約について詳しくは、『[PDF Reference](#)』の 546 ページにある「General Implementation Limits」を参照してください。

セキュリティ ：このメソッドは、バッチ、コンソール、またはメニューイベントでのみ実行できます。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#) を参照してください。

openDoc

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

パラメータ：*cPath*、*[oDoc]*
戻り値：Doc オブジェクト

このメソッドは、*cPath* に指定した PDF ドキュメントを開いて、開いたドキュメントの [Doc オブジェクト](#) を返します。ドキュメントの [disclosed](#) が true でない場合は null が返されます。このメソッドで返される [Doc オブジェクト](#) を使用して、開いたドキュメントに関連する他のメソッドを使用したり、プロパティの値を取得、設定することができます。

cPath には、開くドキュメントのパスをデバイスに依存しない形式で指定します。2 番目のパラメータ *oDoc* を指定した場合は、相対パスを指定することができます。*oDoc* および対象ドキュメントは、どちらもデフォルトのファイルシステムに存在していなければなりません。

`oDoc`には、相対パス `cPath` の解決をするためにベースとする [Docオブジェクト](#) を指定します。

例：

```
/* 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();
```

以下は、上と同じ例で相対パスを使用した場合です。

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

注意： 現在のドキュメントおよび対象ドキュメントは、どちらもデフォルトのファイルシステムに存在する必要があります。

[closeDoc](#) および [setFocus](#) メソッドも参照してください。

popUpMenu

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

パラメータ：*cItem* または配列 ...
戻り値：*cItem*

このメソッドは、現在のマウス位置にポップアップメニューを作成します。メニューには引数で指定した数の項目が含まれ、メニューを選択するとそのメニュー項目の名前が返されます。「-」というメニュー項目名は、メニューのセパレータとして予約されています。

引数が文字列の場合は、メニュー項目としてメニューに表示されます。

引数が配列の場合、配列の最初の要素が親メニュー項目となるサブメニューとして表示されます。必要に応じて、配列中にさらにサブメニューを含めることができます。

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

注意： マウスとポップアップメニューとの間の通信がプラットフォームに依存している場合は、このメソッドを [Field / Mouse Down](#) イベントでのみ呼び出すようにしてください。それ以外のタイミングで実行した場合、一部のプラットフォームで動作しない可能性があります。

response

パラメータ : *cQuestion*、*[cTitle]*、*[cDefault]*、*[bPassword]*

戻り値 : *cResponse*、キャンセルの場合は *null*

このメソッドで表示されるダイアログには、質問と、それに応答するための入力フィールドが含まれます。

cQuestion は、ユーザに表示される質問です。

cTitle はオプションで、ダイアログのウィンドウタイトルに表示されるタイトルです。

cDefault は、質問に対する応答のデフォルト値です。指定がない場合、デフォルト値は表示されません。

bPassword を *true* に設定すると、ユーザが入力する内容をアスタリスク (*) または黒丸 (i) でマスクし、他人に見られないようにできます。

戻り値は、ユーザの応答を含む文字列です。ユーザがダイアログの「キャンセル」ボタンをクリックすると、応答は *null* オブジェクトになります。

例：

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

setInterval

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

パラメータ : *cExpr*、*nMilliseconds*

戻り値 : タイムアウトオブジェクト

このメソッドは、指定時間（ミリ秒単位）おきに評価される式を登録します。例えば、「Color」という名前のフィールドで、1 秒ごとに色が変わる単純なアニメーションを作成するには、次のようにします。

```
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;
}
timeout = app.setInterval("DoIt()", 1000);
// Add a property to our timeout object so that DoIt() can keep a count going.
timeout.count = 0;
```

[clearInterval](#)、[setTimeout](#)、および [clearTimeout](#) メソッドも参照してください。他の例については、[setTimeout](#) メソッドを参照してください。

setTimeout

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

パラメータ : *cExpr*、*nMilliseconds*
 戻り値 : タイムアウトオブジェクト

このメソッドは、指定時間（ミリ秒単位）が経過した後で評価される式を登録します。

[clearTimeout](#)、[setInterval](#)、および [clearInterval](#) メソッドも参照してください。

例：この例では、単純なマーキーを作成して文字列をスクロールさせます。「marquee」という名前のテキストフィールドがあり、そのデフォルト値は「Adobe Acrobat version 5.0 will soon be here!」であるとします。

```
// 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) {}
```

この例では、*try/catch* で「Stop」ボタンのコードをより安全なものにしています。ユーザが「Go」ボタンを押さずに「Stop」ボタンを押すと、*run* および *stoprun* はまだ未定義であるため、「Stop」のコードで例外が発生します。しかし例外が発生した場合には *catch* コードが実行されるので、ユーザが最初に「Stop」ボタンを押しても問題は起きません。

Bookmark オブジェクト

Bookmark オブジェクトは、「しおり」タブに表示されるツリーのノードを表しています。通常、しおりは目的のトピックにすばやく移動するための目次として使用されます。

Bookmark オブジェクトのプロパティ

children

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

型 : 配列

アクセス : R

しおりツリーにおいて、該当するしおりの子の Bookmark オブジェクトで構成される配列を返します。[parent](#) プロパティおよび [Doc オブジェクト](#) の [bookmarkRoot](#) プロパティも参照してください。

例 :

```
/* Dump all bookmarks in the document. */
function DumpBookmark(bm, nLevel)
{
    var s = "";
    for (var i = 0; i < nLevel; i++)
        s += " ";
    console.println(s + "+-" + bm.name);
    if (bm.children != null)
        for (var i = 0; i < bm.children.length; i++)
            DumpBookmark(bm.children[i], nLevel + 1);
}

console.clear();
console.show();
console.println("Dumping all bookmarks in the document.");
DumpBookmark(this.bookmarkRoot, 0);
```

color

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

型 : 配列

アクセス : R/W

このプロパティは、しおりの色を示します。グレー、RGB、または CMYK カラーで値を定義します。カラー配列の定義およびこのプロパティでの値の使用方法について詳しくは、「カラー配列」の節を参照してください。[style](#) プロパティも参照してください。

例：次のスクリプトは、しおりのトップレベルの色を赤色、緑色、青色と変化させます。

```
var bm = bookmarkRoot.children[0]
bm.color = color.black;
var C = new Array(1, 0, 0);
var run = app.setInterval(
    'bm.color = ["RGB",C[0],C[1],C[2]]; C.push(C.shift());', 1000);
var stoprun=app.setTimeout(
    "app.clearInterval(run); bm.color=color.black",12000);
```

注意： このプロパティは、Acrobat Reader では読み取り専用です。

doc

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

型：オブジェクト

アクセス：R

このプロパティは、しおりが存在する [Doc オブジェクト](#) です。

name

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

型：文字列

アクセス：R/W

このプロパティは、「しおり」タブに表示されるしおりのテキスト文字列です。

open

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

型：ブーリアン

アクセス：R/W

このプロパティは、「しおり」タブで該当するしおりの子を表示する（開く）か、子のサブツリーを折りたたむ（閉じる）かを指定します。

parent

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

型：オブジェクトまたは null

アクセス：R

親のしおりを返します。親がない場合は *null* を返します。[children](#) プロパティおよび [Doc オブジェクト](#) の [bookmarkRoot](#) プロパティも参照してください。

style

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

型：整数

アクセス：R/W

このプロパティは、しおりのフォントスタイルを示します。0 は標準、1 はイタリック、2 はボールド、3 はボールドイタリックを表します。[color](#) プロパティも参照してください。

注意： このプロパティは、Acrobat Reader では読み取り専用です。

Bookmark オブジェクトのメソッド

createChild

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

パラメータ：cName、[cExpr]、[nIndex]

戻り値：なし

指定の位置に新しい子しおりを作成します。

cName は、「しおり」タブに表示するしおりの名前です。

cExpr は、ユーザがしおりをクリックするたびに評価される式で、デフォルトでは指定されていません。これは JavaScript アクションでしおりを作成する操作に相当します。詳しくは、『[PDF Reference](#)』の「JavaScript Action」を参照してください。

nIndex は、子しおりの配列（0 ベース）のインデックスで、新しい子を作成する位置を指定します。デフォルトは 0 です。

[children](#) プロパティ、および [insertChild](#)、[remove](#) メソッドも参照してください。

例：

```
// Create a bookmark at the top of the bookmark panel that takes you to the
// next page in the document.
bookmarkRoot.createChild("Next Page", "this.pageNum++");
```

execute

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

パラメータ：なし
戻り値：なし

しおりに関連するアクションを実行します。アクションにはさまざまな種類があります。一般的なアクションタイプの一覧は、『[PDF Reference](#)』の第 7.5.3 節「Actions Types」を参照してください。

[createChild](#) プロパティも参照してください。

insertChild

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

パラメータ：oBookmark、[nIndex]
戻り値：なし

指定のしおりを子として挿入します。指定のしおりが既にツリーに存在する場合は、一度ツリーから切り離してから挿入し直します。また、挿入の循環がチェックされ、循環している場合は挿入不可となります。これにより、しおりがそれ自体の子または孫として挿入されるのを防ぐことができます。

oBookmark は、子として追加する Bookmark オブジェクトです。

nIndex は、子しおりの配列（0 ベース）のインデックスで、新しい子を挿入する位置を指定します。デフォルトは 0 です。

[children](#) プロパティ、および [createChild](#)、[remove](#) メソッドも参照してください。

例：

```
// 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			
-----	-------------------------------------------------------------------------------------	--	--

パラメータ：なし
戻り値：なし

しおり（およびそのすべての子）を、しおりツリーから削除します。

[children](#) プロパティ、および [createChild](#)、[insertChild](#) メソッドも参照してください。



例：

```
// Remove all bookmarks from the document.  
bookmarkRoot.remove();
```

カラー配列

JavaScript では、色は 1、2、4、または 5 個のいずれかの要素を含む配列として表され、それぞれ透明、グレー、RGB、CMYK カラースペースに対応します。配列の最初の要素は、カラースペースのタイプを示す文字列です。それ以降の要素は、0 以上 1 以下の数値です。次の表に、これを示します。

カラースペース	文字列	追加要素の個数
透明	"T"	0
グレー	"G"	1
RGB	"RGB"	3
CMYK	"CMYK"	4

例えば、赤色は ["RGB", 1, 0, 0] と表すことができます。

カラー配列に無効な文字列または不十分な要素が存在する場合は、黒色として解釈されず。

透明カラースペースは、色がまったくなく、そのフィールドの下にあるドキュメントの部分が透けて見えます。

グレーカラースペースの色は、無色の明度が単一の値で表現されます。このカラースペースでは 0 が黒色、1 が白色で、その中間値（「.5」、「.7」など）はグレーの明暗を表します。

RGB カラースペースの色は 3 つの値で表され、値は赤 (Red)、緑 (Green)、青 (Blue) の各要素の強度を示します。RGB はディスプレイ装置で一般的に使用されています。ディスプレイ装置は通常、赤、緑、青を組み合わせせて色を表現するからです。

CMYK カラースペースの色は 4 つの値で表され、青 (Cyan)、赤 (Magenta)、黄 (Yellow)、黒 (black) の各要素の量を示します。これらの色は従来から 4 色カラー印刷で使用されてきたインクの色で、このカラースペースは一般的にカラープリンタで使用されています。必要な色は青、赤、黄のみですが、印刷には黒も使用するのが普通です。これは、黒を使用した方が青、赤、黄のインクを混ぜ合わせるよりも鮮やかな黒を表現でき、他のインクに比べ価格も低いからです。

Color オブジェクト

Color オブジェクトは、基本色を定義する便利な静的オブジェクトです。JavaScript では、Color オブジェクトを介して基本色にアクセスします。カラー配列を必要とするプロパティを設定したりメソッドを呼び出しする場合には、このオブジェクトを使用してください。Color オブジェクトは *AForm.js* で定義されています。

Color オブジェクトのプロパティ

Color オブジェクトは、次の色および関連キーワードを定義しています。

Color オブジェクト	キーワード	同等の JS	バージョン
透明	color.transparent	["T"]	
黒	color.black	["G" 0]	
白	color.white	["G" 1]	
赤	color.red	["RGB" 1 0 0]	
緑	color.green	["RGB" 0 1 0]	
青	color.blue	["RGB" 0 0 1]	
シアン	color.cyan	["CMYK" 1 0 0 0]	
マゼンタ	color.magenta	["CMYK" 0 1 0 0]	
イエロー	color.yellow	["CMYK" 0 0 1 0]	
ダークグレー	color.dkGray	["G" 0.25]	4.0
グレー	color.gray	["G" 0.5]	4.0
ライトグレー	color.ltGray	["G" 0.75]	4.0

例：

```
// This example sets the text color of the field to red
// if the value of the field is negative, else it sets it
// to black.
var f = event.target; /* field that the event occurs at */
f.textColor = event.value < 0 ? color.red : color.black;
```

Color オブジェクトのメソッド

convert

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

パラメータ：カラー配列、*cColorspace*

戻り値：カラー配列

このメソッドは、Color オブジェクトで定義されるカラースペースとカラー値を指定のカラースペースに変換します。グレーのカラースペースへの変換では、カラーテレビの信号を白黒テレビで表示するのと同様に、情報の損失があり得ます。印刷時の注意：RGB から CMYK への変換では、黒の生成やカラー削除パラメータは考慮されていません。

equal

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

パラメータ：カラー配列 1、カラー配列 2

戻り値：*bEqual*

このメソッドは、2 つのカラー配列が同じかどうかを比較します。これを判断するために、必要に応じて変換が行われます（例えば、["RGB" 1 1 0] と ["CMYK" 0 0 1 0] は同じです）。

```
var f = this.getField("foo");
if (color.equal(f.textColor, f.fillColor))
    app.alert("Foreground and background color are the same!");
```

Connection オブジェクト

5.0			⊗
-----	--	--	---

Connection オブジェクトは、データベースセッションをカプセル化するオブジェクトです。Connection オブジェクトは [ADBC オブジェクト](#) の [newConnection](#) メソッドから返されます。

Connection オブジェクトのメソッド

newStatement

5.0			⊗
-----	--	--	---

パラメータ：なし

戻り値：Statement オブジェクトまたは null

`newStatement` メソッドは、[Statement オブジェクト](#) を作成します。データベース操作は、このオブジェクトを介して行うことができます。このメソッドは成功すると Statement オブジェクトを返し、失敗すると `null` を返します。

例：

```
// 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			⊗
-----	--	--	---

パラメータ：なし

戻り値：オブジェクトの配列

`getTableList` メソッドは、データベースに含まれるさまざまなテーブルの情報を取得します。このメソッドは、[TableInfo オブジェクト](#) の配列を返します。このメソッドが失敗することはありませんが、長さがゼロの配列を返すことがあります。

次の表に、[getTableList](#) メソッドから返される [TableInfo オブジェクト](#)のプロパティを一覧にして示します。

TableInfo オブジェクト			
TableInfo オブジェクトには、テーブルの基本情報が含まれます。			
プロパティ	型	アクセス	説明
name	文字列	R	テーブルの識別名を表す文字列です。この文字列を SQL ステートメントで使用して、TableInfo オブジェクトに関連付けられているテーブルを識別することができます。
description	文字列	R	テーブルのデータベース固有の情報を含む文字列です。

例

```
/* 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			⊗
-----	--	--	---

パラメータ : *cName*
戻り値 : *ColumnInfo* オブジェクトの配列

getColumnList メソッドは、テーブルに含まれるさまざまな列の情報を取得します。

cName は、情報を取得する列が存在するテーブルの名前です。

このメソッドは、[ColumnInfo オブジェクト](#)の配列を返します。このメソッドが失敗することはありませんが、長さがゼロの配列を返す場合があります。

次の表に、[ColumnInfo オブジェクト](#)のプロパティを一覧にして示します。

ColumnInfo オブジェクト			
ColumnInfo オブジェクトには、データの列の基本情報が含まれます。			
プロパティ	型	アクセス	説明
name	文字列	R	列の識別名を表す文字列です。この文字列を getColumn メソッドで使用して、ColumnInfo オブジェクトに関連付けられている列を識別することができます。
description	文字列	R	列のデータベース固有の情報を含む文字列です。
type	数値	R	ColumnInfo オブジェクトに関連付けられている列のデータの SQL 型 を識別する数値です。
typeName	文字列	R	ColumnInfo オブジェクトに関連付けられている列のデータ型を示す文字列です。これは、データ型を表すデータベース固有の文字列であり、 <i>type</i> プロパティに含まれる情報とは異なります。このプロパティでは、ユーザ定義のデータ型に関して、有用な情報を得ることができます。

例：

```
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 オブジェクト



Console オブジェクトは JavaScript コンソールにアクセスするための静的なオブジェクトで、デバッグメッセージを表示したり、JavaScript を実行したりすることができます。Console オブジェクトは、Acrobat Reader では動作しません。

Console オブジェクトのメソッド

show

パラメータ : なし
戻り値 : なし

このメソッドは、コンソールウィンドウを表示します。

hide

パラメータ : なし
戻り値 : なし

このメソッドは、コンソールウィンドウを閉じます。

println

パラメータ : *cMessage*
戻り値 : なし

このメソッドは、*cMessage* 文字列の値に改行を付加してコンソールウィンドウに表示します。

```
// This example prints the value of a field to the console window
var f = event.target;
console.println("Field value = " + f.value);
```

clear

パラメータ : なし
戻り値 : なし

このメソッドは、コンソールウィンドウバッファからすべての出力をクリアします。

Data オブジェクト

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

Data オブジェクトは、ドキュメント内の埋め込みファイルまたはデータストリームを表すもので、ドキュメント内のネームツリーに保持されています。詳しくは、『*PDF Reference Manual*』の「*Names Tree*」および「*Embedded File Streams*」の節を参照してください。

Data オブジェクトは、外部ファイルシステムから挿入、照会、抽出することができます。関連するソースファイル、メタデータ、その他のデータを、ドキュメントに関連付けて埋め込むのは良い方法です。

[Docオブジェクト](#) [dataObjects](#) プロパティ、ドキュメントの [createDataObject](#)、[exportDataObject](#)、[getDataObject](#)、[importDataObject](#)、[removeDataObject](#) メソッド、および [Data オブジェクト](#) も参照してください。

Data オブジェクトのプロパティ

creationDate

型：日付

アクセス：R

このプロパティは、埋め込まれたファイルの作成日を示します。

modDate

型：日付

アクセス：R

このプロパティは、埋め込まれたファイルの変更日を示します。

MIMEType

型：文字列

アクセス：R

このプロパティは、データオブジェクトに関連する MIME タイプを示します。。

name

型：文字列

アクセス：R

このプロパティは、データオブジェクトに関連する名前を示します。。

例：

```
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

型：文字列

アクセス：R

このプロパティは、埋め込まれたファイルのデバイスに依存しないパスを示します。

size

型：数値

アクセス：R

このプロパティは、圧縮されていないデータオブジェクトのサイズをバイト単位で示します。

Doc オブジェクト

JavaScript の Doc オブジェクトには、ビューアで開いてる PDF ドキュメントと JavaScript インタプリタとの間のインタフェイスが備わっており、PDF ドキュメントに関するメソッドおよびプロパティが用意されています。

JavaScript による Doc オブジェクトへのアクセス

JavaScript から Doc オブジェクトにアクセスするには、さまざまな方法がありますが、[this オブジェクト](#)を使用するのが最も一般的な方法です。このオブジェクトは、通常、ドキュメントの基盤となる Doc オブジェクトを指し示しています。一部のプロパティおよびメソッドには Doc オブジェクトを返すものもあります。例えば、[activeDocs](#)、[openDoc](#)、[extractPages](#) はすべて、Doc オブジェクトを返します。

例：

```
// Access through "this"
var nPages = this.numPages;           // get number of pages in "this" document
var aCrop = this.getPageBox();       // get the crop box for "this" document

/* Access through return values: From one document, open, modify, save and
** close another. */
var myDoc = app.openDoc("myNovel.pdf", this); // path relative to "this" doc
myDoc.info.Title = "My Great Novel";
myDoc.saveAs(myDoc.path);
myDoc.closeDoc(true);
```

JavaScript はイベントによって実行することができますが、イベントが発生すると [Event オブジェクト](#)が作成されるので、この Event オブジェクトの [target](#) プロパティを介して Doc オブジェクトにアクセスすることもできます。target プロパティは、マウス、フォーカス取得、フォーカス消失、計算、検証、およびフォーマットイベントを開始した [Field オブジェクト](#)を返すので、[Field オブジェクト](#)の [doc](#) プロパティを介して Doc オブジェクトにアクセスすることができます。これ以外のイベントの場合、target プロパティは Doc オブジェクトを指し示します。

例：[Event オブジェクト](#)を介したアクセス

```
// In Mouse, calculate, validate, format, focus, blur events
var myDoc = event.target.doc;

// In all other events (e.g., batch or console events)
var myDoc = event.target;
```

Doc オブジェクトのプロパティ

author



型：文字列

アクセス：R/W

このプロパティは、ドキュメントの作成者を示します。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [info](#) プロパティも参照してください。

注意： このプロパティは、Acrobat Reader では読み取り専用です。

baseUrl

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

型：文字列

アクセス：R/W

ドキュメントのベース URL です。ベース URL は、ドキュメント内の相対 Web リンクの解決に使用されます。

```
console.println("Base URL was " + this.baseUrl);
this.baseUrl = "http://www.adobe.com/products/";
console.println("Base URL is " + this.baseUrl);
```

[URL](#) プロパティも参照してください。

bookmarkRoot

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

型：オブジェクト

アクセス：R

しおりツリーのルートしおりを返します。このしおりはユーザには表示されず、プログラムでツリーや子しおりにアクセスするために使用されます。

使用例については、[Bookmark オブジェクト](#)を参照してください。

calculate

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

型：ブーリアン

アクセス：R/W

このプロパティが *true* の場合はドキュメントで計算を実行できます。*false* の場合、ドキュメントで計算がまったく実行されなくなります。デフォルト値は *true* です。今後のバージョンではアプリケーションレベルの [calculate](#) プロパティに代わってこのプロパティが使用されます。

creator

☹			
---	--	--	--

型：文字列

アクセス：R

このプロパティは、ドキュメントを作成したソフトウェア（「Adobe FrameMaker」、「Adobe PageMaker」など）を示します。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [info](#) プロパティも参照してください。

creationDate

☹			
---	--	--	--

型：日付

アクセス：R

このプロパティは、ドキュメントの作成日を示します。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [info](#) プロパティも参照してください。

dataObjects

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

型：配列

アクセス：R

ドキュメント内のすべての名前付きデータオブジェクトを配列にして返します。

例：

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

Doc オブジェクトの [dataObjects](#) プロパティ、Doc オブジェクトの [createDataObject](#)、[exportDataObject](#)、[getDataObject](#)、[importDataObject](#)、[removeDataObject](#) メソッド、および [Data オブジェクト](#) も参照してください。

delay

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

型：ブーリアン

アクセス：R/W

このプロパティは、ドキュメント内のすべてのフィールドの再描画を一時的に停止させることができます。複数のフィールドに対する一連の変更が終了してから、フィールドの外観をまとめて再描画するために使用するのが一般的です。*delay* プロパティを *true* に設定すると、再度 *false* にするまですべての変更が強制的に待ち状態になります。*false* に設定すると、ページ上のすべてのフィールドが再描画されます。

フィールドレベルの [delay](#) プロパティも参照してください。

dirty

			
--	-----------------------------------------------------------------------------------	--	--

型：ブーリアン

アクセス：R/W

このプロパティは、ドキュメントに変更が加えられた（そのため保存する必要がある）かどうかを示します。例えば、ドキュメント内の状態フィールドの更新など、特に保存の必要がない変更を行った場合には、ドキュメントの *dirty* フラグをリセットすると便利です。

```
var f = this.getField("Status");  
var b = this.dirty;  
f.value = "Press the reset button to clear the form.";  
this.dirty = b;
```

disclosed

5.05			
------	-----------------------------------------------------------------------------------	--	--

型：ブーリアン

アクセス：R/W

このプロパティは、ドキュメントを他のドキュメントの JavaScript からアクセス可能にするかを示します。アプリケーションレベルの [activeDocs](#) 配列に格納されるのは、*disclosed* が true に設定されているドキュメントだけです。アプリケーションレベルの [openDoc](#) メソッドは、対象ドキュメントの *disclosed* が文書レベルの JavaScript で true に設定されている場合は [Doc オブジェクト](#) を返し、それ以外の場合は null を返します。

external

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

型：ブーリアン

アクセス：R

このプロパティは、現在のドキュメントが Acrobat アプリケーションあるいは外部ウィンドウ（Web ブラウザなど）のどちらで表示されているかを示します。

filesize

型：整数

アクセス：R

このプロパティは、ドキュメントのファイルサイズをバイト単位で示します。

icons

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

型 : 配列

アクセス : R

ドキュメントレベルの名前付きアイコンツリーに存在する名前付きの[アイコンオブジェクト](#)を配列にして返します。

例 :

```
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");
```

以下に、アイコンオブジェクトのプロパティをまとめます。

アイコンオブジェクト			
アイコンオブジェクトは、ドキュメントに保存されている Form XObject の外観の隠ぺい表現です。ほとんどの場合、アイコンはボタントイプの Field オブジェクト で使用されます。			
プロパティ	型	アクセス	説明
name	文字列	R	このプロパティは、アイコン名を返します。アイコンに名前が存在するかどうかは、アイコンがドキュメントレベルの名前付きアイコンツリーに存在するかどうかによって決まります。

例 :

```
// list all named icons
for (var i = 0; i < this.icons.length; i++) {
    console.println("icon[" + i + "]=" + this.icons[i].name);
}
```

[Doc オブジェクト](#) の [addIcon](#)、[getIcon](#)、[importIcon](#)、[removeIcon](#) メソッド、[Field オブジェクト](#) の [buttonGetIcon](#)、[buttonImportIcon](#)、[buttonSetIcon](#) メソッド、および [アイコンオブジェクト](#) も参照してください。

info

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

型 : オブジェクト

アクセス : R

Acrobat Reader の場合、PDF ファイルのドキュメント情報ディクショナリから、プロパティを含むオブジェクトが返されます。標準のエントリには、*Title*、*Author*、*Subject*、*Keywords*、*Creator*、*Producer*、*CreationDate*、*ModDate*、および *Trapped* があります。詳しくは、『[PDF](#)

[Reference](#)』の 475 ページの表 8.2 「*Entries in a document information dictionary*」を参照してください。

例：

```
// get title of document
var docTitle = this.info.Title;
```

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

型：オブジェクト

アクセス：R/W

Acrobat では、*info* オブジェクトプロパティは取得および設定が可能で、このオブジェクトのプロパティを設定した場合、ドキュメントが変更されたとみなされます。非標準のプロパティを設定し、他のドキュメント情報フィールドを追加することもできます。このオブジェクトのプロパティに Acrobat Reader で書き込みを行うと、例外が発生します。

例：次のスクリプトを実行します。



```
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]);
```

この場合、出力は次のようになります。

```
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
```

注意： 標準のエントリでは大文字と小文字は区別されません。つまり、*doc.info.Keywords* と *doc.info.keywords* は同じになります。

keywords

		
-------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------

型：文字列

アクセス：R/W

このプロパティは、ドキュメントを説明するキーワード（「forms」、「taxes」、「government」など）を示します。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [info](#) プロパティも参照してください。

注意： このプロパティは、Acrobat Reader では読み取り専用です。

layout

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

型：文字列

アクセス：R/W

現在のドキュメントのページレイアウトを変更します。このプロパティの有効な値は、「SinglePage」、「OneColumn」、「TwoColumnLeft」、および「TwoColumnRight」です。

modDate

☹			
---	--	--	--

型：日付

アクセス：R

このプロパティは、ドキュメントの最終変更日を示します。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [info](#) プロパティも参照してください。

numFields

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

型：整数

アクセス：R

このプロパティは、ドキュメントに含まれるフィールドの合計数を返します。[getNthFieldName](#) メソッドも参照してください。

numPages

型：整数

アクセス：R

このプロパティは、ドキュメントのページ数を示します。

numTemplates

☹			
---	--	--	--

型：整数

アクセス：R

このプロパティは、ドキュメントに含まれるテンプレートの数を返します（[getNthTemplate](#) および [spawnPageFromTemplate](#) メソッドも参照）。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [templates](#) プロパティも参照してください。

path

型：文字列

アクセス：R

このプロパティは、デバイスに依存しないドキュメントのパスを示します（例えば、`/c:/Program Files/Adobe/Acrobat 5.0/Help/AcroHelp.pdf` など）。パスの正確な構文について詳しくは、『[PDF Reference](#)』の 108 ページにある第 3.10.1 節「File Specification Strings」を参照してください。

pageNum

型：整数

アクセス：R/W

このプロパティは、ドキュメントのページを取得、設定します。`pageNum` を特定のページに設定する場合、「0」ベースであることを念頭に置いてください。

```
// This example will go to the first page of the document.  
this.pageNum = 0 ;
```

```
// This example will advance the document to the next page  
this.pageNum++;
```

producer



型：文字列

アクセス：R

このプロパティは、ドキュメントを PDF 変換したソフトウェア（「Acrobat Distiller」、`PDFWriter` など）を示します。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [info](#) プロパティも参照してください。

securityHandler



型：文字列または `null`

アクセス：R

このプロパティは、ドキュメントの暗号化に使用されたセキュリティハンドラ名を返します。セキュリティハンドラが存在しない場合（ドキュメントが暗号化されていないなど）は `null` を返します。次に例を示します。

```
console.println(this.securityHandler != null ?
    "This document is encrypted with " + this.securityHandler + " security." :
    "This document is unencrypted.");
```

ドキュメントが標準のセキュリティハンドラで暗号化されている場合、出力は次のようになります。

```
Encrypted with Standard security.
```

selectedAnnots

5.0			<input checked="" type="checkbox"/>
-----	--	--	-------------------------------------

型 : 配列

アクセス : R

このプロパティは、現在選択されているすべてのマークアップ注釈に対応する [Annot オブジェクト](#) を、配列にして返します。

例 :

```
// 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);
```

[getAnnot](#) および [getAnnots](#) も参照してください。

sounds

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

型 : 配列

アクセス : R

ドキュメント内のすべての名前付き [Sound オブジェクト](#) を配列にして返します。

例 :

```
var s = this.sounds;
for (i = 0; i < s.length; i++)
    console.println("Sound[" + i + "]=" + s[i].name);
```

[getSound](#)、[importSound](#)、[deleteSound](#) メソッド、および [Sound オブジェクト](#) も参照してください。

spellDictionaryOrder

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

型 : 配列

アクセス : R/W

このプロパティでは、ドキュメントの辞書配列の検索順序にアクセスしたり、検索順序を指定したりすることができます。Spelling プラグインは最初にこの配列から単語を検索し、次に「スペルチェック」パネルでユーザが選択した辞書を検索します。ユーザが設定した辞書の検索順序は、[spell.dictionaryOrder](#) プロパティで取得できます。また、現在インストールされている辞書の配列は、[spell.dictionaryNames](#) プロパティで取得することができます。

例えば、医学に関連するフォームを作成してユーザ入力のスペルチェックを行う場合、ユーザ設定されている辞書の順番ではなく、最初に医学辞書でチェックすることができます。

subject

☹	🔒		⊗
---	---	--	---

型：文字列

アクセス：R/W

このプロパティは、ドキュメントのサブタイトルを示します。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [info](#) プロパティも参照してください。

注意： このプロパティは、Acrobat Reader では読み取り専用です。

templates

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

型：配列

アクセス：R

このプロパティは、ドキュメントに含まれるすべての Template オブジェクトの配列を返します。

[Doc オブジェクト](#)の [createTemplate](#)、[getTemplate](#)、[removeTemplate](#) メソッド、および [Template オブジェクト](#)も参照してください。

title

☹	🔒		⊗
---	---	--	---

型：文字列

アクセス：R/W

このプロパティは、ドキュメントのタイトルを示します。今後のバージョンでこのプロパティに代わって使用される Doc オブジェクトの [info](#) プロパティも参照してください。

注意： このプロパティは、Acrobat Reader では読み取り専用です。

URL

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

型：文字列

アクセス：R

このプロパティは、ドキュメントの URL を示します。ドキュメントがローカルにある場合は、「file:///」形式で URL を返します。これは [baseURL](#) とは異なることがあります。

zoom

型：浮動小数点

アクセス：R/W

このプロパティは、現在のページのズームレベルを取得、設定します。有効な値は、8.33% や 1600%などの浮動小数です。

```
// This example will zoom in to twice the current zoom level.  
this.zoom *= 2;
```

```
// This now sets the zoom to 200%  
this.zoom = 200;
```

zoomType

型：文字列

アクセス：R/W

このプロパティは、ドキュメントの現在のズームタイプを示します。有効なズームタイプは、NoVary、FitPage、FitWidth、FitHeight、FitVisibleWidth、および Preferred です。JavaScript では、有効なズームタイプをすべて定義する便利な zoomType オブジェクトが用意されています。以下のようなズームタイプがあります。

ズームタイプ	キーワード
NoVary	zoomtype.none
FitPage	zoomtype.fitP
FitWidth	zoomtype.fitW
FitHeight	zoomtype.fitH
FitVisibleWidth	zoomtype.fitV
Preferred	zoomtype.pref

例：

```
// This example sets the zoom type of the document to fit the width.  
this.zoomType = zoomtype.fitW;
```

Doc オブジェクトのメソッド

addAnnot

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

パラメータ：オブジェクトリテラル
戻り値：Annot オブジェクト

このメソッドは、指定されたオブジェクトリテラルに従って [Annot オブジェクト](#) を作成します。オブジェクトリテラルとは、注釈のプロパティを示す汎用オブジェクト（「[メソッドのパラメータ指定](#)」を参照）のことで、作成する Annot オブジェクトの [type](#)、[rect](#)、[page](#) などを指定します。

例

```
// This example creates a "Square" annotation.  
var sqannot = this.addAnnot({type: "Square", page: 0});
```

これは最も簡単な例で、「Square」タイプの注釈 *sqannot* がページ 0（0 ベースのページ番号システムなので）に作成されます。

注意： オブジェクトリテラルにプロパティを指定しない場合は、指定された注釈 [type](#) のデフォルト値が使用されます。

例：

```
var annot = this.addAnnot({  
    page: 0,  
    type: "Square",  
    rect: [0, 0, 100, 100],  
    name: "OnMarketShare",  
    author: "A. C. Robat",  
    contents: "This section needs revision."  
});
```

addField

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

パラメータ：cName、cFieldType、nPageNum、oCoords
戻り値：オブジェクト

新規のフォームフィールドを作成し、[Field オブジェクト](#)を返します。

cName は、作成する新しいフィールドの名前です。この名前をドットで区切ると階層を示すことができます（例えば、name.last は親ノードの name と子ノードの last を示します）。

cFieldType は、作成するフォームフィールドのタイプです。有効なタイプは、「text」、「button」、「combobox」、「listbox」、「checkbox」、「radiobutton」、「signature」です。

nPageNum は、フィールドを追加するページのインデックスです（0 ベース）。

oCoords は、[回転ユーザスペース](#)座標の 4 つの数値を含む配列で、フォームフィールドのサイズと位置を指定します。座標は、左上の *x*、左上の *y*、右下の *x*、右下の *y* の順で指定します。Field.[rect](#) も参照してください。

例：次のコードをバッチシーケンスなどで使用して、関連する複数ドキュメントの全ページにナビゲーションアイコンを作成することができます。

```
var inch = 72;
for (var p = 0; p < this.numPages; p++) {
    var aRect = this.getPageBox( {nPage: p} );
    aRect[0] += .5*inch;           // position rectangle (.5 inch, .5 inch)
    aRect[2] = aRect[0]+.5*inch;  // from upper left hand corner of page.
    aRect[1] -= .5*inch;          // Make it .5 inch wide
    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("➡")      // a right arrow
    f.delay = false;
}
```

別の例については、[setAction](#) を参照してください。

注意： 情報パネルを使用してバウンディングボックスの座標を取得する場合、情報パネルでは座標系の原点に左上隅が使用されているので注意してください。情報パネルの座標を回転ユーザスペースの座標に変換するには、画面のページの高さから情報パネルの *y* 座標をマイナスするだけです。

addIcon

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

パラメータ : *cName*、アイコンオブジェクト
戻り値 : なし

このメソッドは、*cName* に指定した名前で、新しい名前付き [アイコンオブジェクト](#) をドキュメントレベルのアイコンツリーに追加します。

例 : この例では、ドキュメントのフォームボタンフィールドにアイコンが添付されていることを前提として、そのアイコンに名前を割り当てます。例えば、[getIcon](#) にこの名前を指定してアイコンオブジェクトを取得し、別のボタンで 사용할 ことができます。

```
var f = this.getField("myButton");  
this.addIcon("myButtonIcon", f.buttonGetIcon());
```

[icons](#) プロパティ、[Doc オブジェクト](#) の [getIcon](#)、[importIcon](#)、[removeIcon](#) メソッド、[Field オブジェクト](#) の [buttonGetIcon](#)、[buttonImportIcon](#)、[buttonSetIcon](#) メソッド、および [アイコンオブジェクト](#) も参照してください。

addThumbnails

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

パラメータ : [*nStart*]、[*nEnd*]
戻り値 : なし

指定したページのサムネールを作成します。

nStart と *nEnd* には、ページ範囲を指定します (0 ベース)。 *nStart* および *nEnd* を指定しない場合、ドキュメントの全ページが対象になります。 *nStart* のみを指定した場合は、*nStart* のページのみが対象になり、*nEnd* のみを指定した場合は、0 から *nEnd* までが対象になります。

[removeThumbnails](#) メソッドも参照してください。

addWeblinks

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

パラメータ : [*nStart*]、[*nEnd*]
戻り値 : 整数

指定のページをスキャンして http: 形式のテキストを検索し、それを URL アクションを伴うリンクに変換します。

nStart と *nEnd* には、ページ範囲を指定します (0 ベース)。 *nStart* および *nEnd* を指定しない場合、ドキュメントの全ページが対象になります。 *nStart* のみを指定した場合は、 *nStart* のページのみが対象になり、 *nEnd* のみを指定した場合は、0 から *nEnd* までが対象になります。

このメソッドは、ドキュメントに追加された Web リンクの数を返します。

[removeWeblinks](#) メソッドも参照してください。

bringToFront

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

パラメータ : なし
戻り値 : なし

ビューアでドキュメントが手前に表示されていない場合、手前に表示します。

例 :

```
/* 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;
for (var i = 0; i < d.length; i++)
{
    if (d[i].info.Title == "Annual Report")
        d[i].bringToFront();
}
```

calculateNow

パラメータ : なし
戻り値 : なし

現在のドキュメントにあるすべての計算フィールドで強制的に計算を行うには、このメソッドを使用します。

closeDoc

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

パラメータ : [*bNoSave*]
戻り値 : *false*

このメソッドは、*Doc* オブジェクトに対応するドキュメントを閉じます。 *bNoSave* が *false* (デフォルト) の場合、ドキュメントが変更されていればドキュメントを保存するようプロンプトが表示されます。 *bNoSave* が *true* の場合は、ドキュメントが変更されていてもプロ

ンプトは表示されず、ドキュメントは保存されないまま閉じられます。ユーザの確認がないままデータが失われる可能性もあるので、慎重に使用してください。

注意： ドキュメントが閉じられるとドキュメント状態が急に変化するので、実行中の JS に影響を与える可能性があります。このメソッドは慎重に使用するようにしてください。また、Page イベントあるいは Document イベントでこのメソッドを使用すると、アプリケーションの動作が不安定になることがあります。

createDataObject

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

パラメータ : *cName*、*cValue*、[*cMIMEType*]

戻り値 : なし

データオブジェクトをその場で構築することができます。外部ファイル（ADBC データベース呼び出しなど）は使用せず、他のソースからデータを生成する場合に便利です。

cName は、データオブジェクトに関連付ける名前です。

cValue は、埋め込むデータを含む文字列です。

cMIMEType は、データの MIME タイプです。デフォルトは「text/plain」です。

例：

```
this.createDataObject("MyData", "This is some data.");
```

Doc オブジェクトの [dataObjects](#) プロパティ、Doc オブジェクトの [createDataObject](#)、[exportDataObject](#)、[getDataObject](#)、[importDataObject](#)、[removeDataObject](#) メソッド、および [Data オブジェクト](#) も参照してください。

createTemplate

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

パラメータ : *cName*、[*nPage*]

戻り値 : *Template* オブジェクト


このメソッドは、指定のページから表示可能なテンプレートを作成します。

cName は、ページに関連付ける名前です。

nPage は、対象とするページ（0 ベース）です。*nPage* を指定しない場合、ドキュメントの最初のページを指定したことになります。

このメソッドは、新しく作成された [Template オブジェクト](#) を返します。

[templates](#) プロパティ、[Doc オブジェクト](#)の [getTemplate](#)、[removeTemplate](#) メソッド、および [Template オブジェクト](#)も参照してください。

セキュリティ  : このメソッドは、バッチ、コンソール、またはメニューイベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

deletePages

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

パラメータ : *[nStart]*、*[nEnd]*

戻り値 : なし

ドキュメントからページを削除します。

nStart は、削除するページ範囲の最初のページ (0 ベース) です。*nEnd* パラメータはオプションで、削除するページ範囲の最後のページを示します。*nEnd* を指定しない場合、*nStart* に指定したページのみが削除されます。

nStart と *nEnd* のデフォルト値はどちらも 0 ですから、*this.deletePages()* を実行すると、最初のページ (ページ 0) が削除されます。

[insertPages](#)、[extractPages](#)、および [replacePages](#) メソッドも参照してください。

注意 : ドキュメントの全ページを削除することはできません。少なくとも 1 ページは残しておく必要があります。

deleteSound

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

パラメータ : *cName*

戻り値 : なし

このメソッドは、指定した名前の Sound オブジェクトをドキュメントから削除します。

例 :

```
this.deleteSound("Moo");
```

[sounds](#) プロパティ、[getSound](#)、[importSound](#) メソッド、および [Sound オブジェクト](#)も参照してください。

exportAsFDF

4.0			⊗
-----	--	--	---

パラメータ : *[bAllFields]*、*[bNoPassword]*、*[aFields]*、*[bFlags]*、*[cPath]*
戻り値 : なし

このメソッドは、FDF ファイルをローカルハードドライブにエクスポートします。

オプションの *bAllFields* パラメータが *true* の場合は、値のないフィールドを含むすべてのフィールドをエクスポートし、*false* (デフォルト) の場合は、値のないフィールドをエクスポートしません。

オプションの *bNoPassword* パラメータが *true* (デフォルト) の場合は、「password」フラグが設定された FDF テキストフィールドはエクスポートされません。

オプションの *aFields* パラメータは、フィールド名の配列、または 1 つのフィールド名を含む文字列です。このパラメータを指定した場合は、*bAllFields* または *bNoPassword* で除外されるフィールドを除いて、指定のフィールドのみがエクスポートされます。このパラメータを省略した場合または値が *null* の場合は、フォームに含まれるすべてのフィールド (*bAllFields* と *bNoPassword* で除外されるフィールドを除く) がエクスポートされます。フィールドのサブツリー全体をエクスポートするには、親のフィールド名をこのパラメータに指定します。以下の例を参照してください。

オプションの *bFlags* パラメータが *true* の場合は、エクスポートする FDF にフィールドのフラグが含まれます。デフォルトは *false* です。


5.0 追記事項

オプションの *cPath* は、デバイスに依存しない FDF のパス名です (デバイスに依存しないパス名の形式について詳しくは、『[PDF Reference](#)』の第 3.10.1 節を参照)。パス名には、現在のドキュメントの位置に対する相対パスも指定できます。パラメータを省略すると、エクスポート先のファイルを選択するダイアログが表示されます。

例 :

```
/* Export the entire form (including empty fields) with flags. */
this.exportAsFDF(true, true, null, true);
/* Export the name subtree with no flags. */
this.exportAsFDF(false, true, "name");
```

上の例は、サブツリー全体を簡単にエクスポートする方法を示しています。*aFields* パラメータに「name」を指定すると、「name.title」、「name.first」、「name.middle」、「name.last」などがエクスポートされます。

セキュリティ  : 「cPath」パラメータを指定した場合、バッチ、コンソール、またはメニューイベントでしかこのメソッドを実行できなくなります。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

exportAsXFDF

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

パラメータ : [bAllFields]、[bNoPassword]、[aFields]、[cPath]
戻り値 : なし


このメソッドは、XFDF ファイルをローカルハードドライブにエクスポートします。XFDF は、Acrobat フォームデータを XML で表したものです。詳しくは、[弊社の CD で提供されるマニュアル](#)の『Forms System Implementation Notes』を参照してください。

オプションの *bAllFields* パラメータが *true* の場合は、値のないフィールドを含むすべてのフィールドをエクスポートし、*false* (デフォルト) の場合は、値のないフィールドはエクスポートされません。

オプションの *bNoPassword* パラメータが *true* (デフォルト) の場合は、「password」フラグが設定された XFDF テキストフィールドはエクスポートされません。

オプションの *aFields* パラメータは、複数のフィールド名の配列、または 1 つのフィールド名を含む文字列です。このパラメータを指定した場合は、*bAllFields* または *bNoPassword* で除外されるフィールドを除いて、指定のフィールドのみがエクスポートされます。このパラメータを省略した場合または値が *null* の場合は、フォームに含まれるすべてのフィールド (*bAllFields* と *bNoPassword* で除外されるフィールドを除く) がエクスポートされます。

オプションの *cPath* は、デバイスに依存しない XFDF のパス名です (デバイスに依存しないパス名の形式について詳しくは、『[PDF Reference](#)』の第 3.10.1 節を参照)。パス名には、現在のドキュメントの位置に対する相対パスも指定できます。パラメータを省略すると、エクスポート先のファイルを選択するダイアログが表示されます。

セキュリティ  : cPath パラメータを指定した場合、バッチ、コンソール、またはメニューイベントでしかこのメソッドを実行できなくなります。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

exportDataObject

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

パラメータ : *cName*、*[cDIPath]*

戻り値 : なし

このメソッドは、指定のデータオブジェクトを外部ファイルにエクスポートします。


cName は、抽出するデータオブジェクトの名前です。

cDIPath はオプションで、データオブジェクトのエクスポート先となるデバイスに依存しないパスを指定します。このパスには、絶対パスあるいは現在のドキュメントに対する相対パスのどちらでも指定できます。*cDIPath* を指定しない場合、保存場所を指定するプロンプトが表示されます。パスの正確な構文について詳しくは、『*PDF Reference Manual*』の「*File Specification Strings*」を参照してください。

例 :

```
// Prompt the user for a file and location to extract to.
this.exportDataObject("MyData");
// Extract to Foo.xml.
this.exportDataObject("MyData2", "../Foo.xml");
```

Doc オブジェクトの [dataObjects](#) プロパティ、Doc オブジェクトの [createDataObject](#)、[exportDataObject](#)、[getDataObject](#)、[importDataObject](#)、[removeDataObject](#) メソッド、および [Data オブジェクト](#) も参照してください。

セキュリティ:  *cDIPath* パラメータを指定した場合、バッチ、コンソール、メニューイベント時、あるいは外部呼び出し (OLE など) を介してしかこのメソッドを実行できなくなります。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#) を参照してください。

extractPages

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

パラメータ : *[nStart]*、*[cEnd]*、*[cPath]*

戻り値 : Doc オブジェクトまたは *null*

現在のドキュメントからページを抽出し、抽出したページで新しいドキュメントを作成します。

nStart と *nEnd* には、ソースドキュメントから抽出するページの範囲を指定します (0 ベース)。 *nStart* および *nEnd* を指定しない場合、ドキュメントの全ページが対象になります。 *nStart* のみを指定した場合は、*nStart* のページのみが対象になり、 *nEnd* のみを指定した場合は、0 から *nEnd* までが対象になります。

`cPath` には、新しいドキュメントの保存先となるパスをデバイスに依存しない形式で指定します。デバイスに依存しないパス名の形式について詳しくは、『[PDF Reference](#)』の第 3.10.1 節を参照してください。パス名には、現在のドキュメントの位置に対する相対パスも指定できます。この場合、新しいドキュメントはこのパスに保存されてから閉じられるので、戻り値は `null` オブジェクトになります。

`cPath` を指定しない場合、新しいドキュメントがビューアで開かれ、その [Doc オブジェクト](#) が返されます。


[deletePages](#)、[insertPages](#)、および [replacePages](#) メソッドも参照してください。

例：次のバッチシーケンスでは、ファイルごとに各ページを抽出し、それを一意の名前でフォルダに保存することができます。例えば、1 つの PDF 文書の中に、複数のクライアントの請求書がページ別に作成されているとします。このような場合、請求書の送付や印刷をするためには、PDF 文書をページごとに切り分けられると便利です。

```
/* Extract Pages to Folder */
// regular expression acquire the base name of file
var re = /\.*\$/|¥.pdf$/ig;

// filename is the base name of the file Acrobat is working on
var filename = this.path.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)
}
```

セキュリティ  : `cPath` パラメータを指定した場合、バッチ、コンソール、メニューイベント時、あるいは外部呼び出し (OLE など) を介してしかこのメソッドを実行できなくなります。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#) を参照してください。

flattenPages

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

パラメータ : `[nStart]`、`[nEnd]`
戻り値 : なし

指定のページ範囲にあるすべての注釈をページコンテンツに変換します。

nStart と *nEnd* には、現在のドキュメントのページ範囲を指定します (0 ベース)。 *nStart* のみを指定した場合は、*nStart* のページのみが対象になり、どちらのパラメータも指定しない場合は、全ページが対象になります。

注意： このメソッドの使用には十分注意してください。指定のページ範囲に存在するフォームフィールド、コメント、リンクを含むすべての注釈がページコンテンツに同化し、外観は残されますが、もはや注釈ではなくなります。

getAnnot

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

パラメータ : *nPage*、*cName*

戻り値 : *Annot* オブジェクトまたは *null*

このメソッドは、指定のページ *nPage* に存在する指定の名前 *cName* の [Annot オブジェクト](#) を返します。指定の注釈が存在しない場合は、*null* が返されます。

例：

```
var ann = this.getAnnot(0, "OnMarketShare");
if (ann == null)
    console.println("Not Found!")
else
    console.println("Found it! type: " + ann.type);
```

getAnnots

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

パラメータ : [*nPage*]、[*nSortBy*]、[*bReverse*]、[*nFilterBy*]

戻り値 : *Annot* オブジェクトの配列

このメソッドは、オプションパラメータで設定した条件を満たす [Annot オブジェクト](#) の配列を返します。

nPage はページ番号 (0 ベース) で、これを指定した場合、そのページに存在する注釈のみが返されます。*nPage* を指定しない場合、検索条件を満たす全ページの注釈が取得されます。

nSortBy はオプションで、配列に適用するソート方法を指定します。次の表に、*nSortBy* の有効な値を示します。

<i>nSortBy</i> の値	
名前	説明
ANSB_None	デフォルト、ソートなし、 <i>nSortBy</i> を省略した場合と同じ
ANSB_Page	ソートのプライマリキーとしてページ番号を使用
ANSB_Author	ソートのプライマリキーとして作成者を使用
ANSB_ModDate	ソートのプライマリキーとして変更日を使用
ANSB_Type	ソートのプライマリキーとして注釈タイプを使用

bReverse が *true* の場合、*nSortBy* で指定した配列のソート順が降順になります。

nFilterBy を指定した場合、特定の条件を満たす注釈のみが取得されます。次の表に、*nFilterBy* の有効な値を示します。

<i>nFilterBy</i> の値	
名前	説明
ANFB_ShouldNone	デフォルト、 <i>nFilterBy</i> を省略した場合と同じ
ANFB_ShouldPrint	印刷される注釈のみが対象
ANFB_ShouldView	表示される注釈のみが対象
ANFB_ShouldEdit	編集可能な注釈のみが対象
ANFB_ShouldAppearInPanel	「注釈」タブに表示される注釈のみが対象
ANFB_ShouldSummarize	注釈の一覧に表示される注釈のみが対象
ANFB_ShouldExport	書き出し可能な注釈のみが対象

例：

```
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));
```

[getAnnot](#) および [syncAnnotScan](#) も参照してください。特に、メソッドの後に記載された注意事項もご覧ください。

getDataObject

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

パラメータ : *cName*
 戻り値 : データオブジェクト

このメソッドは、指定の名前に対応するデータオブジェクトを返します。

cName は、取得するデータオブジェクトの名前です。

例 :

```
var d = this.getDataObject("MyData");
console.show();
console.clear();
for (var i in d)
    console.println("MyData." + i + "=" + d[i]);
```

Doc オブジェクトの [dataObjects](#) プロパティ、Doc オブジェクトの [createDataObject](#)、[exportDataObject](#)、[getDataObject](#)、[importDataObject](#)、[removeDataObject](#) メソッド、および [Data オブジェクト](#) も参照してください。

getField

パラメータ : *cName*
 戻り値 : *Field* オブジェクト

このメソッドは、PDF ドキュメントにある [Field オブジェクト](#) を JavaScript 変数にマッピングします。*cName* パラメータは、対象となるフィールドの名前です。このメソッドは、PDF ドキュメントのフォームフィールドを表す *Field* オブジェクトを返します。

例 :

```
// 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
```

getIcon

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

パラメータ : *cName*

戻り値 : アイコンオブジェクト

この関数は、ドキュメントに含まれる指定の名前に対応する [アイコンオブジェクト](#) を返します。指定の名前のアイコンが存在しない場合、*null* が返されます。

例 : 以下の例は、コンボボックスのカスタムキーストロークのスクリプトです。コンボボックスの表示項目名は、ドキュメントに埋め込まれているアイコンの名前であるとしてます。コンボボックスで異なる項目を選択すると、その項目名に対応するアイコンが「myPictures」ボタンフィールドの表面に表示されます。

```
if (!event.willCommit) {  
    var b = this.getField("myPictures");  
    var i = this.getIcon(event.change);  
    b.buttonSetIcon(i);  
}
```

この詳しい例については、[buttonSetIcon](#) を参照してください。

[icons](#) プロパティ、[Doc オブジェクト](#) の [addIcon](#)、[importIcon](#)、[removeIcon](#) メソッド、[Field オブジェクト](#) の [buttonGetIcon](#)、[buttonImportIcon](#)、[buttonSetIcon](#) メソッド、および [アイコンオブジェクト](#) も参照してください。

getNthFieldName

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

パラメータ : *nIndex*

戻り値 : 文字列

このメソッドは、ドキュメントに含まれる *n* 番目のフィールド名を取得します ([numFields](#) プロパティを参照)。

例 :

```
// 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

☹			⊗
---	--	--	---

パラメータ : *nIndex*

戻り値 : 文字列

このメソッドは、ドキュメントに含まれる n 番目のテンプレート名を取得します。

今後のバージョンではこのメソッドに代わって [Doc オブジェクト](#) の [templates](#) プロパティと [getTemplate](#) メソッド、そして [Template オブジェクト](#) が使用されます。

getPageBox

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

パラメータ : $[cBox]$ 、 $[nPage]$

戻り値 : 4 つの数値の配列

$cBox$ には、「Art」、「Bleed」、「BBox」、「Crop」（デフォルト）、「Trim」のいずれかを指定できます。これらのボックスの定義について詳しくは、『[PDF Reference](#)』の 524 ページにある第 8.6.1 節「Page Boundaries」を参照してください。デフォルトは「Crop」です。

$nPage$ は、対象となるページの番号（0 ベース）で、 $nPage$ を指定しない場合、ドキュメントの最初のページが対象となります。

このメソッドは、ページの名前付きボックスを囲む [回転ユーザスペース](#) の矩形を返します。

[setPageBoxes](#) メソッドも参照してください。

例：メディアボックスの寸法を取得します。

```
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			
-----	--	--	--

パラメータ : $[nPage]$

戻り値 : 文字列

指定ページのページラベル情報を返します。

$nPage$ は、対象となるページの番号（0 ベース）で、 $nPage$ を指定しない場合、ドキュメントの最初のページが対象となります。

適切な例については、[setPageLabels](#) メソッドも参照してください。

getPageNthWord

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

パラメータ : [nPage]、[nWord]、[bStrip]
戻り値 : 文字列


ページ上の n 番目の単語を返します。

$nPage$ は、対象となるページの番号 (0 ベース) で、 $nPage$ を指定しない場合、ドキュメントの最初のページが対象となります。

$nWord$ は、取得する単語のインデックス (0 ベース) です。 $nWord$ を指定しない場合、ページの最初の単語が対象となります。

$bStrip$ は、区切り文字と空白文字を取り除いてから単語を返すことを示すブーリアンです。デフォルトは *true* です。

[getNthTemplate](#)、[getPageNumWords](#)、および [selectPageNthWord](#) メソッドも参照してください。

セキュリティ  : ドキュメントのセキュリティでコンテンツの抽出ができないように設定されていると、このメソッドで例外が発生します。

getPageNthWordQuads

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

パラメータ : [nPage]、[nWord]
戻り値 : 四辺形の配列

ページ上の n 番目の単語を囲む [quads](#) の配列を返します。[quads](#) は、*Underline*、*StrikeOut*、*Highlight*、*Squiggly* などのテキストマークアップ注釈を作成するために使用できます。

$nPage$ は、対象となるページの番号 (0 ベース) で、 $nPage$ を指定しない場合、ドキュメントの最初のページが対象となります。

$nWord$ は、取得する単語のインデックス (0 ベース) です。 $nWord$ を指定しない場合、ページの最初の単語が対象となります。


[getPageNthWord](#) メソッドも参照してください。

例 : 次の例では、ドキュメントの 2 ページ目にある 5 番目の単語に下線を付けます。

```
var annot = this.addAnnot({
    page: 1,
    type: "Underline",
    quads: this.getPageNthWordQuads(1, 4),
```

```
author: "A. C. Acrobat",
contents: "Fifth word on second page"
});
```

他の例については、[checkWord](#) および [Highlight](#)、[Strikeout](#)、[Underline](#)、[Squiggle](#) を参照してください。

セキュリティ  : ドキュメントのセキュリティでコンテンツの抽出ができないように設定されていると、このメソッドで例外が発生します。

getPageNumWords

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

パラメータ : [nPage]
戻り値 : 数値

ページに含まれる単語の数を返します。

nPage は、対象となるページの番号 (0 ベース) で、nPage を指定しない場合、ドキュメントの最初のページが対象となります。

例 :

```
// 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 on this page.");
```

[getNthTemplate](#)、[getPageNthWord](#)、および [selectPageNthWord](#) メソッドも参照してください。

getPageRotation

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

パラメータ : [nPage]
戻り値 : 整数

指定ページの回転角度を取得します。

nPage は、対象となるページの番号 (0 ベース) で、nPage を指定しない場合、ドキュメントの最初のページが対象となります。

0、90、180、または 270 を返します。

[setPageRotations](#) メソッドも参照してください。

getPageTransition

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

パラメータ : *nPage*

戻り値 : 配列

指定ページの画面表示の切り替え方法を取得します。

nPage は、対象となるページの番号（0 ベース）で、*nPage* を指定しない場合、ドキュメントの最初のページが対象となります。

このメソッドは、[*nDuration*, *cTransition*, *nTransDuration*] という 3 つの値の配列を返します。

nDuration は、ビューアが自動的にページを切り替えるまでの最大表示時間です。-1 という値は、自動的にページが切り替わらないことを示します。

cTransition は、画面が切り替わる際にページに適用される効果の名前です。効果の有効な値については、アプリケーションプロパティの [transitions](#) を参照してください。

nTransDuration は、効果の継続時間（秒単位）です。

[setPageTransitions](#) メソッドも参照してください。

getSound

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

パラメータ : *cName*

戻り値 : *Sound* オブジェクト

このメソッドは、指定の名前に対応する *Sound* オブジェクトを返します。

例 :

```
var s = this.getSound("Moo");
console.println("Playing the " + s.name + " sound.");
s.play();
```

[sounds](#) プロパティ、[importSound](#)、[deleteSound](#) メソッド、および [Sound オブジェクト](#) も参照してください。

getTemplate

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

パラメータ : *cName*

戻り値 : *Template* オブジェクトまたは *null*

このメソッドは、名前付きテンプレートをドキュメントから取得します。名前付きテンプレートがドキュメントに存在しない場合、*null* を返します。

cName は、取得するテンプレートの名前です。

[templates](#) プロパティ、[Doc オブジェクト](#) の [createTemplate](#)、[removeTemplate](#) メソッド、および [Template オブジェクト](#) も参照してください。

getURL

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

パラメータ : *cURL*、*[bAppend]*

戻り値 : なし

このメソッドは、GET を使用して指定の URL をインターネットから取りだします。

cURL には、完全修飾の URL または相対 URL のどちらでも指定できます。URL の最後にパラメータを指定することも可能です。

現在のドキュメントをブラウザ内で表示しているか、Acrobat Web Capture が使用できない場合は、Weblink プラグインを使用して目的の URL を取り出します。

Acrobat で動作している場合、現在のドキュメントの URL は、ドキュメントのベース URL、先頭ページの URL（ドキュメントが Web Capture された場合）、ファイルシステムのいずれかから取得されます。

オプションの *bAppend* パラメータが *true*（デフォルト）に設定されている場合、取得したページを現在のドキュメントに追加します。ドキュメントを Web ブラウザで表示しているか Acrobat Web Capture プラグインが使用できない場合、あるいは URL が「file:///」形式の場合は、このフラグは *false* であるとみなされます。

gotoNamedDest

パラメータ : *cName*

戻り値 : なし

このメソッドは、PDF ドキュメント内の名前付きの移動先に移動します。名前付きの移動先およびその作成方法について詳しくは、[『PDF Reference』](#) の 387 ページを参照してください。

例：次の例では、ドキュメントを開き、そのドキュメント内の名前付きのジャンプ先に進みます。

```
// 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		
-----	-----------------------------------------------------------------------------------	-----------------------------------------------------------------------------------

パラメータ：[cPath]
戻り値：なし

このメソッドは、指定の FDF ファイルをインポートします。cPath パラメータには、FDF ファイルのパス名をデバイスに依存しない形式で指定します。デバイスに依存しないパス名の形式について詳しくは、『[PDF Reference](#)』の第 3.10.1 節を参照してください。このパス名の形式は、[submitForm](#) メソッドあるいはメニューの「ファイル／書き出し／フォームデータ」でエクスポートされた FDF ファイル内の /F キーの値に似ています。パス名には、現在のドキュメントの位置に対する相対パスも指定できます。このパラメータを省略すると、ファイルを選択するダイアログが表示されます。

例：「ページを開く」イベントのアクションである次のコードでは、関数 *ProcResponse* が定義されているかをチェックし、未定義の場合は、FDF ファイルにある文書レベルの JavaScript を取り込みます。

```
if (typeof ProcResponse == "undefined")
    this.importAnFDF("myDLJS.fdf");
```

ここでは、相対パス名を使用しています。この方法は、PostScript ファイルから作成した PDF ファイルに、文書レベルの JavaScript を自動的に取り込むのに便利な方法です。

[importAnXFDF](#) および [importTextData](#) も参照してください。

importAnXFDF

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

パラメータ：[cPath]
戻り値：なし

このメソッドは、XML フォームデータを含む指定の XFDF ファイルをインポートします。cPath パラメータには、XFDF ファイルのパス名をデバイスに依存しない形式で指定します。

デバイスに依存しないパス名の形式については、『[PDF Reference](#)』の第 3.10.1 節を参照してください。パス名には、現在のドキュメントの位置に対する相対パスも指定できます。パラメータを省略すると、ファイルを選択するダイアログが表示されます。

[importAnFDF](#) および [importTextData](#) も参照してください。XFDF については、[弊社の CD で提供されるマニュアル](#)の『*Forms System Implementation Notes*』を参照してください。

importDataObject

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

パラメータ : *cName*、[*cDIPath*]

戻り値 : なし

このメソッドは、外部ファイルをドキュメントにインポートし、指定の名前を「データオブジェクト」に関連付けます。このデータオブジェクトは、後で抽出したり操作をしたりすることができます。


cName は、データオブジェクトに関連付ける名前です。

cDIPath は、オプションのパラメータで、ユーザのハードドライブにあるデータファイルのパス名をデバイスに依存しない形式で指定します。このパスには、絶対パスあるいは現在のドキュメントに対する相対パスのどちらでも指定できます。*cDIPath* を指定しない場合、データファイルを選択するプロンプトが表示されます。パスの正確な構文について詳しくは、『*PDF Reference Manual*』の「*File Specification Strings*」を参照してください。

例 :

```
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"));
```

Doc オブジェクトの [dataObjects](#) プロパティ、Doc オブジェクトの [createDataObject](#)、[exportDataObject](#)、[getDataObject](#)、[importDataObject](#)、[removeDataObject](#) メソッド、および [Data オブジェクト](#) も参照してください。

セキュリティ  : *cDIPath* パラメータを指定した場合、バッチ、コンソール、メニューイベント時、あるいは外部呼び出し（OLE など）を介してしかこのメソッドを実行できなくなります。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

importIcon

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

パラメータ : *cName*、*[cDIPath]*、*[nPage]*

戻り値 : 整数

このメソッドは、アイコンをドキュメントにインポートし、それに指定の名前を関連付けます。

cDIPath は、オプションのパラメータで、ユーザのハードドライブにある PDF ファイルのパス名をデバイスに依存しない形式で指定します。このパスには、絶対パスあるいは現在のドキュメントに対する相対パスのどちらでも指定できます。*cDIPath* は、バッチ環境内あるいはコンソールからしか指定できません。パスの正確な構文について詳しくは、『[PDF Reference](#)』の第 3.10.1 節「File Specification Strings」を参照してください。

cDIPath を指定しない場合、*nPage* パラメータも無効になり、PDF ファイルを指定してページを選択するためのプロンプトが表示されます。

nPage には、アイコンとしてインポートする PDF ファイルのページ番号（0 ベース）を指定します。デフォルトは 0 です。

このメソッドは、正常に実行されたかどうかを示すコードを返します。

リターンコード	
コード	説明
0	エラーなし
1	ユーザがダイアログをキャンセル
-1	選択したファイルを開けない
-2	選択したページが無効

このメソッドは、後で利用する名前付きのアイコンを、ドキュメントにまとめて埋め込むのに便利です。例えば、ユーザがリストボックスで特定の項目を選択した場合、その項目に応じた絵をリストボックスの横に表示することができます。旧バージョンの Acrobat でこの処理を行う場合、いくつかのフィールドの表示と非表示を切り替える必要がありました。これは面倒な作業でしたが、現在では次のようなスクリプトで同じことを行うことができます。

```
var f = this.getField("StateListBox");  
var b = this.getField("StateButton");  
b.buttonSetIcon(this.getIcon(f.value));
```

この例では、同じ処理を行うために 1 つのフィールドしか使用していません。


名前付きのアイコンをドキュメントに追加するための簡単なユーザインタフェースを作成することもできます。例えば、アイコンの名前を含む *IconName* というフィールドと、アイ

コンをドキュメントに追加する `IconAdd` という 2 つのフィールドがあるとします。`IconAdd` の「マウスボタンを放す」アクションのスクリプトは、次のようになります。


```
var t = this.getField("IconName");
this.importIcon(t.value);
```

このようなスクリプトをバッチ処理で使用すると、フォルダ内で選択したすべてのアイコンファイルをドキュメントに埋め込むことができます。

[icons](#) プロパティ、[Doc オブジェクト](#)の [addIcon](#)、[getIcon](#)、[removeIcon](#) メソッド、[Field オブジェクト](#)の [buttonGetIcon](#)、[buttonImportIcon](#)、[buttonSetIcon](#) メソッド、および[アイコンオブジェクト](#)も参照してください。

セキュリティ  : `cDIPath` パラメータを指定した場合、バッチ、コンソール、またはメニューイベントでしかこのメソッドを実行できなくなります。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

importSound

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

パラメータ : `cName`、`[cDIPath]`
戻り値 : なし

このメソッドは、サウンドをドキュメントにインポートし、それに指定の名前を関連付けます。


`cName` は、Sound オブジェクトに関連付ける名前です。

`cPath` は、オプションのパラメータで、ユーザのハードドライブにあるサウンドファイルのパスをデバイスに依存しない形式で指定します。このパスには、絶対パスあるいは現在のドキュメントに対する相対パスのどちらでも指定できます。`cPath` を指定しない場合、サウンドファイルを選択するプロンプトが表示されます。パスの正確な構文について詳しくは、『[PDF Reference](#)』の第 3.10.1 節「File Specification Strings」を参照してください。

例 :

```
this.importSound("Moo");
this.getSound("Moo").play();
this.importSound("Moof", "./moof.wav");
this.getSound("Moof").play();
```

[sounds](#) プロパティ、[getSound](#)、[deleteSound](#) メソッド、および [Sound オブジェクト](#)も参照してください。

セキュリティ  : `cDIPath` パラメータを指定した場合、バッチ、コンソール、またはメニューイベントでしかこのメソッドを実行できなくなります。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

importTextData

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

パラメータ : [cPath]、[nRow]

戻り値 : なし

このメソッドは、テキストファイルからデータを 1 行インポートします。テキストファイルの各行は列ごとにタブで区切られている必要があります。最初の行はデータの列名で、PDF ファイルに存在するテキストフィールドのフィールド名と同じです。データの行番号は 0 ベースなので、データの最初の行は行番号が 0 になります (列名の行は含まれません)。データの行をインポートすると、各列のデータはそれぞれ対応するフィールドの値になります。

cPath には、テキストファイルの相対パス名をデバイスに依存しない形式で指定します。このパラメータを指定しない場合、テキストデータファイルを選択するプロンプトが表示されます。

nRow は、インポートするデータの行番号 (0 ベース) です。ただし、ヘッダ行はカウントしません。このパラメータを指定しない場合、インポートする行を選択するプロンプトが表示されます。

例 : 「First」、「Middle」、「Last」という名前のテキストフィールドがあるとします。また、テキストデータファイルの 1 行目は *First*、*Middle*、*Last* という 3 語がタブで区切られており、さらに、タブで区切られた 4 行の名前のデータが続いているとします。

First	Middle	Last
Al	Recount	Gore
George	Dubya	Bush
Alan	Cutrate	Greenspan
Bill	Outgoing	Clinton

```
// Import the first row of data from "myData.txt".
this.importTextData("/c/data/myData.txt", 0)

/* 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)
```

[ADBC オブジェクト](#) および関連するプロパティとメソッドを使用して、同じことが可能です。その場合、データファイルはスプレッドシートやデータベースでもかまいません。

insertPages

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

パラメータ : [nPage]、cPath、[nStart]、[nEnd]

戻り値 : なし


ソースドキュメントのページを現在のドキュメントに挿入します。

nPage は、ページを挿入する位置を示すページの番号 (0 ベース) です。ソースドキュメントのページはこのページの後に挿入されます。ドキュメントの先頭ページの前にページを挿入するには、*nPage* を -1 に設定します。

cPath には、挿入するページを含む PDF ファイルのパス名をデバイスに依存しない形式で指定します。デバイスに依存しないパス名の形式について詳しくは、『[PDF Reference](#)』の第 3.10.1 節を参照してください。パス名には、現在のドキュメントの位置に対する相対パスも指定できます。

nStart と *nEnd* には、挿入するソースドキュメントのページ範囲を指定します (0 ベース)。*nStart* および *nEnd* を指定しない場合、ドキュメントの全ページが対象になります。*nStart* のみを指定した場合は、*nStart* のページのみが対象になり、*nEnd* のみを指定した場合は、0 から *nEnd* までが対象になります。

[deletePages](#) および [replacePages](#) メソッドも参照してください。

セキュリティ : このメソッドは、バッチ、コンソール、またはメニューイベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

mailDoc

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

パラメータ: [*bUI*]、[*cTo*]、[*cCc*]、[*cBcc*]、[*cSubject*]、[*cMsg*]
戻り値: なし

このメソッドは、現在の PDF ドキュメントを保存し、それをすべての宛先に添付ファイルとして送信します。ユーザとのやり取りの有無は、*bUI* の値によって決まります。このパラメータが *true* (デフォルト) に設定されていると、メーラーの新規メッセージウィンドウがユーザに表示され、他のパラメータ値がセットされます。

bUI が *false* の場合、必須パラメータは *cTo* のみで、他のパラメータはオプションになります。*cTo*、*cCc*、*cBcc* パラメータに複数の宛先を指定する場合は、セミコロン「;」で区切ってください。*cSubject* および *cMsg* の長さは 64K バイトまで有効です。

例:

```
/* This will pop up the compose new message window */
this.mailDoc(true);
/* This will send out the mail with the attached PDF file to fun1@fun.com and
fun2@fun.com */
this.mailDoc(false, "fun1@fun.com", "fun2@fun.com", "", "This is the subject",
"This is the body.");
```

注意： このメソッドを Windows で使用するには、クライアントマシンで、デフォルトのメールプログラムを MAPI 対応に設定する必要があります。

mailForm

4.0			⊗
-----	--	--	---

パラメータ : *bUI*、*cTo*、*[cCc]*、*[cBcc]*、*[cSubject]*、*[cMsg]*
戻り値 : なし

このメソッドは、フォームデータをエクスポートし、その FDF ファイルをすべての宛先に添付ファイルとして送信します。ユーザとのやり取りの有無は、*bUI* の値によって決まります。このパラメータが *true* に設定されていると、メーラーの新規メッセージウィンドウがユーザに表示され、他のパラメータ値がセットされます。

bUI が *false* の場合、必須パラメータは *cTo* のみで、他のパラメータはオプションになります。*cTo*、*cCc*、*cBcc* パラメータに複数の宛先を指定する場合は、セミコロン「;」で区切ってください。*cSubject* および *cMsg* の長さは 64K バイトまで有効です。

例 :

```
/* This will pop up the compose new message window */
this.mailForm(true);
/* This will send 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.");
```

注意： このメソッドを Windows で使用するには、クライアントマシンで、デフォルトのメールプログラムを MAPI 対応に設定する必要があります。

movePage

5.0	📄		⊗
-----	---	--	---

パラメータ : *[nPage]*、*[nAfter]*
戻り値 : なし

ドキュメント内でページを移動します。

nPage は、移動するページの番号（0 ベース）で、デフォルトは 0 です。

nAfter は、ページを移動する位置を示すページの番号 (0 ベース) で、このページの後に移動されます。ドキュメントの先頭ページの前にページを移動するには、*nAfter* を -1 に設定します。デフォルトではドキュメントの最後に移動されます。

例：ドキュメントのページ順を逆にします。

```
for (i = this.numPages - 1; i >= 0; i--)  
    this.movePage(i);
```

print

パラメータ : [*bUI*]、[*nStart*]、[*nEnd*]、[*bSilent*]、[*bShrinkToFit*]、[*bPrintAsImage*]、
[*bReverse*]、[*bAnnotations*]
戻り値 : なし

このメソッドは、ドキュメントの全ページまたは特定のページを印刷します。

bUI が *true* (デフォルト) の場合、指定したパラメータが印刷ダイアログの UI にセットされてユーザに表示されるので、ユーザは足りない情報を設定したり内容を確認したりすることができます。

nStart と *nEnd* には、ページ範囲 (0 ベース) を指定します。*nStart* および *nEnd* を指定しない場合、ドキュメントの全ページが対象になります。*nStart* のみを指定した場合は、*nStart* のページのみが対象となり、*nEnd* のみを指定した場合は、0 から *nEnd* ままで対象になります。

nStart と *nEnd* を両方とも指定する場合、*bUI* は *false* でなければなりません。

bSilent が *true* の場合、キャンセルダイアログを表示せずにドキュメントを印刷します。デフォルトは *false* です。

5.0 追記事項

bShrinkToFit が *true* の場合、用紙の印刷可能領域に収まるようにするため、(必要に応じて) ページが縮小されます。*false* の場合は縮小されません。デフォルトは *false* です。

bPrintAsImage が *true* の場合、ページがイメージとして印刷されます。デフォルトは *false* です。

bReverse が *true* の場合、印刷されるページ順序が逆になります。デフォルトは *false* です。

bAnnotations が *true* (デフォルト) の場合、注釈が印刷されます。

例：

```
// This Example will print 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});
```

removeDataObject

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

パラメータ : *cName*
戻り値 : なし

このメソッドは、指定した名前に対応するデータオブジェクトをドキュメントから削除します。

cName は、削除するデータオブジェクトの名前です。

例 :

```
this.removeDataObject("MyData");
```

Doc オブジェクトの [dataObjects](#) プロパティ、Doc オブジェクトの [createDataObject](#)、[exportDataObject](#)、[getDataObject](#)、[importDataObject](#)、[removeDataObject](#) メソッド、および [Data オブジェクト](#) も参照してください。

removeField

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

パラメータ : *cName*
戻り値 : なし

cName に指定したフィールドをドキュメントから削除します。指定したフィールドが複数のページで表示されている場合は、すべて削除されます。

removeIcon

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

パラメータ : *cName*
戻り値 : なし

このメソッドは、指定した名前のアイコンをドキュメントから削除します。

[icons](#) プロパティ、[Doc オブジェクト](#) の [addIcon](#)、[getIcon](#)、[importIcon](#) メソッド、[Field オブジェクト](#) の [buttonGetIcon](#)、[buttonImportIcon](#)、[buttonSetIcon](#) メソッド、および [アイコンオブジェクト](#) も参照してください。

removeTemplate


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

パラメータ : *cName*
戻り値 : なし

この関数は、指定した名前のテンプレートをドキュメントから削除します。

cName は、削除するテンプレートの名前です。

[templates](#) プロパティ、[Data オブジェクト](#)の [createDataObject](#)、[getSound](#) メソッド、および [Template オブジェクト](#)も参照してください。

セキュリティ : このメソッドは、バッチまたはコンソールイベントでのみ実行できます。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

removeThumbnails

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

パラメータ: *[nStart]*、*[nEnd]*

戻り値: なし

指定したページのサムネールをドキュメントから削除します。

nStart と *nEnd* には、ページ範囲を指定します (0 ベース)。 *nStart* および *nEnd* を指定しない場合、ドキュメントの全ページが対象になります。 *nStart* のみを指定した場合は、*nStart* に指定したページのみが対象になり、*nEnd* のみを指定した場合は、0 から *nEnd* までが対象になります。

[addThumbnails](#) メソッドも参照してください。

removeWeblinks

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

パラメータ: *[nStart]*、*[nEnd]*

戻り値: 整数

指定したページをスキャンして、Web 上の URL にジャンプするリンクを削除します。

nStart と *nEnd* には、ページ範囲を指定します (0 ベース)。 *nStart* および *nEnd* を指定しない場合、ドキュメントの全ページが対象になります。 *nStart* のみを指定した場合は、*nStart* に指定したページのみが対象になり、*nEnd* のみを指定した場合は、0 から *nEnd* までが対象になります。

このメソッドは、ドキュメントから削除した Web リンクの数进行返します。

[addWeblinks](#) メソッドも参照してください。

注意： このメソッドは、Acrobat で UI から作成した Web リンクのみを削除します。JavaScript によって実行される Web リンク ([getURL](#) など) は削除されません。

replacePages



パラメータ : `[nPage]`、`cPath`、`[nStart]`、`[nEnd]`
戻り値 : なし


現在のドキュメントのページをソースドキュメントのページで置換します。

`nPage` は、現在のドキュメントの置換開始位置を示すページ番号 (0 ベース) で、デフォルトは 0 です。

`cPath` には、ソースとなる PDF ファイルのパス名を、デバイスに依存しない形式で指定します。デバイスに依存しないパス名の形式について詳しくは、『[PDF Reference](#)』の第 3.10.1 節を参照してください。パス名には、現在のドキュメントの位置に対する相対パスも指定できます。

`nStart` と `nEnd` には、ソースとなるドキュメントのページ範囲を指定します (0 ベース)。`nStart` および `nEnd` を指定しない場合、ドキュメントの全ページが対象になります。`nStart` のみを指定した場合は、`nStart` に指定したページのみが対象になり、`nEnd` のみを指定した場合は、0 から `nEnd` までが対象になります。

[deletePages](#)、[extractPages](#)、および [insertPages](#) メソッドも参照してください。

セキュリティ  : このメソッドは、バッチ、コンソール、またはメニューイベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

resetForm



パラメータ : `[aFields]`
戻り値 : なし

このメソッドは、ドキュメントに含まれるフィールドの値をリセットします。`aFields` パラメータを指定すると、そのフィールドのみがリセットされます。このパラメータを指定しないか値が `null` の場合、フォーム内のすべてのフィールドがリセットされます。親のフィールド名を配列に含めると、フィールドのサブツリー全体を簡単にリセットすることができます。例えば、フィールド配列の要素に「name」を指定すると、「name.first」、「name.last」などのフィールドがリセットされます。

```
var fields = new Array(2);
fields[0] = "P1.OrderForm.Description";
fields[1] = "P1.OrderForm.Qty";
this.resetForm(fields);
```

注意： フィールドをリセットすると値がデフォルトに戻り、テキストフィールドの場合は通常、空白になります。

saveAs



パラメータ : *cPath*

戻り値 : なし


このメソッドは、*cPath* で指定されるパスにファイルを保存します。*cPath* は必須パラメータであり、パス名はデバイスに依存しない形式で指定する必要があります。ファイルは最適化されずに保存されます。

例：以下のコードをバッチシーケンスとして使用することができます。フォームを含む PDF ファイルが既に開かれているものとし、データベースから取り出したデータをフォームに埋め込んでファイルを保存する場合、スクリプトのアウトラインは以下のようになります。

```
// 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");
```

例：[newDoc](#) および [addField](#) メソッドを使用してフォームを動的にレイアウトし、データベースから取得した値を埋め込み、保存することができます。

```
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
```

セキュリティ ：このメソッドは、バッチ、コンソール、またはメニューイベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

scroll

パラメータ : nX 、 nY
戻り値 : なし

このメソッドは、 nX および nY に指定された現在のページ上の点を、現在の表示の中心までスクロールします。これらの座標は[回転ユーザースペース](#)で指定する必要があります。ユーザースペース座標系については、『[PDF Reference](#)』の 126 ページを参照してください。

selectPageNthWord

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

パラメータ : $[nPage]$ 、 $[nWord]$ 、 $[bScroll]$
戻り値 : なし

現在のページ番号を $nPage$ に変更し、ページ上の指定の単語を選択します。

$nPage$ は、対象とするページ (0 ベース) です。 $nPage$ を指定しない場合、ドキュメントの最初のページを指定したことになります。

$nWord$ は、選択する単語のインデックス (0 ベース) です。 $nWord$ を指定しない場合、ページの最初の単語を指定したことになります。

$bScroll$ は、選択した単語が表示されていない場合、表示されるようにスクロールするかどうかを示します。デフォルトは true です。

[getNthTemplate](#)、[getPageNthWord](#)、および [getPageNumWords](#) メソッドも参照してください。

setPageBoxes

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

パラメータ : $[cBox]$ 、 $[nStart]$ 、 $[nEnd]$ 、 $[rBox]$
戻り値 : なし

指定ページの名前付きボックスを囲む矩形を設定します。

$nStart$ と $nEnd$ は、対象となるドキュメントのページ番号 (0 ベース) です。 $nStart$ および $nEnd$ を指定しない場合、ドキュメントの全ページが対象になります。 $nStart$ のみを指定した場合は、 $nStart$ に指定したページのみが対象になります。

$cBox$ には、「Art」、「Bleed」、「Crop」、「Media」、「Trim」のいずれかを指定できます。「BBox」は読み取り専用であり、[getPageBox](#) でのみサポートされます。これらのボックスの定義については、『[PDF Reference](#)』の 524 ページにある第 8.6.1 節「Page Boundaries」を参照してください。

rBox は、4 つの数値の配列で、ボックスのサイズを [回転ユーザスペース](#) で指定します。*rBox* を指定しない場合、ボックスは削除されます。

[getPageBox](#) メソッドも参照してください。

setPageLabels



パラメータ : *[nPage]*、*[aLabel]*

戻り値 : なし

このメソッドは、指定したページにページ番号のスタイルを設定します。ページ番号のスタイルは、別のスタイルが設定されているページまで有効となります。

nPage は、スタイルを設定するページ番号です (0 ベース)。

aLabel を指定しない場合、指定したページから次にページ番号スタイルが設定されているページまで、スタイルの設定が解除されます。

aLabel を指定する場合、3 つの要素の配列 *[cStyle, cPrefix, nStart]* を指定します。

cStyle は、ページ番号のスタイルで、「D」(10 進数)、「R」、「r」(ローマ数字の大文字／小文字)、「A」、「a」(英字の大文字／小文字) のいずれかを指定します。これらのスタイルの正確な定義については、『[PDF Reference](#)』の第 7.3.1 節を参照してください。

cPrefix は、ページラベルの数字部分の前に付ける文字列です。

nStart は、指定したページラベルの開始番号を示す序数です。

例: 10 ページのドキュメントがあり、最初の 3 ページには小文字のローマ数字、次の 5 ページには数字 (1 で開始)、最後の 2 ページには「付録 -」の接頭辞と英字のページラベルを付けるとします。

```
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);
```

この結果、コンソールには次のよう出力されます。

```
i, ii, iii, 1, 2, 3, 4, 5, Appendix-A, Appendix-B
```

例: ドキュメントからページラベルをすべて削除します。

```
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);
    }
}

```

[getPageLabel](#) メソッドも参照してください。

setPageRotations

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

パラメータ : *[nStart]*、*[nEnd]*、*[nRotate]*
 戻り値 : なし

現在のドキュメントの指定のページを回転します。

nStart と *nEnd* には、対象となるドキュメントのページ範囲を指定します (0 ベース)。 *nStart* および *nEnd* を指定しない場合、ドキュメントの全ページが対象になります。 *nStart* のみを指定した場合は、*nStart* に指定したページのみが対象になり、*nEnd* のみを指定した場合は、0 から *nEnd* までが対象になります。

nRotate には、対象ページに適用する回転角度を指定します。0、90、180、270 のいずれかを指定してください。 *nRotate* を指定しない場合、0 を指定したことになります。

[getPageRotation](#) メソッドも参照してください。

setPageTransitions

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

パラメータ : *[nStart]*、*[nEnd]*、*[aTrans]*
 戻り値 : なし

nStart と *nEnd* には、対象となるドキュメントのページ範囲を指定します (0 ベース)。 *nStart* および *nEnd* を指定しない場合、ドキュメントの全ページが対象になります。 *nStart* のみを指定した場合は、*nStart* に指定したページのみが対象になります。

aTrans を指定しない場合、指定したページの画面切り替え設定がすべて削除されます。

画面表示の切り替え方法は、*[nDuration, cTransition, nTransDuration]* という 3 つの値の配列で指定します。

nDuration は、ビューアが自動的にページを切り替えるまでの最大表示時間です。 *nDuration* を -1 に設定すると、ページは自動的に切り替わらなくなります。

cTransition は、画面が切り替わる際にページに適用される効果の名前です。効果の有効な値については、[transitions](#) を参照してください。

nTransDuration は、効果の継続時間（秒単位）です。

[getPageTransition](#) メソッドも参照してください。

spawnPageFromTemplate



パラメータ : *cTemplate*、*[nPage]*、*[bRename]*、*[bOverlay]*

戻り値 : なし

このメソッドでは、[getNthTemplate](#) などから返されるテンプレート名 *cTemplate* を使用します。オプションの *nPage* には、テンプレートを適用するページの番号（0 ベース）を指定します。指定したページが既に存在する場合は、テンプレートの内容がそのページに追加されます（ただし、パラメータ *bOverlay* を参照）。*nPage* を省略した場合、新しいページがドキュメントの最後に追加されます。オプションのパラメータ *bRename* は、フィールド名を変更する必要があるかどうかを示すブーリアンで、デフォルトは *true* です。

4.0 追記事項

bOverlay が *false* の場合、*nPage* で指定したページ上ではなく、そのページの前にテンプレートが挿入されます。*bOverlay* のデフォルトは *true* です。

例 :

```
var n = this.numTemplates;
var cTempl;
for (i = 0; i < n; i++) {
    cTempl = this.getNthTemplate(i);
    this.spawnPageFromTemplate(cTempl);
}
```

[Doc オブジェクト](#) の [templates](#) プロパティ、[createTemplate](#) メソッド、および今後のバージョンでこのメソッドに代わって使用される [Template オブジェクト](#) の [spawn](#) メソッドを参照してください。

注意： テンプレート機能は、Acrobat Reader では動作しません。

submitForm

--	--	--	--

パラメータ : *cURL*、*[bFDF]*、*[bEmpty]*、*[aFields]*、*[bGet]*、*[bAnnotations]*、*[bXML]*、*[blncrChanges]*、*[bPDF]*、*[bCanonical]*、*[bExclNonUserAnnots]*、*[bExclFKey]*、*[cPassword]*

戻り値 : なし

このメソッドは、フォームを特定の URL に送信します。最初のパラメータ *cURL* は、送信先の URL です。送信した結果を FDF としてブラウザウィンドウ内のドキュメントに返す場合、*cURL* の文字列の最後に「#FDF」を付ける必要があります。

オプションの *bFDF* パラメータは、FDF または HTML のどちらの形式で送信するかを示すブーリアンです。このパラメータを *true* (デフォルト) に設定すると、フォームデータを FDF として送信します。*false* の場合は、HTML (URL エンコード) として送信します。

オプションの *bEmpty* パラメータはブーリアンで、*true* の場合は、値のないフィールドも含めてすべてのフィールドを送信し、*false* の場合は、値がないフィールドを除外します。*bEmpty* のデフォルトは *false* です。

オプションの *aFields* パラメータは、送信するフィールド名の配列、または 1 つのフィールド名を含む文字列です。このパラメータを指定した場合は、*bEmpty* で除外されるフィールドを除いて、指定のフィールドのみが送信されます。このパラメータを省略した場合または値が *null* の場合は、フォームに含まれるすべてのフィールド (*bEmpty* で除外されるフィールドを除く) が送信されます。

4.0 追記事項

オプションの *bGet* パラメータはブーリアンで、*true* の場合は送信に GET HTTP メソッドを使用し、*false* (デフォルト) の場合は POST を使用します。GET を使用できるのは、送信に Acrobat Web Capture を使用し (ブラウザのインタフェースでサポートされるのは POST のみ)、HTML としてデータを送信する場合 (つまり、パラメータ *bFDF*、*bXML*、および *bPDF* がすべて *false*) のみです。

5.0 追記事項

オプションの *bAnnotations* パラメータはブーリアンで、*true* の場合は、送信する FDF または XML に注釈を含めます。デフォルトは *false* です。*bFDF* または *bXML* が *true* の場合のみ適用されます。

オプションの *bXML* パラメータはブーリアンで、*true* の場合は XML として送信することを示します。デフォルトは *false* です。

オプションの *blncrChanges* パラメータはブーリアンで、*true* の場合は、PDF に加えた変更内容を FDF に含めて送信します。デフォルトは *false* です。*bFDF* が *true* の場合のみ適用されます。Acrobat Reader では使用できません。

オプションの *bPDF* パラメータが *true* の場合は、完全な PDF ドキュメントをそのまま送信します。デフォルトは *false* です。*bPDF* が *true* の場合、他に設定が必要なパラメータは *cURL* のみになります。Acrobat Reader では使用できません。

オプションの *bCanonical* パラメータはブーリアンで、*true* の場合は、すべての日付を標準の形式に変換して送信します (D:YYYYMMDDHHmmSSOHH'mm' など。詳しくは『[PDF Reference](#)』を参照)。デフォルトは *false* です。

オプションの *bExclNonUserAnnots* パラメータはブーリアンで、*true* の場合は、作成者が現在のユーザとは違う注釈をすべて除外します。デフォルトは *false* です。

オプションの *bExclFKey* パラメータはブーリアンで、*true* の場合は「F」キーを除外します。デフォルトは *false* です。

FDF を暗号化してから送信する必要がある場合は、暗号鍵の生成に使用するパスワード *cPassword* を指定するか、ブーリアンを渡す必要があります。*cPassword* が *true* (引用符は付けません) の場合、パスワードの入力を指示するダイアログが表示されます。ただし、暗号化された FDF をユーザが送信または受信するときに (その Acrobat セッション内で) 既にパスワードを入力している場合、このダイアログは表示されません (この場合、入力済のパスワードが使用されます)。パスワードがプログラムから渡されたかダイアログで入力されたかに関係なく、この新しいパスワードが記憶され、以降の暗号化された FDF の送受信で使用されます。このパラメータは、*bFDF* が *true* の場合のみ有効です。

親のフィールド名を *aFields* パラメータの配列または文字列に含めると、フィールドのサブツリー全体を簡単に送信することができます。

例 :

```
/* Submit the form to the server */
this.submitForm("http://myserver/cgi-bin/myscript.cgi#FDF");
/* Or */
this.submitForm("http://myserver/cgi-bin/myscript.cgi#FDF",
    true, false, "name");
```

上の例は、フィールドのサブツリー全体を簡単に送信する方法を示しています。*aFields* パラメータに「name」を指定すると、「name.title」、「name.first」、「name.middle」、「name.last」などが送信されます。

例 :

```
this.submitForm({
    cURL: "http://myserver/cgi-bin/myscript.cgi#FDF",
    bXML: true
});
```

注意： submitForm メソッドを使用する場合、Web ブラウザ内でドキュメントを表示しているか、Acrobat Web Capture プラグインがインストールされている必要があります。ただし、URL に「mailto」スキームを使用する場合、SendMail プラグインが存在していれば、Web ブラウザ内でドキュメントを開いている必要はありません。

https プロトコルの使用がサポートされているので、セキュリティ保護された接続が可能です。

syncAnnotScan

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

パラメータ：なし

戻り値：なし

ドキュメント全体の注釈を表示または処理するには、すべての注釈を検出する必要があります。これは時間を要する処理なので、アイドル時にすべてのページを調べて注釈を検索するバックグラウンドタスクを実行するのが一般的です。このメソッドを呼び出すと、このメソッドから戻るまでにすべての注釈が確実にスキャンされます。

例：

```
this.syncAnnotScan();
annotations = this.getAnnots({nSortBy:ANSB_Author});
// now, do something with the annotations.
```

このコードの 2 行目は、syncAnnotScan から戻るまで、つまり、ドキュメントの注釈のスキャンが完了するまでは実行されません。

注意： バックグラウンドのスキャン処理によって注釈の完全なリストが取得されていなくても、ほとんどの注釈関連のコードは問題なく動作します。syncAnnotScan は、注釈の完全なリストを必要とする場合に実行するのが一般的です。

[getAnnots](#) も参照してください。

Event オブジェクト

すべての JavaScript は特定のイベントの結果として実行されます。各イベントにはタイプと名前があります。ここでは、イベントをタイプと名前の対にして詳しく説明します。

Acrobat JavaScript では、イベントごとに *event* オブジェクトが作成されます。各イベントの発生時にはこの *event* オブジェクトにアクセスすることが可能で、イベントの現在の状態を取得したり操作したりすることができます。

JavaScript のスクリプト作成時は、イベントが発生するタイミングとその処理順序を把握しておく必要があります。一部のメソッドまたはプロパティは特定のイベント時にしかアクセスできないので、イベントに関する知識が必要です。

イベントのタイプ／名前の組み合わせ

App / Init

ビューアを起動すると、アプリケーション初期化イベントが発生し、[フォルダレベルの JavaScript](#) と呼ばれるスクリプトファイルが、アプリケーションおよびユーザの JavaScripts フォルダから読み込まれます。スクリプトファイルは、*Config.js*、*glob.js*、他のすべてのファイル、ユーザファイルの順にロードされます。

このイベントは、イベントオブジェクトの [name](#) および [type](#) プロパティを定義します。

このイベントは、[rc](#) リターンコードに応答しません。

Batch / Exec

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

バッチイベントは、バッチシーケンスで各ドキュメントが処理される時に発生します。バッチシーケンスの一部として作成された JavaScript では、実行時にこのイベントオブジェクトにアクセスすることができます。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答します。リターンコードが *false* に設定されていると、バッチシーケンスは停止します。

Bookmark / Mouse Up

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

このイベントは、JavaScript が割り当てられているしおりをユーザがクリックするたびに発生します。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、クリックしたしおりオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

Console / Exec

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

コンソールイベントは、ユーザが JavaScript をコンソールで評価するたびに発生します。

このイベントは、イベントオブジェクトの [name](#) および [type](#) プロパティを定義します。

このイベントは、[rc](#) リターンコードに応答しません。

Doc / DidPrint

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

このイベントは、ドキュメントの印刷後に発生します。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

Doc / DidSave

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

このイベントは、ドキュメントの保存後に発生します。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

Doc / Open

このイベントは、ドキュメントを開くたびに発生します。ドキュメントを開くと、文書レベルのスクリプト関数がスキャンされ、公開されたスクリプトを実行できるようになります。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、[targetName](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

Doc / WillClose

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

このイベントは、ドキュメントを閉じる前に発生します。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

Doc / WillPrint

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

このイベントは、ドキュメントの印刷前に発生します。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

Doc / WillSave

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

このイベントは、ドキュメントの保存前に発生します。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

External / Exec

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

このイベントは、OLE や AppleScript などを使用した外部アクセス、または FDF をロードした結果として発生します。

このイベントは、イベントオブジェクトの [name](#) および [type](#) プロパティを定義します。

このイベントは、[rc](#) リターンコードに応答しません。

Field / Blur

4.05			
------	--	--	--

blur イベントは、フィールドがフォーカスを失うとき、すべてのイベントの最後に発生します。フォーカスの消失がマウスクリックによるものでなくても（例えば Tab キーの使用によっても）このイベントは発生します。

このイベントは、イベントオブジェクトの [modifier](#)、[name](#)、[shift](#)、[target](#)、[targetName](#)、[type](#)、および [value](#) プロパティを定義します。イベントのターゲットとなるのは、検証スクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答しません。

Field / Calculate

このイベントは、フォームに加えられた変更によって、計算スクリプトが割り当てられているすべてのフィールドで計算が実行されるときに発生します。変更されたフィールドの値に関連するフィールドは、この時点ですべて再計算されます。これらのフィールドでは、[Field / Validate](#)、[Field / Blur](#)、および [Field / Focus](#) イベントが順次発生します。

計算フィールドは、他の計算フィールドの計算結果に依存していることがあります。計算順序の配列には、ドキュメント内で計算スクリプトが割り当てられているすべてのフィールドが順番にリストされています。フィールド全体の計算が必要とされる場合、この配列インデックスの 0 番目から最後まで、各フィールドが順に計算されます。

フィールドの計算順序を変更するには、Adobe Acrobat メニューの ツール／フォーム／フィールドの計算順序の設定を使用します。

このイベントは、イベントオブジェクトの [name](#)、[source](#)、[target](#)、[targetName](#)、[type](#)、および [value](#) プロパティを定義します。イベントのターゲットとなるのは、計算スクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答します。リターンコードが *false* に設定されているとフィールドの値は変更されず、*true* に設定されていると [value](#) プロパティの値が使用されます。

Field / Focus

4.05			
------	--	--	--

focus イベントは、フィールドをマウスでクリックしてフォーカスを移した後、マウスボタンを放す前に発生します。フィールドをアクティブにしたのがマウスクリックでなくても（例えば Tab キーの使用によっても）このイベントは発生します。ユーザがフィールドの操作を始める前にしておきたい前処理がある場合、このとき行うのが最適です。

このイベントは、イベントオブジェクトの [modifier](#)、[name](#)、[shift](#)、[target](#)、[targetName](#)、[type](#)、および [value](#) プロパティを定義します。イベントのターゲットとなるのは、検証スクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答しません。

Field / Format

format イベントは、関連する計算がすべて終了した後に発生します。このイベントでフィールドに割り当てた JavaScript を実行し、データ値の表示方法（表現または外観とも呼ぶ）を変更することができます。例えばデータ値が数値で、これを通貨として表示したい場合、フォーマットスクリプトで値の前にドル記号（\$）を付け、小数点以下 2 桁までに制限することができます。

このイベントは、イベントオブジェクトの [commitKey](#)、[name](#)、[target](#)、[targetName](#)、[type](#)、[value](#)、および [willCommit](#) プロパティを定義します。イベントのターゲットとなるのは、フォーマットスクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答しません。しかし [value](#) プロパティを使用して、フィールドの値をフォーマットすることができます。

Field / Keystroke

keystroke イベントは、テキストボックスまたはコンボボックスフィールドにユーザがキー入力（切り取りおよび貼り付け操作も含む）をするか、コンボボックスまたはリストボックスフィールドの項目を選択するたびに発生します。キーストロークスクリプトでは、ユーザが使用できるキーの種類を制限することができます。例えば、数値のフィールドでは、数値のみを入力可能にすることができます。

リストボックスフィールドでは、ユーザインタフェースから選択の変更スクリプトを割り当てることができます。このスクリプトは項目を選択するたびに実行されますが、この動作は *keystroke* イベントとして実装されており、ユーザの選択した項目がキー入力値に相当します。コンボボックスも同様で、ドロップダウンリストで選択した値がテキストフィールドに貼り付けられる動作を「キー入力」と考えることができます。

検証イベントが発生する前に、キーストロークスクリプトの最後の呼び出しが行われ、[willCommit](#) プロパティが *true* に設定されます。キー入力が行われている段階で、値が確定される前にフィールド値の最終チェックを行うと便利です。このようにすると、キー入力単位で部分的にしかチェックできないような、特に複雑なフォーマットを効果的に処理することができます。

このイベントは、イベントオブジェクトの [commitKey](#)、[change](#)、[changeEx](#)、[keyDown](#)、[modifier](#)、[name](#)、[selEnd](#)、[selStart](#)、[shift](#)、[target](#)、[targetName](#)、[type](#)、[value](#)、および [willCommit](#) プロパティを定義します。イベントのターゲットとなるのは、キーストロークスクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答します。リターンコードが *false* に設定されていると、キー入力は無視されます。入力された文字をスクリプトで置換したい場合は、[change](#) プロパティをキー入力として使用できます。フィールド内の現在のテキスト選択範囲は、[selEnd](#) および [selStart](#) プロパティで変更できます。

Field / Mouse Down

mouse down イベントは、フォームフィールドをクリックして、マウスボタンがまだ押されている状態のときに発生します。このイベント時に実行する処理はできるだけ少なくしてください（短い音を鳴らすなど）。*mouse enter* イベントがまだ発生していない場合、*mouse down* イベントは発生しません。

このイベントは、イベントオブジェクトの [modifier](#)、[name](#)、[shift](#)、[target](#)、[targetName](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、検証スクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答しません。

Field / Mouse Enter

mouse enter イベントは、フィールドを囲む矩形内にマウスポインタを移動したときに発生します。このイベントは、テキストフィールドにヘルプを表示するというような処理に適しています。

このイベントは、イベントオブジェクトの [modifier](#)、[name](#)、[shift](#)、[target](#)、[targetName](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、検証スクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答しません。

Field / Mouse Exit

mouse exit イベントは *mouse enter* イベントの逆で、フィールドを囲む矩形外にマウスポインタを移動したときに発生します。*mouse enter* イベントがまだ発生していない場合、*mouse exit* イベントは発生しません。

このイベントは、イベントオブジェクトの [modifier](#)、[name](#)、[shift](#)、[target](#)、[targetName](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、検証スクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答しません。

Field / Mouse Up

mouse up イベントは、フォームフィールドをクリックして、マウスボタンを放したときに発生します。このイベントは、フォームを送信するアクションなどを割り当てるのに適しています。*mouse down* イベントがまだ発生していなければ、*mouse up* イベントは発生しません。

このイベントは、イベントオブジェクトの [modifier](#)、[name](#)、[shift](#)、[target](#)、[targetName](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、検証スクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答しません。

Field / Validate

フィールドタイプに関係なく、ユーザによって不適切な値がフィールドに入力されることがあります。ユーザがフィールドの外側をクリックしたり、別のフィールドにタブキーで移動したり、Enter キーを押したりすると、入力された新しいデータ値が確定されます。

validate イベントは、値が確定されてから最初に発生するイベントなので、これを利用して入力された値が正しいかを JavaScript で確認することができます。*validate* イベントが正常に処理されると、次のイベントである *calculate* イベントが発生します。

このイベントは、イベントオブジェクトの [change](#)、[changeEx](#)、[keyDown](#)、[modifier](#)、[name](#)、[shift](#)、[target](#)、[targetName](#)、[type](#)、および [value](#) プロパティを定義します。イベントのターゲットとなるのは、検証スクリプトを実行しているフィールドです。

このイベントは、[rc](#) リターンコードに応答します。リターンコードが *false* に設定されていると、フィールドの値は無効であるとみなされ、値は変更されません。

Link / Mouse Up

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

このイベントは、JavaScript アクションを含むリンクをユーザがアクティブにすると発生します。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

Menu / Exec

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

メニューイベントは、メニュー項目に割り当てられた JavaScript を実行するたびに発生します。Acrobat 5.0 では、メニュー項目を追加し、それに JavaScript アクションを関連付けることができます。次に例を示します。

```
app.addItem({ cName: "Hello", cParent: "File",  
  cExec: "app.alert('Hello',3);", nPos: 0});
```

「`app.alert('Hello',3);`」というスクリプトは、メニューイベントで実行されます。メニューイベントを発生させるには、次の 2 通りの方法があります。

1. ユーザインタフェース。ユーザがメニュー項目をクリックすると、スクリプトが実行されます。
2. プログラム。(ボタンフィールドの *mouse up* イベントなどで)
`app.execMenuItem("Hello")` が実行されると、このスクリプトが実行されます。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、[targetName](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、現在アクティブなドキュメント（開いている場合）です。

このイベントは、メニュー項目に `cEnable` または `cMarked` プロシージャが設定されている場合、[rc](#) リターンコードに応答します。リターンコードが `false` の場合は、メニュー項目が使用不可になるかメニューのチェックが解除されます。リターンコードが `true` の場合は、イベント処理が行われます。

Page / Open

4.05			
------	--	--	--

このイベントは、新しいページが表示されてページが描画された後に発生します。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

Page / Close

4.05			
------	--	--	--

このイベントは、表示中のページが現在のページでなくなる場合に発生します（新しいページに切り替えられたり、ドキュメントが閉じられたときなど）。

このイベントは、イベントオブジェクトの [name](#)、[target](#)、および [type](#) プロパティを定義します。イベントのターゲットとなるのは、ドキュメントオブジェクトです。

このイベントは、[rc](#) リターンコードに応答しません。

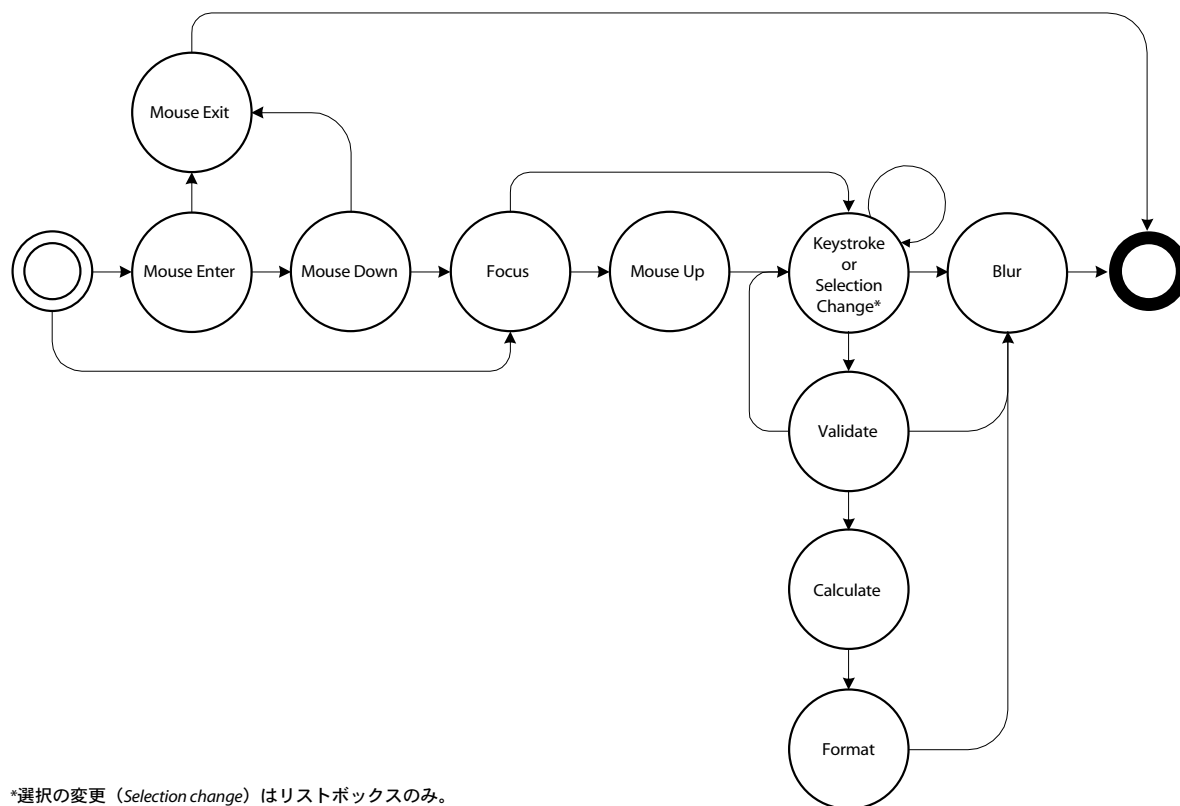
ドキュメントイベント処理

ドキュメントを開くと、[Doc / Open](#) イベントが発生し、関数がスキャンされ、公開されているスクリプトがすべて実行されます。次に、PDF ファイルの `NeedAppearances` キーが *AcroForm* デクシオナリで `true` に設定されている場合、ドキュメント内のすべてのフィールドのフォーマットスクリプトが実行されます（`NeedAppearances` キーおよび *AcroForm* デクシオナリについて詳しくは、『[PDF Reference](#)』の第 3.6.1 および 7.6.1 節を参照）。そして最後に、[Page / Open](#) イベントが発生します。

注意： `NeedAppearances` キーが `true` に設定されたフォームフィールドを含む PDF を作成する場合、Web サイトなどで公開する前に必ず「名前を付けて保存」を実行してください。「名前を付けて保存」を実行すると、フォームの外観が生成されてファイルの中に保存されます。これにより、Web ブラウザにファイルをロードする Reader のパフォーマンスが向上します。

フォームイベント処理

次の図に、フォームイベントの発生順序を示します。この図では、注意すべき特定の依存関係を示しています。例えば、Focus イベントが発生していなければ、Mouse Up イベントが発生することはありません。



*選択の変更 (Selection change) はリストボックスのみ。

Event オブジェクトのプロパティ

change

型：文字列

アクセス：R/W

このプロパティは、ユーザが入力した値の変更を示します。変更に対しては置換を行うことができます。例えば、JavaScript で特定の文字を置き換えることが可能です。値の変更は、個々のキー入力であったり、あるいは文字列の入力（フィールドに貼り付けを行う場合など）であったりします。

changeEx

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

型：各種

アクセス：R

このプロパティは、変更の書き出し値を含み、リストボックスおよびコンボボックスの [Field / Keystroke](#) イベントでのみ使用できます（リストボックスの場合は、プロパティダイアログの「選択の変更」タブでキーストロークスクリプトを入力します）。リストボックスについては以上ですが、コンボボックスについてはもう少し説明が必要です。

コンボボックスの場合、[changeEx](#) プロパティは、コンボボックスのドロップダウンリストを使用している場合のみ使用できます（マウスまたはキーボードで項目を選択するなど）。また、コンボボックスが編集可能で、ユーザが入力操作を行った場合、[Field / Keystroke](#) イベントはテキストフィールドの場合と同様です（つまり、[changeEx](#) または [keyDown](#) イベントプロパティはありません）。

commitKey

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

型：数値

アクセス：R

このプロパティは、フォームフィールドからフォーカスが移動する際の処理を決定するために使用できます。有効な値は次の通りです。

- 0 - 値が確定されなかった（Esc キーが押されたなど）
- 1 - フィールドの外側がクリックされたことにより値が確定された
- 2 - Enter キーにより値が確定された
- 3 - タブキーによって別のフィールド移動したことで値が確定された

例えば、値の確定後に警告ダイアログを自動的に表示するには、次のコードをフィールドのフォーマットスクリプトに追加します。

```
if (event.commitKey != 0)
    app.alert("Thank you for your new field value.");
```

keyDown

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

型：ブーリアン

アクセス：R

このプロパティは、リストボックスとコンボボックスのキーストロークイベントでのみ使用でき、項目の選択に矢印キーを使用した場合に *true*、それ以外の場合は *false* になります。リストボックスについては以上ですが、コンボボックスについてはもう少し説明が必要です。

コンボボックスの場合、[keyDown](#) プロパティは、コンボボックスのドロップダウンリストを使用している場合のみ使用できます（マウスまたはキーボードで項目を選択するなど）。また、コンボボックスが編集可能で、ユーザが入力操作を行った場合、[Field / Keystroke](#) イベントはテキストフィールドの場合と同様です（つまり、[changeEx](#) または [keyDown](#) イベントプロパティはありません）。

modifier

型 : ブーリアン

アクセス : R

このプロパティは、特定のイベントで修飾キーが押されているかどうかを判断するブーリアンです。Microsoft Windows プラットフォームの修飾キーは Ctrl キーで、Macintosh プラットフォームの修飾キーは Option または Command キーです。[modifier](#) プロパティは、UNIX ではサポートされていません。

name

4.05			
------	--	--	--

型 : 文字列

イベント : すべて

アクセス : R

このプロパティは、テキスト文字列による現在のイベント名です。イベントは、[type](#) (タイプ) と名前で一意に識別されます。有効な名前には、Keystroke、Validate、Focus、Blur、Format、Calculate、Mouse Up、Mouse Down、Mouse Enter、Mouse Exit、Open、Close、Will Save、Did Save、Will Print、Did Print、Init、および Exec があります。

rc

型 : ブーリアン イベント : Keystroke、Validate、Menu

アクセス : R/W

このプロパティは検証に使用され、一連のイベントで特定のイベントを成功させるかどうかを示します。値の変更や確定を不可にするには、*rc* を *false* に設定します。デフォルトでは、*rc* は *true* に設定されています。

selEnd

型 : 整数

アクセス : R/W

このプロパティは、キーストロークイベントで現在テキストが選択されている範囲の終了位置を示します。

selStart

型 : 整数

アクセス : R/W

このプロパティは、キーストロークイベントで現在テキストが選択されている範囲の開始位置を示します。

shift

型 : ブーリアン

アクセス : R

このプロパティは、特定のイベントで Shift キーが押されているかどうかを判断するブーリアンです。

source

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

型：オブジェクト

アクセス：R

このプロパティには、計算イベントを発生させた Field オブジェクトが含まれます。これは通常、イベントのターゲット、つまり計算中のフィールドとは異なります。

target

型：オブジェクト

アクセス：R

このプロパティには、イベントを発生させたターゲットオブジェクトが含まれます。mouse、focus、blur、calculate、validate、format のすべてのイベントでは、イベントを発生させた [Field オブジェクト](#) になります。page open および page close など他のイベントでは、ドキュメントまたは [this オブジェクト](#) になります。

targetName

型：文字列

アクセス：R

このプロパティは、実行中の JavaScript 名を返そうと試みます。これをデバッグに使用すると、例外の原因となるコードの識別に役立ちます。targetName の一般的な値は、次の通りです。

- App : Init イベントで実行されるフォルダレベルのスクリプトファイル名
- Doc : Exec イベントで実行される文書レベルのスクリプト名
- Batch : Exec イベントで処理中の PDF ファイル名
- Field : Exec イベントが発生したフィールド名
- Menu : Exec イベントが発生したメニュー項目名

識別可能な名前が存在すると、例外発生時に Acrobat EScript から targetName が通知されます。

例：フォルダレベルのスクリプトファイル *conserve.js* の最初の行にエラーが含まれており、Acrobat ビューアを起動した時点で例外が発生するとします。メッセージには、問題の原因が明確に示されています。

```
uncaught exception: conserve.js:App:Init:1: Missing required argument for
App.alert. ==> Parameter cMsg.
```

type

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

型：文字列

アクセス：R

このプロパティは、テキスト文字列による現在のイベントタイプです。イベントは、タイプと [name](#)（名前）で一意に識別されます。有効なタイプには、Batch、Console、App、Doc、Page、External、Bookmark、Link、Field、および Menu があります。

value

型：各種

アクセス：R/W

[Field / Validate](#) イベントの場合、フィールドで確定された値が *value* になります。現在のフィールドの値がそのフィールドの *value* プロパティです。コンボボックスの場合は、表示されている値であり、書き出し値ではありません（書き出し値については、[changeEx](#) プロパティを参照）。

例えば、次の JavaScript は、フィールドの値が 0 ～ 100 の範囲内であることを検証します。

```
if (event.value < 0 || event.value > 100) {  
    app.beep(0);  
    app.alert("Invalid value for field " + event.target.name);  
    event.rc = false;  
}
```

[Field / Calculate](#) イベントの場合、このプロパティを JavaScript で設定すると、イベント完了時のフィールド値になります。例えば、次の JavaScript は、フィールドの値を SubTotal フィールドの値 + 税金に設定します。

```
var f = this.getField("SubTotal");  
event.value = f.value * 1.0725;
```

[Field / Format](#) イベントの場合、このプロパティを JavaScript で設定すると、フィールド値をフォーマットすることができます。デフォルトでは、このプロパティはユーザが確定した値を含みます。コンボボックスの場合は、表示されている値であり、書き出し値ではありません（書き出し値については、[changeEx](#) プロパティを参照）。

例えば、次の JavaScript は、フィールドの値を通貨の形式にフォーマットします。

```
event.value = util.printf("$%.2f", event.value);
```

[Field / Keystroke](#) イベントの場合、このプロパティはフィールドの現在値を示します。例えば、テキストフィールドの場合、フィールドにキー入力する直前のテキストを示します。

[Field / Blur](#) および [Field / Focus](#) イベントの場合は、フィールドの現在値になります。

5.0 追記事項

複数項目を選択可能なリストボックスを操作して ([Field オブジェクト](#)の `multipleSelection` プロパティを参照)、そのフィールド値が配列になった場合 (複数の項目が選択された場合)、`event.value` の値を取得しようとするすると空の文字列が返されます。この場合、値の設定を行うこともできません。しかしこの動作は妥当なものと言えます。なぜなら、`event.value` は主に [Field / Validate](#) と [Field / Format](#) イベントのために用意されたものであり、これらはリストボックスではあまり意味を持たないからです。しかし、[Field / Calculate](#)、[Field / Focus](#)、および [Field / Blur](#) イベントでも `field.value` の設定ができますから、リストボックスで複数項目の選択を許す場合には [value](#) を使用することができます。

willCommit

型 : ブーリアン

アクセス : R

このプロパティを使用して、データが確定される前に現在のキー入力を検証することができます。例えば、数値データ用のフォームフィールドに文字データが入力されなかったか等をチェックするのに便利です。JavaScript は、最後の `keystroke` イベントの後、かつフィールドの検証前に、このプロパティを `true` に設定します。

例 :

```
var value = event.value
if (event.willCommit)
    // Final value checking.
else
    // Keystroke level checking.
```

Field オブジェクト

Field オブジェクトは、Acrobat フォームフィールド（Acrobat フォームツールまたは [Doc オブジェクト](#) の [addField](#) メソッドを使用して作成したフィールド）を表します。フィールドの作成者が、境界線の色やフォントなど既存のフィールドプロパティを変更するのと同様に、JavaScript ユーザは Field オブジェクトを使用してフィールドに変更を加えることができます。

ここで示すさまざまなプロパティやメソッド、そのコード例に加え、「[どうすればフォームフィールドをプログラムで作成できますか？](#)」および「[クイックリファレンス：フォーム](#)」の節では、さらに詳しい例と、JavaScript によるフォームフィールドの作成 / 制御方法を示しています。

JavaScript によるフィールドへのアクセス

フィールドにアクセスするには、[Doc オブジェクト](#) メソッドのインタフェースに用意されているメソッドを使用して、フィールドを JavaScript 変数に「結び付ける」必要があります。Field オブジェクトのプロパティを変更したり、メソッドにアクセスしたりしているうちに、1 つのフィールドに複数の変数が結び付けられることがあります。フィールドへの変更は、そのフィールドに結び付けられているすべての変数に影響します。

```
var f = this.getField("Total");
```

この例のようにすると、変数 *f* を介して *Total* というフォームフィールドを操作できるようになります。

Field オブジェクトのプロパティ

alignment



型：文字列

フィールド：テキスト

アクセス：R/W

このプロパティは、テキストフィールドにおけるテキストの整列方法を示します。有効な整列方法は、「left」、「center」、および「right」です。

```
var f = this.getField("MyText");  
f.alignment = "center";
```

borderStyle



型：文字列

フィールド：すべて

アクセス：R/W

このプロパティは、フィールドの境界線のスタイルを示します。有効な境界線のスタイルは、*solid*、*dashed*、*beveled*、*inset*、および *underline* です。境界線のスタイルにより、矩形の境界線の描画方法が決まります。

- *solid* スタイルは、矩形の周囲を実線で描きます。
- *dashed* スタイルは、周囲を破線で描きます。
- *beveled* スタイルは *solid* スタイルと似ていますが、実線の内側に斜影（浮き出た外観、ベベル）が加えられます。
- *inset* スタイルは、実線スタイルと似ていますが、実線の内側に切り込み（くぼんだ外観）が加えられます。
- *underline* スタイルは、矩形の底辺（下線）を描きます。

border オブジェクトは便利な静的定数で、フィールドの境界線スタイルをすべて定義しています。次の例では、フィールドの境界線スタイルを実線に設定します。

```
var f = this.getField("MyField");
f.borderStyle = border.s; /* border.s evaluates to "solid" */
```

次の表に、*borderStyle* プロパティとそれに関連するキーワードを定義します。

タイプ	キーワード
solid	border.s
beveled	border.b
dashed	border.d
inset	border.i
underline	border.u

buttonAlignX

5.0

型：整数

フィールド：ボタン

アクセス：R/W

buttonAlignX プロパティは、ボタン上でアイコンの左に配分するスペースの割合を示します。値は 0 ～ 100 パーセントが有効で、デフォルト値は 50 です。アイコンのサイズ変更で縦横比が保たれない場合（アイコン左右のスペースに違いはなくなるので）、このプロパティは使用されません。

buttonAlignY

5.0

型：整数

フィールド：ボタン

アクセス：R/W

buttonAlignY プロパティは、ボタン上でアイコンの下に配分するスペースの割合を示します。値は 0 ～ 100 パーセントが有効で、デフォルト値は 50 です。アイコンのサイズ変更で縦横比が保たれない場合（アイコン上下のスペースに違いはなくなるので）、このプロパティは使用されません。

buttonPosition

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

型：整数

フィールド：ボタン

アクセス：R/W

buttonPosition プロパティは、ボタン上のテキストとアイコンの配置方法を示します。*position* オブジェクトでは、便利なキーワードが定義されています。

アイコン／テキストの配置	キーワード
テキストのみ	position.textOnly
アイコンのみ	position.iconOnly
アイコンを上、テキストを下	position.iconTextV
テキストを上、アイコンを下	position.textIconV
アイコンを左、テキストを右	position.iconTextH
テキストを左、アイコンを右	position.textIconH
テキストをアイコンに重ねる	position.overlay

buttonScaleHow

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

型：整数

フィールド：ボタン

アクセス：R/W

buttonScaleHow プロパティは、ボタンに合わせて（必要に応じて）アイコンを拡大／縮小する方法を示します。*scaleHow* オブジェクトでは、便利なキーワードが定義されています。

アイコンの拡大／縮小方法	キーワード
元の縦横比を保つ	scaleHow.proportional
ボタンに合わせる	scaleHow.anamorphic

buttonScaleWhen

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

型：整数

フィールド：ボタン

アクセス：R/W

`buttonScaleWhen` プロパティは、ボタンに合わせて（必要に応じて）アイコンを拡大／縮小する条件を示します。`scaleWhen` オブジェクトでは、便利なキーワードが定義されています。

アイコンの拡大／縮小条件	キーワード
常に調整	<code>scaleWhen.always</code>
調整しない	<code>scaleWhen.never</code>
アイコンが大きすぎる場合	<code>scaleWhen.tooBig</code>
アイコンが小さすぎる場合	<code>scaleWhen.tooSmall</code>

calcOrderIndex

			
--	-----------------------------------------------------------------------------------	--	--

型：整数

フィールド：コンボボックス、テキスト

アクセス：R/W

このプロパティで、ドキュメント内のフィールドの計算順序を変更することができます。計算を行うテキストまたはコンボボックスフィールドをドキュメントに追加すると計算順序の配列にフィールド名が追加され、ドキュメント内でのフィールドの計算順序が決まります。`calcOrderIndex` プロパティは、Acrobat Form ツールの「フィールドの計算」ダイアログの機能に相当します。次に例を示します。

```
var a = this.getField("newItem");  
var b = this.getField("oldItem");  
a.calcOrderIndex = b.calcOrderIndex + 1;
```

この例では、[Doc オブジェクト](#)のメソッド [getField](#) で、「oldItem」フィールドの後に追加された「newItem」フィールドを取得しています。そして、「oldItem」フィールドの `calcOrderIndex` を変更して、「oldItem」フィールドが「newItem」フィールドの前に計算されるようにしています。

charLimit

			
--	-------------------------------------------------------------------------------------	--	--

型：整数

フィールド：テキスト

アクセス：R/W

このプロパティは、テキストフィールドに入力できる文字数を制限します。

currentValueIndices

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

型：整数または配列 フィールド：コンボボックス、リストボックス アクセス：R/W

取得：このプロパティは、リストボックスまたはコンボボックスフィールドの現在値のインデックスを配列にして返します。インデックスは 0 ベースです。フィールドの値が 1 つの場合は整数を返し、複数の場合は整数の配列を昇順にソートして返します。フィールドの現在値がリストに存在するものではない場合（編集可能なコンボボックスで有り得ます）、-1 を返します。

例：複数選択が可能なリストボックスの「選択の変更」に次のコードを割り当てると、現在の選択内容をコンソールに出力します。

```
if (event.willCommit) {
    var f = event.target;
    var a = f.currentValueIndices;
    if (typeof a == "number")
        console.println("Selection: " + f.getItemAt(a, false));
    else {
        console.println("Selection:");
        for (var i = 0; i < a.length; i++)
            console.println("    " + f.getItemAt(a[i], false));
    }
}
```

設定：このプロパティでリストボックスまたはコンボボックスの値を設定することができます。1 つの整数または整数の配列を指定できますが、リストボックスまたはコンボボックスの項目を 1 つだけ選択する場合には、整数を指定します。この整数はリスト項目の配列のインデックス（0 ベース）です。編集可能なコンボボックスで設定したい値がリストに含まれていない場合、Field.value プロパティを使用してください。それ以外の場合は currentValueIndices を使用してリストボックスとコンボボックスの値を設定します。複数選択が可能なリストボックスで（リストボックスでのみ可能なわけですが）複数の項目を選択したい場合、選択する項目インデックスの配列（昇順にソートされている必要があります）をこのプロパティに渡してください。JavaScript でリストボックスの項目を複数選択するには、currentValueIndices プロパティを使用するしかありません。

例：次のコードでは、リストボックスの 2 番目と 4 番目（0 ベースのインデックス値はそれぞれ 1 と 3）の項目を選択します。

```
var f = getField("myList");
f.currentValueIndices = [1,3];
```

リストボックスで複数選択を可能にするには、[multipleSelection](#) プロパティを使用します。

defaultValue

			
--	-----------------------------------------------------------------------------------	--	--

型：文字列

フィールド：ボタンと署名以外のすべて

アクセス：R/W

このプロパティは、フィールドのデフォルト値を示し、フォームがリセットされた場合にはこの値がフィールドに設定されます。コンボボックスおよびリストボックスでは、デフォルト値の設定に書き出し値または項目名のどちらでも使用できます。矛盾が生じた場合（書き出し値と項目名の文字列が同じで、それぞれが別の項目として選択リストに存在する場合など）、書き出し値の値が適用されます。

例：

```
var f = this.getField("Name");  
f.defaultValue = "Enter your name here.";
```

doNotScroll

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

型：ブーリアン

フィールド：テキスト

アクセス：R/W

このプロパティが *true* に設定されているとテキストフィールドはスクロールせず、フィールドを囲む矩形領域に表示が限定されます。このプロパティを *true* または *false* に設定することは、フィールドの「オプション」タブにある「スクロールしない」オプションを操作することと同じです。

doNotSpellCheck

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

型：ブーリアン

フィールド：コンボボックス（編集可能なもの）、テキスト

アクセス：R/W

このプロパティが *true* に設定されている場合、編集可能なテキストフィールドでスペルチェックが実行されません。このプロパティを *true* または *false* に設定することは、フィールドの「オプション」タブにある「スペルチェックしない」オプションを操作することと同じです。

delay

型：ブーリアン

フィールド：すべて

アクセス：R/W

このプロパティは、フィールドの外観の再描画を一時的に停止させることができます。複数のフィールドプロパティに対する一連の変更が終了してから、フィールドの外観を再描画するために使用するのが一般的です。プロパティを *true* に設定すると、再度 *false* にするまでフィールドの変更が強制的に待ち状態になります。*false* に設定するとフィールドが再描画され、最新の設定が反映されます。

```
// 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;
f.strokeWidth = 2;
// force the changes now
f.delay = false;
```

複数のフィールドの変更を同時に反映させたい場合、ドキュメントレベルの [delay](#) プロパティを使用することができます。

display

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

型：整数

フィールド：すべて

アクセス：R/W

このプロパティは、画面および印刷で、フィールドの表示／非表示を制御します。

説明	キーワード
画面および印刷でフィールドを表示する	display.visible
画面および印刷でフィールドを非表示にする	display.hidden
画面ではフィールドを表示するが、印刷はしない	display.noPrint
画面ではフィールドを非表示にするが、印刷はする	display.noView

今後のバージョンでは [hidden](#) および [print](#) に代わってこのプロパティが使用されます。

例：

```
// 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

型：オブジェクト

フィールド：すべて

アクセス：R

このプロパティは、フィールドが属すドキュメントの [Docオブジェクト](#) を返します。詳しくは、「[Docオブジェクト](#)」の節を参照してください。

editable

			
--	-----------------------------------------------------------------------------------	--	--

型 : ブーリアン

フィールド : コンボボックス

アクセス : R/W

コンボボックスを編集可能にすることができます。これはユーザによるキー入力が可能になるということです。このプロパティは、ユーザによるキー入力が可能であるか、あるいは用意された選択肢からのみ選択するのを示します。

```
var f = this.getField("myComboBox");  
f.editable = true;
```

exportValues

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

型 : 配列

フィールド : チェックボックス、ラジオボタン

アクセス : R/W

このプロパティは、フィールドの書き出し値の配列です。ラジオボタンフィールドの場合、フィールドをグループとして動作させる必要があります。つまり、1つのボタンが選択された場合、それをフィールド全体の値として扱う必要があります。チェックボックスフィールドの場合、書き出し値が指定されていなければ、フィールドがチェックされたときのデフォルト値は「はい」（あるいは対応する言語の文字列）になります。チェックを解除したときの値は「Off」になります（ラジオボタンフィールドでどのボタンもチェックされていない場合も同様）。このプロパティの配列には、同じフィールド名を持つフィールドの数だけ、書き出し値の要素が存在します。配列の要素は、グループを構成する個々のフィールドの書き出し値であり、フィールドが作成された順に並んでいます（タブオーダーには影響されません）。

例 :

```
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			
-----	-------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------	--

型 : ブーリアン


フィールド : テキスト

アクセス : R/W

テキストフィールドの `fileSelect` プロパティを `true` に設定すると、フィールドの「オプション」タブの「ファイルの選択に使用する」オプション（簡単に言うとファイル選択フラグ）が設定されます。この場合のフィールド値は、フォームと共に送信するファイルのパス名であることを示します。

ユーザは、フィールドにパス名を直接入力することも、ファイルの参照によって入力することもできます（[browseForFileToSubmit](#) メソッドを参照）。

注意： ファイル選択フラグは、[multiline](#)、[charLimit](#)、[password](#)、および [defaultValue](#) プロパティと相互に排他的です。また、Macintosh プラットフォームでファイル選択フラグを設定すると、フィールドが読み取り専用として処理されるので、パス名をファイルの参照によってフィールドに入力する必要があります（[browseForFileToSubmit](#) メソッドを参照）。

セキュリティ ：このプロパティは、バッチ、メニュー、またはコンソールイベントでのみ使用できます。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

fillColor



型：配列

フィールド：すべて

アクセス：R/W

このプロパティは、フィールドの背景色を示します。フィールドの背景色とは、フィールドを囲む矩形の色のことです。値は透明、グレー、RGB、または CMYK カラーのいずれかを使用して定義します。カラー配列の定義およびこのプロパティでの値の使用方法について詳しくは、[「カラー配列」](#)の節を参照してください。

```
var f = this.getField("myField");
if (color.equal(f.fillColor, color.red))
    f.fillColor = color.blue;
else
    f.fillColor = color.yellow;
```

旧バージョンでは、このプロパティは `bgColor` という名前でした。`bgColor` の使用は現在推奨されていませんが、下位互換のために残されています。

hidden



型：ブーリアン

フィールド：すべて

アクセス：R/W

このプロパティは、ユーザに対してフィールドを表示するか非表示にするかを制御します。値が *false* であればフィールドを表示し、*true* であれば非表示にします。*hidden* のデフォルト値は *false* です。

```
// Set the field to hidden
var f = this.getField("myField");
f.hidden = true;
```

今後のバージョンでこのプロパティに代わって使用される [display](#) プロパティも参照してください。

highlight



型：文字列

フィールド：ボタン

アクセス：R/W

このプロパティは、ユーザがボタンをクリックしたときのボタンの状態を示します。サポートされている 4 つの強調表示モードは、*none*、*invert*、*push*、および *outline* です。

- *none* では、ボタンをクリックしても視覚的な変化はありません。
- *invert* では、ボタンを囲む矩形領域が一時的に反転します。
- *push* では、ボタン上のアイコンまたはテキスト（存在する場合）が一時的に下に押されて表示されます。
- *outline* では、ボタンを囲む境界線が一時的に反転します。

highlight オブジェクトでは、ボタンに有効なすべての特性が定義されているので便利です。次の表に、*highlight* オブジェクトとそれに関連するキーワードを示します。

タイプ	キーワード
none	highlight.n
invert	highlight.i
push	highlight.p
outline	highlight.o

次の例では、ボタンの *highlight* プロパティを「invert」に設定します。

```
// set the highlight mode on button to invert
var f = this.getField("myButton");
f.highlight = highlight.i;
```

lineWidth



型：整数

フィールド：すべて

アクセス：R/W

このプロパティは、フィールドを囲む矩形の境界線の太さを示します。線の色が透明の場合、このプロパティによる影響はありません（境界線のスタイルが *beveled* の場合を除く）。JavaScript で *lineWidth* プロパティを設定するには、次の整数値を使用します。

線幅	キーの値
なし	0
細い	1
標準	2
太い	3

例：

```
// Change the border width of the Text Box to medium thickness
f.lineWidth = 2
```

lineWidth のデフォルト値は 1（細い）です。整数値はいくつを指定しても構いませんが、値が 5 を超えるとフィールドの外観が歪んでしまう可能性があります。

旧バージョンでは、このプロパティは *borderWidth* という名前でした。*borderWidth* の使用は現在推奨されていませんが、下位互換のために残されています。

multiline



型：ブーリアン フィールド：テキスト アクセス：R/W

このプロパティは、フィールドにおけるテキストの折り返し方法を示します。*multiline* が *false*（デフォルト）に設定されていると、テキストフィールドには 1 行しか含めることができませんが、*true* に設定されている場合は複数行が有効となり、フィールドの右端では行が折り返されます。

multipleSelection



型：ブーリアン フィールド：リストボックス アクセス：R/W

このプロパティが *true* の場合、リストボックスフィールドで複数項目の選択が可能であることを示します。

Event.[type](#)、Field.[value](#)、および [currentValueIndices](#) も参照してください。

name

型：文字列

フィールド：すべて

アクセス：R

このプロパティでは、フィールドの完全修飾名に文字列オブジェクトとしてアクセスできます。

```
var f = this.getField("myField");
console.println(f.name);           // displays "myField" in console window
```

numItems

型：整数

フィールド：コンボボックス、リストボックス

アクセス：R

コンボボックスまたはリストボックスに含まれる項目数を返します。

page

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

型：整数または配列

フィールド：すべて

アクセス：R

フィールドのページ番号またはその配列を返します。ドキュメントに存在する同じ名前のフィールドの外観が1つだけの場合、そのフィールドが存在するページの番号（0 ベース）が整数で返されます。同じ名前のフィールドの外観が複数存在する場合、それら各フィールドが存在するページの番号を要素とした整数の配列（0 ベース）が返されます。配列における番号の順序は、フィールドの各ウィジェットを作成した順になります（タブオーダーには影響されません）。

例：

```
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");
```

password

			
--	-------------------------------------------------------------------------------------	--	--

型：ブーリアン

フィールド：テキスト

アクセス：R/W

このプロパティを true にすると、フィールドに入力されたデータかわりにアスタリスクが表示されます。データの送信を行った場合は、実際に入力したデータが送信されます。パスワード属性が設定されたフィールドのデータは、ドキュメントをディスクに保存する際に失われてしまいます。

print



型：ブーリアン

フィールド：すべて

アクセス：R/W

このプロパティは、あるフィールドを印刷するかどうかを示します。ドキュメントを印刷する際にフィールドも印刷するには、*print* プロパティを *true* に設定し、フィールドが印刷されないようにするには *false* に設定します。このプロパティは、印刷されたページには不要な制御ボタンやフィールドを隠すために使用できます。

```
var f = this.getField("myField");  
f.print = false;
```

このプロパティの使用は推奨されません。代わりに [display](#) プロパティを使用してください。

readonly



型：ブーリアン

フィールド：すべて

アクセス：R/W

このプロパティは、フィールドの読み取り専用属性を設定または取得します。フィールドが読み取り専用になると、ユーザにフィールドは表示されますが、変更することはできなくなります。

rect



型：配列

フィールド：すべて

アクセス：R/W

このプロパティは、フォームフィールドのサイズと位置を示し、[回転ユーザスペース](#)の4つの数値の配列を引数に取る（設定の場合）か、返します（取得の場合）。座標の順序は、左上の *x*、左上の *y*、右下の *x*、右下の *y* となります。

例：

```
// 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;
```


例：

```
// 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
```

注意： [Annot オブジェクト](#)にも [rect](#) プロパティがありますが、1) 座標は[回転ユーザスペース](#)ではなく、2) [Field.rect](#) の配列要素の順序とも異なります。

required



型：ブーリアン

フィールド：ボタン以外のすべて

アクセス：R/W

このプロパティは、フィールドの必須属性を設定、取得します。必須属性が `true` の場合、ユーザが送信ボタンをクリックしてフィールド値を送信するには、フィールドが非ヌル値でなければいけません。フィールド値が `null` の場合、ユーザに警告メッセージが表示され、送信は行われません。

例：

```
var f = this.getField("myField");
f.required = true;
```

strokeColor



型：配列

フィールド：すべて

アクセス：R/W

このプロパティは、フィールドを囲む矩形を線幅で指定された太さの線で描くときの色を指定します。透明、グレー、RGB、または CMYK カラーのいずれかを使用して、値を定義します。カラー配列の定義およびこのプロパティでの値の使用方法については、「[カラー配列](#)」の節を参照してください。

旧バージョンでは、このプロパティは `borderColor` という名前でした。`borderColor` の使用は現在推奨されていませんが、下位互換のために残されています。

style

			
--	-----------------------------------------------------------------------------------	--	--

型：文字列 フィールド：チェックボックス、ラジオボタン アクセス：R/W

このプロパティでは、チェックボックスまたはラジオボタンのスタイルを設定できます。つまり、選択されたチェックボックスまたはラジオボタンを示すマークを設定できます。有効なスタイルには、「check」（チェックマーク）、「cross」（十字形）、「diamond」（ひし形）、「circle」（円形）、「star」（星型）、および「square」（四角形）があります。次の表に、style プロパティとそれに関連するキーワードを示します。

スタイル	キーワード
check	style.ch
cross	style.cr
diamond	style.di
circle	style.ci
star	style.st
square	style.sq

次の例に、このプロパティとスタイルオブジェクトの使用法を示します。

```
var f = this.getField("myCheckbox");  
f.style = style.ci;
```

submitName

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

型：文字列 フィールド：すべて アクセス：R/W

このプロパティが空でない場合、フォーム送信でフィールドの [name](#) の代わりに使用されます。HTML 形式（URL エンコードなど）で送信する場合にのみ有効です。

textColor

			
--	-------------------------------------------------------------------------------------	--	--

型：配列 フィールド：すべて アクセス：R/W

このプロパティは、フィールドの前景色を示します。テキスト、ボタン、またはリストボックスフィールドのテキストの色、およびチェックボックスまたはラジオボタンフィールドのチェックの色を表します。値の定義の仕方は、[fillColor](#) プロパティと同じです。カラー配列の定義およびこのプロパティでの値の設定、使用方法について詳しくは、[カラー配列](#)の節を参照してください。

```
var f = this.getField("myField");  
f.textColor = color.red;
```

旧バージョンでは、このプロパティは *fgColor* という名前でした。*fgColor* の使用は現在推奨されていませんが、下位互換のために残されています。

注意： 透明カラースペースを使用して *textColor* を設定すると、例外が発生します。

textFont



型：文字列 フィールド：ボタン、コンボボックス、リストボックス、テキスト
アクセス：R/W

textFont プロパティは、テキストフィールド、コンボボックス、リストボックス、またはボタンに配置するテキストのフォントを示します。有効なフォントは、「フォント」オブジェクトのプロパティとして次のように定義されています。

テキストのフォント	キーワード
Times-Roman	font.Times
Times-Bold	font.TimesB
Times-Italic	font.TimesI
Times-BoldItalic	font.TimesBI
Helvetica	font.Helv
Helvetica-Bold	font.HelvB
Helvetica-Oblique	font.HelvI
Helvetica-BoldOblique	font.HelvBI
Courier	font.Cour
Courier-Bold	font.CourB
Courier-Oblique	font.CourI
Courier-BoldOblique	font.CourBI
Symbol	font.Symbol
ZapfDingbats	font.ZapfD

次の例に、このプロパティの使用法とフォントオブジェクトを示します。

```
// set the font of "myField" to Helvetica
var f = this.getField("myField");
f.textFont = font.Helv;
```

5.0 追記事項

`textFont` にフォントの PostScript 名を設定すると、任意のフォントを使用してテキストフィールド、コンボボックス、リストボックス、またはボタンにテキストを配置できます。

例：

```
// set the font of "myField" to Viva-Regular
var f = this.getField("myField");
f.textFont = "Viva-Regular";
```

注意： フォントオブジェクトで定義されている以外の任意のフォントを使用すると、旧バージョンのビューアで互換性に関する問題が生じます。

textSize

			
--	-----------------------------------------------------------------------------------	--	--

型：整数

フィールド：すべて

アクセス：R/W

このプロパティは、すべてのコントロールに使用されるテキストのサイズ（ポイント単位）を示します。チェックボックスおよびラジオボタンフィールドでは、テキストサイズによってチェックのサイズが決まります。有効なサイズは 0 と 4 ～ 144 の範囲です。テキストサイズ 0 は、すべてのテキストデータがフィールドの矩形に収まる最大のポイントサイズを使用すること（自動調整）を意味します。

```
// set the text size of myField to 28 point
this.getField("myField").textSize = 28;
```

type

型：文字列

フィールド：すべて

アクセス：R

これは読み取り専用のプロパティで、フィールドのタイプを文字列として返します。返される有効なタイプには、「button」（ボタン）、「checkbox」（チェックボックス）、「combobox」（コンボボックス）、「listbox」（リストボックス）、「radiobutton」（ラジオボタン）、「signature」（署名）、および「text」（テキスト）があります。

userName

			
--	-------------------------------------------------------------------------------------	--	--

型：文字列

フィールド：すべて

アクセス：R/W

このプロパティでは、フィールドの説明を文字列として返したり設定したりすることができます。フィールドの説明は、マウスカーソルをフィールド内に置いたときにツールヒントのテキストとして使用されます。また、エラーメッセージを生成する際、フィールドの名前（ユーザに表示するには分かりにくい名前であることが多い）に代わる分かりやすい名前として使用することもできます。

value



型：各種

フィールド：ボタン以外のすべて

アクセス：R/W

このプロパティは、ユーザが入力したフィールドデータの値を取得します。`value` は、フィールドのタイプに応じて、文字列、日付、または数値のいずれかになります。`value` は、計算フィールドの作成に使用するのが一般的です。

```
var oil = this.getField("Oil");
var filter = this.getField("Filter");
event.value = (oil.value + filter.value) * 1.0825;
```

この例では、「Oil」フィールドと「Filter」フィールドの値を合計し、それに州の消費税を加えた値を計算フィールドに設定しています。`value` は、フィールドプロパティの中でも最も重要なプロパティであるといえます。

注意： 署名フィールドでは、フィールドが署名済の場合には非ヌル文字列が値として返されます。

5.0 追記事項

フィールドが複数選択可能なリストボックス（[multipleSelection](#) を参照）である場合、配列を渡してフィールドの値を設定することもできます。同様に、複数の値が現在選択されているリストボックスでは、`value` に配列が返されます。

[Event オブジェクト](#) の [type](#) に関する注意も参照してください。関連プロパティとしては、[valueAsString](#) があります。

注意： ただし、複数選択可能なリストボックスの値の取得および設定には、[currentValueIndices](#) プロパティの方が適しています（そして最も効果的です）。

valueAsString

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

型：各種

フィールド：ボタン以外のすべて

アクセス：R

[value](#) プロパティは、フィールドの内容を「適切なフォーマット」に変換しようと試みます。例えば、「020」というフィールド値は整数型に変換され、頭のゼロが除かれて「20」が返されます。一方、`valueAsString` はフィールドの値を JavaScript 文字列として返すので、この例の場合、「020」という値が返されます。

Field オブジェクトのメソッド

browseForFileToSubmit

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

パラメータ：なし

戻り値：なし

このメソッドをファイル選択フラグが設定された（チェックされた）テキストフィールドで実行すると、標準のファイル選択ダイアログが表示されます。ダイアログで入力したパスは、テキストフィールドの値として自動的に割り当てられます。

例：次のコードでは、ファイル選択フラグが設定されたテキストフィールドを参照しているものとします。これをボタンフィールドの「マウスボタンを放す」アクションに割り当てます。

```
var f = this.getField("resumeField");  
f.browseForFileToSubmit();
```

ファイル選択フラグの設定には、フィールドプロパティである [fileSelect](#) を使用することができます。

注意： [fileSelect](#) フラグがクリアされた（チェックされていない）テキストフィールドでこのメソッドを実行すると、例外が発生します。

buttonGetCaption

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

パラメータ：`[nFace]`

戻り値：文字列

このメソッドは、ボタンに関連付けられているキャプションを返します。オプションの `nFace` パラメータを指定して、ボタンが通常の状態のキャプション（0）、ボタンを押下した状態のキャプション（1）、またはボタン上にマウスポインタを置いた状態（ロールオーバー）のキャプション（2）を取得することができます。

例：この例では、アイコンとテキストを表示しているボタンを想定し、キャプションの左右に矢印を表示します。

```
// 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);
```

ロールオーバーに同じアイコンと、テキストに矢印を挿入したキャプションを設定しても、同じ効果が得られます。ただし、この方法は処理速度が遅く、アイコンが明滅する原因となります。上記のコードではキャプションのみを変更してアイコンは変更しないので、速やかに表示を切り替えられます。

buttonGetIcon

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

パラメータ：[nFace]
戻り値：アイコンオブジェクト

このメソッドは、ボタンに関連付けられている[アイコンオブジェクト](#)を返します。オプションの *nFace* パラメータを指定して、ボタンが通常の状態のアイコン（0）、ボタンを押下した状態のアイコン（1）、またはボタン上にマウスポインタを置いた状態（ロールオーバー）のアイコン（2）を取得することができます。*nFace* が指定されていない場合は、0 であるとみなされます。

例：

```
// 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);
```

他の使用例については、[buttonSetIcon](#) および [buttonImportIcon](#) も参照してください。

buttonImportIcon

パラメータ：[cPath]、[nPage]
戻り値：整数

このメソッドは、ボタンの外観を別の PDF ファイルからインポートします。

5.0 追記事項

`buttonImportIcon` メソッドには、2 つのオプションパラメータがあります。どちらのパラメータも指定しない場合、PDF ファイルを選択するために、システムで利用可能なダイアログが表示されます。

`cPath` は、デバイスに依存しないファイルのパス名です。デバイスに依存しないパス名の形式については、『PDF Reference』の第 3.10.1 節を参照してください。

`nPage` には、アイコンに変換する対象ファイルのページ番号（0 ベース）を指定します。デフォルトは 0 です。

このメソッドは、整数を返します。

リターンコード	
コード	説明
1	ユーザがダイアログをキャンセルした。
0	エラーなし。
-1	選択したファイルを開けなかった。
-2	選択したページが無効。

例：社員情報データベースに接続中であると仮定します。データベースとのやり取りには、[ADBC オブジェクト](#)および関連オブジェクトを使用しています。社員のレコードを取得し、`FirstName`、`SecondName`、および `Picture` という 3 つの列を使用します。データベースの `Picture` 列には、PDF 形式で保存された社員の写真を指す、デバイスに依存しないパス名が入っています。この場合、以下のようなスクリプトが考えられます。

```
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");
```

ボタンフィールドである「`myPicture`」は、アイコンとキャプションの両方を表示するように設定されているとします。コードでは社員の姓と名を連結し、写真のキャプションを形成しています。アイコンの取得に失敗した場合には代替りのアイコンをインポートしています。

buttonSetCaption

5.0



パラメータ：`cCaption`、`[nFace]`
戻り値：なし

このメソッドは、ボタンにキャプション（テキスト）`cCaption` を関連付けます。オプションの `nFace` パラメータを指定して、ボタンが通常の状態のキャプション（0）、ボタンを押下した状態のキャプション（1）、またはボタン上にマウスポインタを置いた状態（ロールオーバー）のキャプション（2）を設定することができます。`nFace` が指定されていない場合は、0 であるとみなされます。アイコンとキャプションを実際にボタンに配置する方法については、ボタン／キャプションの配置に関連するプロパティを参照してください。

例：

```
var f = this.getField("myButton");
f.buttonSetCaption("Hello");
```

詳しい例については、関連メソッド [buttonGetCaption](#) を参照してください。

buttonSetIcon

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

パラメータ：*olcon*、*[nFace]*

戻り値：なし

このメソッドは、[アイコンオブジェクト](#) *olcon* をボタンに関連付けます。オプションの *nFace* パラメータを指定して、ボタンが通常の状態のキャプション (0)、ボタンを押下した状態のキャプション (1)、またはボタン上にマウスポインタを置いた状態 (ロールオーバー) のキャプション (2) を設定することができます。*nFace* が指定されていない場合は、0 であるとみなされます。アイコンを実際にボタンにレンダリングする方法について詳しくは、ボタンの配置に関連するプロパティを参照してください。

例：この例では、ドキュメントに含まれるすべての名前付きアイコンから、アイコン名のリストボックスを作成しています。リストボックスで項目を選択すると、その名前のアイコンがフィールド「myPictures」のボタンに表示されます。以下のコードをボタンフィールド「myButton」の「マウスボタンを放す」アクションに割り当ててください。

```
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'
+ '});
```

[importIcon](#) の例に示される対話的な方法やバッチシーケンスによって、名前付きアイコン自体をドキュメントにインポートすることもできます。

[buttonGetIcon](#) も参照してください。

checkThisBox

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

パラメータ : *nWidget*、*[bCheckIt]*
戻り値 : なし

このメソッドの最初のパラメータである *nWidget* は、ラジオボタンまたはチェックボックスのグループで個々のウィジェットを示すインデックス (0 ベース) です。インデックスの順番は、個々のウィジェットが作成された順になります (タブオーダーには影響されません)。2 番目のオプションパラメータはブーリアンで、対象となるウィジェットにチェックを入れるかどうかを示します。デフォルトは *true* です。チェックボックスはチェックを解除することもできますが、ラジオボタンはチェックを解除できません。しかし、どうしてもラジオボタンのチェックを解除する必要がある場合には、(例えば、ドキュメントレベルの [resetForm](#) メソッドを使用して) ラジオボタンをリセットすることができます。ただしこの方法は、ラジオボタンのデフォルトが「チェックしない」になっている場合のみ有効です ([defaultIsChecked](#) メソッドを参照)。

例 :

```
// check the box "ChkBox"
var f = this.getField("ChkBox");
f.checkThisBox(0,true);
```

注意 : ラジオボタンのグループ内で重複する書き出し値が存在しないのであれば、選択したいラジオボタンの書き出し値を [value](#) プロパティに設定して、チェックを入れることもできます (空の文字列を渡すとチェックを解除することができます)。

clearItems

			
--	-------------------------------------------------------------------------------------	--	--

パラメータ : なし
戻り値 : なし

このメソッドは、リストボックスまたはコンボボックスの値をすべてクリアします。

```
// Clear the field myList
var f = this.getField("myList");
f.clearItems();
```

defaultIsChecked

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

パラメータ : *nWidget*、*[blsDefaultChecked]*
戻り値 : ブーリアン

このメソッドの最初のパラメータである *nWidget* は、ラジオボタンまたはチェックボックスのグループで個々のウィジェットを示すインデックス (0 ベース) です。インデックスの順番は、個々のウィジェットが作成された順になります (タブオーダーには影響されません)。2 番目のオプションパラメータはブーリアンで、(フィールドがリセットされたときなどに) 対象となるウィジェットをデフォルトでチェックするかどうかを示します。デフォルトは *true* です。

例 :

```
// change the default of "ChkBox" to checked
var f = this.getField("ChkBox");
f.defaultIsChecked(0,true);
this.resetForm(["ChkBox"]);
```

注意 : ラジオボタンのグループ内で重複する書き出し値が存在しないのであれば、デフォルトでチェックしたいラジオボタンの書き出し値を [defaultValue](#) プロパティに設定することもできます (空の文字列を渡すとデフォルトでチェックされるボタンはなくなります)。

deleteItemAt

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

パラメータ : *[nIdx]*
戻り値 : なし

このメソッドは、コンボボックスまたはリストボックスの項目を削除します。パラメータ *nIdx* は、リストから削除する項目のインデックス (0 ベース) です。 *nIdx* が指定されていない場合、現在選択されている項目が削除されます。

リストボックスの場合、現在選択されている項目を削除すると、どの項目も選択されていない状態になります。この状態で同じフィールドに対してパラメータなしでメソッドを再度実行すると、削除対象の項目が存在しないため動作が不安定になることがあります。したがって、このメソッドを正しく動作させるためには、あらかじめ項目を選択しておく必要があります。項目を選択するには、フィールドプロパティの [currentValueIndices](#) を使用します。

例 :

```
var a = this.getField("MyListBox");
a.deleteItemAt(); // delete current item, and...
a.currentValueIndices = 0; // select top item in list
```

getArray

パラメータ : なし
戻り値 : フィールドの配列

このメソッドは、指定した親フィールドに属する子フィールド（値を持つことのできるフィールドなど）の配列を返します。このメソッドは、テーブルですべての子フィールドの値を合計したい場合など特に便利です。

```
// 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

パラメータ : *nIdx*、[*bExportValue*]

戻り値 : リストボックスまたはコンボボックスの書き出し値または項目名

このメソッドは、コンボボックスまたはリストボックスの項目の内部値を取得します。パラメータ *nIdx* は、リストで取得する項目のインデックスです。*nIdx* が -1 に設定されていると、リストの最後の項目の値が返されます。

5.0 追記事項

2 番目のオプションパラメータ *bExportValue* の値によって、返される値の種類が決まります。*bExportValue* が *true*（デフォルト）に設定されており、目的の項目に書き出し値がある場合は、書き出し値が返され、それ以外の場合は項目名が返されます。*bExportValue* が *false* に設定されていると、常に項目名が返されます。

例：次の 2 つの例では、リストボックス「myList」に 3 つの項目が含まれていることを前提にしています。「First」の書き出し値は 1、「Second」の書き出し値は 2、「Third」には書き出し値がないとします。

```
// returns value of first item in list, which is 1
var f = this.getField("myList");
var v = f.getItemAt(0);
```

次の例では、2 番目のオプションパラメータの使用法を示しています。

```
for (var i=0; i < f.numItems; i++)
    console.println(f.getItemAt(i,true) + ": " + f.getItemAt(i,false));
```

コンソールの出力は、次のようになります。

```
1:      First
2:      Second
Third:   Third
```

このように、2 番目のパラメータを *false* に設定すると、書き出し値が存在する場合でも項目名（表示値）を取得することができます。

insertItemAt

			
--	-----------------------------------------------------------------------------------	--	--

パラメータ : *cName*、*[cExport]*、*[nIdx]*

戻り値 : なし

このメソッドは、コンボボックスまたはリストボックスに新しい項目を挿入します。

cName は、項目名、つまりフィールドに表示される名前です。このメソッドは、リストボックスまたはコンボボックスでのみ動作します。

cExport は、この項目が選択されたときのフィールドの書き出し値です。書き出し値が存在しない場合、*cName* が書き出し値として使用されます。

nIdx は、リストで項目の挿入箇所を示すインデックスです。*nIdx* が 0 の場合、*cName* がリストの先頭に挿入されます。*nIdx* が -1 の場合は、*cName* がリストの最後に挿入されます。*nIdx* のデフォルト値は 0 です。

```
var l = this.getField("myList");  
l.insertItemAt("sam", "s", 0); /* inserts sam to top of list l */
```

isBoxChecked

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

パラメータ : *nWidget*

戻り値 : ブーリアン

このメソッドの最初のパラメータである *nWidget* は、ラジオボタンまたはチェックボックスのグループで個々のウィジェットを示すインデックス（0 ベース）です。インデックスの順番は、個々のウィジェットが作成された順になります（タブオーダーには影響されません）。このメソッドは、対象となるウィジェットが現在チェックされているかどうかを示すブーリアンを返します。

例 :

```
var f = this.getField("ChkBox");  
if(f.isBoxChecked(0))  
    app.alert("The Box is Checked");  
else  
    app.alert("The Box is not Checked");
```

注意： ラジオボタンのグループ内で重複する書き出し値が存在しないのであれば、[value](#) プロパティを取得して、現在チェックされているウィジェットの書き出し値を取得することもできます（チェックされているウィジェットが存在しない場合、文字列「Off」が返されます）。

isDefaultChecked

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

パラメータ : *nWidget*
戻り値 : ブーリアン

このメソッドの最初のパラメータである *nWidget* は、ラジオボタンまたはチェックボックスのグループで個々のウィジェットを示すインデックス（0 ベース）です。インデックスの順番は、個々のウィジェットが作成された順になります（タブオーダーには影響されません）。このメソッドは、フィールドがリセットされたときなどに、ウィジェットがデフォルトでチェックされるようになっているかを示すブーリアンを返します。

例：

```
var f = this.getField("ChkBox");
if (f.isDefaultChecked(0))
    app.alert("The Default: Checked");
else
    app.alert("The Default: Unchecked");
```

注意： ラジオボタンのグループ内で重複する書き出し値が存在しないのであれば、[defaultValue](#) プロパティを取得して、デフォルトでチェックされるウィジェットの書き出し値を取得することもできます（デフォルトでチェックされるウィジェットが存在しない場合、文字列「Off」が返されます）。

setAction

			
--	-------------------------------------------------------------------------------------	--	--

パラメータ : *cTrigger*、*cScript*
戻り値 : なし

このメソッドは、イベントに応じたフィールドのアクションを設定します。

cTrigger : アクションのイベントを示す文字列です。このパラメータの有効な値には、「MouseUp」、「MouseDown」、「MouseEnter」、「MouseExit」、「OnFocus」、「OnBlur」、「Keystroke」、「Validate」、「Calculate」、および「Format」があります。

cScript : イベントが発生したときに実行する JavaScript コードを指定します。

例 :

```
var f = this.addField("actionField", "button", 0 , [20, 100, 100, 20]);
f.setAction("MouseUp", "app.beep(0);");
f.fillColor = color.ltGray;
f.buttonSetCaption("Beep");
f.borderStyle = border.b;
f.lineWidth = 3;
f.strokeColor = color.red;
f.highlight = highlight.p;
```

[buttonSetIcon](#) にも、このメソッドの別の例を示しています。

setFocus

4.05			
------	--	--	--

パラメータ : なし
戻り値 : なし

このメソッドは、フィールドにキーボードフォーカスを設定します。フォーカスを設定したフィールドが表示画面の外にある場合、そのフィールドを表示するために、表示ページの変更やスクロールが行われます。該当フィールドのドキュメントがビューアで手前に表示されていない場合、自動的に手前に表示されます。

例 :

```
// Search for a certain open doc, then focus in on the field of interest.
// This will only work on documents with disclosed set to true
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:"
        d[i].getField("name").setFocus(); // also brings the doc to front.
        break;
    }
}
```

[bringToFront](#) メソッドも参照してください。

setItems

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

パラメータ : oArray
戻り値 : なし

このメソッドは、コンボボックスまたはリストボックスの項目リストを設定します。唯一のパラメータである *oArray* は配列で、このメソッドの呼び出しに必要です。*oArray* の各要素は、文字列または配列に変換可能なオブジェクトである必要があります。要素が文字列に変換される場合、その文字列が項目名と書き出し値に使用されます。要素が配列の場合、その配列は 2 つのサブ要素で構成されている必要があり、最初のサブ要素は項目名の文字列に変換され、2 つ目の要素は書き出し値として使用されます。

例：

```
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]]);
```

[clearItems](#)、[getItemAt](#)、および [insertItemAt](#) フィールドメソッドも参照してください。

signatureInfo

5.0			⊗
-----	--	--	---

パラメータ：*[oSig]*
戻り値：オブジェクト アクセス：*R*

署名のプロパティを列挙する [signatureInfo オブジェクト](#) を返します。署名ハンドラは、そのハンドラ固有のプロパティを追加指定することもできます。この種の汎用オブジェクトは、署名時にも使用されます。

デフォルトでは、このメソッドは、署名を作成したハンドラを使用して、*signatureInfo* オブジェクトを取得します。オプションの署名ハンドラオブジェクト *oSig* パラメータを指定した場合、署名の作成に使用されたハンドラのエンジンが *oSig* でなくても、*signatureInfo* オブジェクトの取得に *oSig* が使用されます。

すべての署名ハンドラでは、次のプロパティが定義されています。

signatureInfo オブジェクト			
プロパティ	型	アクセス	説明
date	日付	R	PDF 日付形式で表した署名の日付
handlerName	文字列	R	署名ハンドラの言語に依存しない名前

handlerUserName	文字列	R	署名ハンドラの言語に依存した名前
location	文字列	R/W	署名地
name	文字列	R	署名者
numFieldsAltered	数値	R	前回の署名と「今回」の署名の間に変更されたフィールド数
numFieldsFilledIn	数値	R	前回の署名と「今回」の署名の間に記入されたフィールド数
numPagesAltered	数値	R	前回の署名と「今回」の署名の間に変更されたページ数
numRevisions	数値	R	ドキュメントの変更回数
reason	文字列	R/W	ユーザ指定の署名理由
revision	数値	R	署名に対応するバージョン
status	数値	R	プロセッサに負担のかかる signatureValidate メソッドを最後に実行したときの状態。下の status プロパティのリターンコード の表を参照してください。
statusText	文字列	R	プロセッサに負担のかかる signatureValidate メソッドを最後に実行したときの状態を示す、ユーザへの表示に適した、言語に依存したテキスト文字列

書き込み可能なプロパティを、オブジェクトの署名時に指定することができます。
[signatureSign](#) メソッドを参照してください。

次の表に、Field.signatureInfo.status から返されるコードおよびその意味を示します。

status プロパティのリターンコード	
状態コード	意味
-1	署名フィールドではありません。
0	署名がされていません。
1	不明の状態です。
2	署名が無効です。
3	ドキュメントの署名は有効ですが、署名者の ID を検証できません。
4	ドキュメントの署名および署名者の ID が有効です。

例：

```
var f = this.getField("mySignature"); // get signature field

var sigInfo = f.signatureInfo();
```

```
// returns "Acrobat Self-Sign Security"
console.println(sigInfo.handlerName);

var msg = "Status = " + sigInfo.status +
    " (" + sigInfo.statusText + ")";
console.println(msg);

console.println(sigInfo.name);
console.println(sigInfo.date);
console.println(sigInfo.location);
console.println(sigInfo.reason);
```

注意： certificatesなどの署名ハンドラの一部のプロパティからは、署名が検証されるまでヌル値が返されることがあります。したがって、[signatureValidate](#) の後で *signatureInfo* を再度呼び出す必要があります (certificates は、[PPKLite 署名ハンドラオブジェクト](#)のプロパティです)。

signatureSign



パラメータ : *oSig*、*[oInfo]*、*[cDIPPath]*
 戻り値 : *bSuccess*

指定の署名ハンドラでフィールドに署名します。

oSig は、署名に使用する署名ハンドラオブジェクトです。[Security オブジェクト](#)の [handlers](#) プロパティおよび [getHandler](#) メソッドを参照してください。一部の署名ハンドラでは、署名前にユーザのログインが要求されます。

oInfo は、署名の書き込み可能なプロパティを指定する汎用オブジェクトです。[signatureInfo](#) メソッドおよび [PPKLite 署名ハンドラオブジェクト](#) も参照してください。

cDIPPath には、署名後のファイルの保存先パスを、デバイスに依存しない形式で指定します。*cDIPPath* を指定しない場合、ファイルは元の場所に保存されます。

署名が正常に行われると、このメソッドは *true* を返し、そうでない場合は *false* を返します。署名フィールドは、署名前にクリアする必要があります。[Doc オブジェクト](#)の [resetForm](#) メソッドを参照してください。


次の例では、PPKLite 署名ハンドラで、署名フィールドに署名をします。

```
var ppklite = security.getHandler("Adobe.PPKLite");    // choose handler
ppklite.login("dps017", "/C/signatures/DPSmith.apf"); // login
var f = this.getField("mySignature");                 // get signature field
this.resetForm(["mySignature"]);                      // clear it, and ...
```

```
f.signatureSign(ppk-lite, // sign it
{ password: "dps017", // provide password
  location: "San Jose, CA", // ... see note below
  reason: "I am approving this document",
  contactInfo: "dpsmith@adobe.com",
  appearance: "Fancy"});
```

[PPKLite 署名ハンドラオブジェクト](#)の [getHandler](#) および [login](#) メソッドを参照してください。

注意： 上の例では、パスワードを指定しています。これは、パスワードのタイムアウト期限が切れているかどうかによって、指定する必要があるかどうかが決まります。パスワードのタイムアウトはユーザインタフェース（ツール／Self-Sign セキュリティ／ユーザ設定）で設定するか、あるいは [setPasswordTimeout](#) メソッドを使用してプログラムから設定できます。

セキュリティ ：このメソッドは、バッチ、コンソール、またはメニュー、アプリケーション初期化イベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

signatureValidate

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

パラメータ：[oSig]

戻り値：nStatus

署名の妥当性の状態を返します。署名に使用された署名ハンドラによっては、このメソッドはコンピュータにける負荷が大きくなり、処理時間が大幅にかかることがあります。戻り値は、[signatureInfo.status](#) に示されます。[signatureInfo.statusText](#) プロパティも参照してください。

デフォルトでは、このメソッドは署名の作成に使用されたハンドラを使用します。オプションとして、署名ハンドラオブジェクトである oSig パラメータを指定できます。この場合、署名の作成に使用されたハンドラのエンジンが oSig でなくても、署名の検証に oSig が使用されます。

署名の作成に使用された元のハンドラがインストールされていない場合などには、oSig を使用して署名を検証することができます。

oSig が署名の作成に使用されたハンドラと同じ場合、異なるログオンコンテキストによって検証を行うことができます。

例：

```
var f = this.getField("mySignature") // get signature field
var sigInfo = f.signatureInfo();
if (f.signatureValidate())
    if (sigInfo.status < 3)
        var msg = "Signature not valid!  " + sigInfo.statusText;
    else
        var msg = "Signature valid!  " + sigInfo.statusText;
else
    var msg = "Validation failed!";
app.alert(msg);
```

FullScreen オブジェクト

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

FullScreen オブジェクトは、全画面表示（プレゼンテーションモード）の設定およびプロパティのインタフェースです。FullScreen オブジェクトを取得するには、[App オブジェクト](#)の `fs` プロパティを使用します。

FullScreen オブジェクトのプロパティ

backgroundColor

型：カラー配列

アクセス：R/W

このプロパティは、全画面表示モードにおける画面の背景色を示します。詳しくは、「[カラー配列](#)」を参照してください。

例

```
app.fs.backgroundColor = color.ltGray;
```

clickAdvances

型：ブーリアン

アクセス：R/W

このプロパティは、ページのどこかをマウスでクリックすることによって、次のページに切り替えるかどうかを示します。

cursor

型：数値

アクセス：R/W

このプロパティは、全画面表示モードにおけるマウスポインタの動作を示します。`cursor` オブジェクトでは、有効なカーソルの動作がすべて定義されているので便利です。

カーソルの動作	キーワード
常に非表示	<code>cursor.hidden</code>
数秒間だけ表示	<code>cursor.delay</code>
常に表示	<code>cursor.visible</code>

例：

```
app.fs.cursor = cursor.visible;
```

defaultTransition

型：数値

アクセス：R/W

このプロパティは、全画面表示モードでページを切り替える際に使用するデフォルトの効果を示します。効果名の有効なリストについては、[transitions](#) プロパティを参照してください。

例：ドキュメントを全画面表示モードにします。

```
app.fs.defaultTransition = "WipeDown";  
app.fs.isFullScreen = true;
```

注意： 効果なしに設定するには、`app.fs.defaultTransition = ""`; とします。

escapeExits

型：ブーリアン

アクセス：R/W

このプロパティは、全画面表示モードを Esc キーで取り消しするかどうかを示します。

isFullScreen

型：ブーリアン

アクセス：R/W

このプロパティは、Acrobat ビューアの通常の表示モードと全画面表示モードを切り替えます。

例：

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

上の例では `isFullScreen` を `true` に設定し、Acrobat ビューアを全画面表示モードにしています。`isFullScreen` を `false` にすると、通常の表示モードになります。この場合の通常の表示モードとは、Acrobat アプリケーションが全画面表示モードになる前の状態を指します。

全画面表示モードは、Acrobat ビューアウィンドウでドキュメントを開いている場合のみ動作します。

例については、[defaultTransition](#) を参照してください。

注意： Web ブラウザ内で表示している PDF ドキュメントは、全画面表示モードにすることはできません。

loop

型：ブーリアン

アクセス：R/W

このプロパティは、全画面表示モードで（マウスクリック、キーボード、タイマー設定によって）次のページを表示する際、ドキュメントの最後のページに達したら最初のページへ戻るかどうかを示します。

timeDelay

型：数値

アクセス：R/W

このプロパティは、全画面表示モードで自動的に次のページを表示するまでのデフォルトの秒数を示します。ページの自動的な切り替えを有効／無効にするには、[useTimer](#) を参照してください。

例：

```
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

型：配列

アクセス：R

このプロパティは、ビューアに実装されている有効な効果名を、文字列の配列で返します。次のスクリプトを実行します。

```
console.println "[" + app.fs.transitions + "]" ;
```

結果は次のようになります。

```
[Replace,WipeRight,WipeLeft,WipeDown,WipeUp,SplitHorizontalIn,
SplitHorizontalOut,SplitVerticalIn,SplitVerticalOut,BlindsHorizontal,
BlindsVertical,BoxIn,BoxOut,GlitterRight,GlitterDown,GlitterRightDown,
Dissolve,Random]
```

注意： 効果なしに設定するには、`app.fs.defaultTransition = ""` とします。

[defaultTransition](#) も参照してください。

usePageTiming

型：ブーリアン

アクセス：R/W

このプロパティは、全画面表示モードで自動的にページを切り替える際、各ページに設定した値を有効にするかどうかを示します。[setPageTransitions](#) を使用して、各ページの効果のプロパティをプログラムから設定することができます。

useTimer

型：ブーリアン

アクセス：R/W

このプロパティは、全画面表示モードで自動的にページを切り替えるかどうかを示します。次のページに切り替えるまでのデフォルトの時間間隔を設定するには、[timeDelay](#) を参照してください。

Global オブジェクト

Global オブジェクトは静的な JavaScript オブジェクトであり、ドキュメント間でデータを共有したり、セッションをまたがってデータを永続的にしたりすることができます。これを永続的グローバルデータと呼びます。ドキュメント間でのグローバルデータの共有および通知は、*subscribe* メカニズムを介して行われます。これにより、グローバルデータの変数を監視し、値の変化をドキュメント間で通知することができます。

Global オブジェクトのプロパティ

グローバルデータは、Global オブジェクトにプロパティを追加して設定することができます。プロパティの型には、文字列、ブーリアン、または数値が有効です。例えば、「radius」という変数を追加し、すべてのドキュメントのスクリプトからこの変数にアクセス可能にするには、スクリプトを次のように定義するだけです。

```
global.radius = 8;
```

これで、現在のビューアセッションのすべてのドキュメントからグローバル変数「radius」を認識できるようになります。このことをもう少し明確にしてみましょう。例えば、A.pdf および B.pdf という2つのファイルをビューアで開き、A.pdf で上記の宣言を行ったとします。このようにすると、半径が `global.radius` である球の体積を、A.pdf または B.pdf のどちらからでも計算できるようになります。

```
var V = (4/3) * Math.PI * Math.pow(global.radius, 3);
```

これを実行すると、2144.66058 という同じ結果が得られます。*global.radius* の値を変更して上記のスクリプトを再度実行すると、それに応じて *V* の値も変わります。

変数またはプロパティを Global オブジェクトから削除するには、*delete* 演算子を使用して、定義済みのプロパティを削除します。予約済みの JavaScript キーワードである *delete* について詳しくは、[コア JavaScript 1.4 のマニュアル](#)を参照してください。例えば、プロパティ「radius」を Global オブジェクトから削除するには、次のスクリプトを実行します。

```
delete global.radius
```

Global オブジェクトのメソッド

setPersistent



パラメータ : *cVariable*、*bPersist*
戻り値 : なし

このメソッドは、*cVariable* を永続的に設定します。*bPersist* は *true* に設定してください。これは、*cVariable* が Acrobat の起動をまたがって存在できることを意味します。*bPersist* が *false* の場合（グローバルプロパティのデフォルト）、ドキュメントをまたがってプロパティにアクセスすることはできますが、Acrobat ビューアのセッションをまたがることはできません。例えば、「radius」プロパティを永続的にして他のドキュメントからもアクセスできるようにするには、次のようにします。

```
global.radius = 8; // declare radius to be global
global.setPersistent("radius", true); // now say it's persistent
```

上で定義した体積の計算では、`global.radius` の値を変更するまで、ビューアセッションをまたがって同じ結果を得ることができます。

注意： 永続的なグローバルデータは、ブーリアン、数値、文字列型の変数にのみ適用することができ、変数の最大サイズは 32K に制限されます。32K を超えたデータは、すべて切り捨てられます。

Acrobat 用のスクリプトを作成する JavaScript 開発者は、永続的グローバル変数を設定する場合、何らかの命名規則の使用が推奨されます。1 つの方法は、変数名をすべて会社名から始めることです。例えば、会社名が `xyz` の場合は、変数をすべて「`xyz_`」で始めます。このようにすると、複数のドキュメント間で他の永続的グローバル変数との衝突を防ぐことができます。

注意： 永続的なグローバル変数は、アプリケーションの終了時に[フォルダレベルの JavaScript](#) のユーザフォルダに `glob.js` というファイル名で記録され、アプリケーションの開始時に再ロードされます。

subscribe

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

パラメータ : `cVariable`、`fCallback`
戻り値 : なし

このメソッドは、グローバルプロパティである `cVariable` をサブスクライブ（予約、購読）します。このプロパティに変更が発生すると、それが他のドキュメントで加えられた変更であっても、`fCallback` に指定した関数が呼び出されます。1 つのグローバルプロパティに対して、複数のサブスクライブが可能です。

`subscribe` メソッドを使用すると、次の例に示すように、サブスクライブしたグローバル変数値の変更に応じて、自動的にフィールド値を更新することができます。

例 : `setRadius.pdf` および `calcVolume.pdf` という 2 つのファイルを、Acrobat または Reader で開いているとします。

- `setRadius.pdf` には、`global.radius = 2;` というコードを持つボタンを作成します。
- `calcVolume.pdf` には、次のコードを `subscribe` という名前で文書レベルの JavaScript に割り当てます。

```
// In the Tools > JavaScripts > Document JavaScripts
global.subscribe("radius", RadiusChanged);
```

```
function RadiusChanged(x)                                // callback function
{
    var V = (4/3) * Math.PI * Math.pow(x,3);
    getField("MyVolume").value = V;                      // put value in text field
}
```

- 両方のファイルをビューアで開き直し、setRadius.pdf ファイルのボタンをクリックすると、calcVolume.pdf にあるテキストフィールド「MyVolume」の値が直ちに 33.51032 に更新されます (*global.radius* = 2 から計算された値)。

コールバック関数の構文は、次の通りです。


```
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 >
}
```

Identity オブジェクト

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

Identity オブジェクトは、アプリケーションの現在のユーザを識別する静的なオブジェクトです。

Identity オブジェクトのプロパティ

セキュリティ  : ユーザのプライバシーを保護するため、以下のプロパティは、バッチ、コンソール、メニュー、およびアプリケーション初期化イベントでのみアクセス可能です。

corporation

型 : 文字列

アクセス : R

このプロパティは、ユーザ情報パネルに入力されている会社名です。

email

型 : 文字列

アクセス : R

このプロパティは、ユーザ情報パネルに入力されている電子メールアドレスです。

loginName

型 : 文字列

アクセス : R

このプロパティは、オペレーティングシステムに登録されているログイン名です。

name

型 : 文字列

アクセス : R

このプロパティは、ユーザ情報パネルに入力されている名前です。

例 :

```
console.println("Your name is " + identity.name);  
console.println("Your e-mail is " + identity.email);
```

Index オブジェクト

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

Index オブジェクトは、[Search オブジェクト](#)のさまざまなメソッドから返される、作成不可能なオブジェクトです。

Index オブジェクトのプロパティ

available

型：ブーリアン

アクセス：R

このプロパティは、選択および検索を行うために、インデックスが使用可能かどうかを示します。ネットワークがダウンしていたり、CD-ROM が挿入されていなかったり、あるいはインデックスの管理者がインデックスの保守作業をしているなどの理由で、インデックスを使用できないこともあります。

name

型：文字列

アクセス：R

このプロパティは、インデックス管理者がインデックス作成時に指定したインデックスの名前です。

例：

```
// 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);
}
```

このメソッドについて詳しくは、「[Search オブジェクト](#)」の節にある [indexes](#) プロパティを参照してください。このプロパティは、検索エンジンで現在アクセス中の Index オブジェクトの配列を返します。

path

型：文字列

アクセス：R

このプロパティは、インデックスが存在するデバイス依存のパスを示します。パスの正確な構文について詳しくは、『[PDF Reference](#)』の第 3.10.1 節「File Specification Strings」を参照してください。

selected

型：ブーリアン

アクセス：R/W

このプロパティは、検索にインデックスを利用するかどうかを示します。*selected* が *true* の場合、クエリの一環としてインデックスが検索され、*false* の場合は検索されません。このプロパティを設定または解除することは、インデックスリストダイアログで選択状態をチェックすることと同じです。

PlugIn オブジェクト

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

PlugIn オブジェクトは、ビューアの機能を拡張するために追加されている拡張機能を表します。[App オブジェクト](#)の `plugins` メソッドも参照してください。

PlugIn オブジェクトのプロパティ

certified

型：ブーリアン

アクセス：R

このプロパティが `true` の場合、プラグインがアドビシステムズ社によって認証されていることを示します。アプリケーションやドキュメントでセキュリティ違反が発生しないことを確認するため、認証済のプラグインに対しては十分なテストが行われています。ユーザは、認証済のプラグインのみをロードするようにビューアを設定することもできます。

例：

```
var aPlugins = app.plugins;
var j=0;
for (var i=0; i < aPlugins.length; i++)
    if (!aPlugins[i].certified) j++;
console.println("Report: There are " + j + " uncertified plugins loaded.");
```

loaded

型：ブーリアン

アクセス：R

このプロパティが `true` の場合、プラグインがロードされていることを示します。

name

型：文字列

アクセス：R

プラグインの名前を返します。

例：

```
// 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);
```

path

型：文字列

アクセス：R

このプロパティは、プラグインのデバイスに依存しないパスを示します。パスの正確な構文について詳しくは、『[PDF Reference](#)』の第 3.10.1 節「File Specification Strings」を参照してください。

version

型：数値

アクセス：R

プラグインのバージョン番号を返します。バージョン番号の整数部分はメジャーバージョン、小数部分はマイナーバージョンおよびアップデートバージョンを示します。例えば、5.11 は、プラグインのメジャーバージョンが 5、マイナーバージョンが 1、アップデートバージョンが 1であることを示します。

PPKLite 署名ハンドラオブジェクト

PPKLite 署名ハンドラオブジェクトは、Acrobat Self-Sign セキュリティの機能を JavaScript に公開します。

このオブジェクトの取得には、[Security オブジェクト](#)の [getHandler](#) メソッドを使用できます。

[signatureInfo オブジェクト](#)の一連の標準プロパティに加えて、PPKLite 署名ハンドラでは以下のプロパティが公開されます。

プロパティ	型	アクセス	説明
appearance	文字列	R/W	フィールドへの署名時に使用される外観です。
contactInfo	文字列	R/W	電子メールアドレスや電話番号など、信頼性の確認に使用されるユーザ指定の連絡先情報です。
certificates	配列	R	署名者を識別する証明書の階層を示す配列です。最初の要素は署名者の証明書で、以降の要素には、前の要素の証明書を発行した証明機関が続きます。PPKLite の場合、証明書は自己発行なので、エントリは 1 つしかありません。
password	文字列	W	署名フィールドに署名するために必要なパスワードです。

証明書オブジェクトは、次の 6 つの読み取り専用プロパティで構成される汎用オブジェクトです。

プロパティ	型	アクセス	説明
binary	文字列	R	バイナリの証明書を 16 進文字列で返します。
issuerDN	オブジェクト	R	証明書の発行者の識別名 (Distinguished name) です。
MD5Hash	文字列	R	証明書の MD5 ハッシュです。これにより、この証明書が一意に識別されます。
serialNumber	文字列	R	issuerDN と併用され、この証明書を一意に識別します。
SHA1Hash	文字列	R	証明書の SHA1 ハッシュです。これにより、この証明書が一意に識別されます。
subjectCN	文字列	R	署名者の共通名です。
subjectDN	オブジェクト	R	署名者の識別名です。

subjectDN および *issuerDN* は、それ自体が次のプロパティを持つ RDN（Relative Distinguished Name：相対識別名）オブジェクトです。

RDN オブジェクト			
プロパティ	型	アクセス	説明
c	文字列	R	国
cn	文字列	R	共通名
o	文字列	R	組織名
ou	文字列	R	組織単位

例：以下に、これらのプロパティにアクセスする方法を示します。

```
var f = this.getField("mySignature"); // uses the ppk-lite 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);

// now display some information about this the distinguished name of signer
console.println("subjectDN.cn = " + certificate.subjectDN.cn);
console.println("subjectDN.o = " + certificate.subjectDN.o);
```

PPKLite オブジェクトのプロパティ

appearances

5.0			⊗
-----	--	--	---

型：配列

アクセス：R

指定の署名ハンドラで使用可能な外観の、言語に依存しない名前を配列にして返します。署名ハンドラで外観がサポートされていない場合、*null* オブジェクトが返されます。

isLoggedIn

5.0			⊗
-----	--	--	---

型：ブーリアン

アクセス：R

PPKLite 署名ハンドラオブジェクトに現在ログインしている場合は、*true* を返します。

例：

```
var ppklite = security.getHandler("Adobe.PPKLite", true);
console.println( "Is logged in = " + ppklite.isLoggedIn ); // false
ppklite.login( "dps017", "/C/signatures/DPSmith.apf");
console.println( "Is logged in = " + ppklite.isLoggedIn ); // true
```

loginName

5.0			⊗
-----	--	--	---

型：文字列

アクセス：R

署名ハンドラにログインしているユーザ名を返します。誰もログインしていない場合は、空の文字列になります。

loginPath

5.0			⊗
-----	--	--	---

型：文字列

アクセス：R

署名ハンドラへのログインに使用されたユーザのプロファイルファイルのパスを、デバイスに依存しない形式で返します。誰もログインしていない場合は、空の文字列になります。

name

5.0			⊗
-----	--	--	---

型：文字列

アクセス：R

name プロパティは、署名ハンドラの言語に依存しない名前を返します。これは、常に "Adobe.PPKLite" になります。

signInvisible

5.0			⊗
-----	--	--	---

型 : ブーリアン

アクセス : R

署名ハンドラで不可視署名を作成可能かどうかを示します。

signVisible

5.0			⊗
-----	--	--	---

型 : ブーリアン

アクセス : R

署名ハンドラで可視署名を作成可能かどうかを示します。

uiName

5.0			⊗
-----	--	--	---

型 : 文字列

アクセス : R

署名ハンドラの言語に依存した文字列を返します。この文字列は、ユーザインタフェイスでの使用に適しています。

PPKLite オブジェクトのメソッド

login

5.0			⊗
-----	--	--	---

パラメータ : *[cPassword]*、*[cDIPath]*

戻り値 : *bSuccess*

署名ハンドラにログインします。

cPassword は、ユーザのプロファイルにアクセスするために必要なパスワードです。

cDIPath は、ユーザのプロファイルファイルのデバイスに依存しないパスです。

正常にログインすると *true* を返し、それ以外の場合は *false* を返します。

例 :

```
// use "Adobe.PPKLite" handler engine for the UI
var ppklite = security.getHandler("Adobe.PPKLite");
// login
ppklite.login("dps017", "/C/signatures/DPSmith.apf");
```

```
..... make a signature field and sign it .....  
ppklite.logout();
```

署名フィールドへの署名については、[signatureSign](#) を参照してください。

logout

5.0			⊗
-----	--	--	---

パラメータ : なし
戻り値 : なし

署名ハンドラからログアウトします。上記の [login](#) を参照してください。

newUser

5.0			⊗
-----	--	--	---

パラメータ : *cPassword*、*cDIPath*、*oRDN*
戻り値 : なし

このメソッドは、Self-Sign 電子署名のプロファイルファイルを作成します。

必須パラメータである *cPassword* は、作成するプロファイルのパスワードです。

パラメータ *cDIPath* は、新規ユーザプロファイルのデバイスに依存しないパスです。

oRDN は、汎用オブジェクトまたは RDN オブジェクトです。RDN（相対識別名）は、証明書の発行者または署名者名を含むオブジェクトです（[RDN オブジェクト](#)を参照）。RDN または汎用オブジェクトで唯一の必須フィールドは、*cn* です。*c* を指定する場合、ISO 3166 規格に準拠した 2 文字 ("US" など) を使用してください。

例 :

```
// Third parameter specified as a generic object  
var ppklite = security.getHandler("Adobe.PPKLite");  
ppklite.newUser( "testpasswd", "/d/temp/FredNewUser.apf",  
    { cn: "Fred NewUser", c: "US" } );
```

例 : 検証済の署名から証明書を取得し、新規ユーザプロファイルを作成することもできます。

```
var f = this.getField( "mySignature" );  
f.signatureValidate();  
var sigInfo = f.signatureInfo();  
var certs = sigInfo.certificates;  
var issuerDN = certs[0].issuerDN;  
var subjectDN = certs[0].subjectDN;
```

```
// issuerDN and subjectDN have get/set properties cn, o, ou, c
subjectDN.cn = "DP Smith";
ppklite.newUser( "dps017", "/d/profiles/DPSmith.apf", subjectDN );
```

setPasswordTimeout

5.0			⊗
-----	--	--	---

パラメータ : *cPassword*、*iTimeout*
 戻り値 : なし

署名後パスワードが期限切れになるまでの秒数 *iTimeout* を設定します。常に期限切れにする（パスワードが常に必要）には、*iTimeout* を 0 に設定します。パスワードが決して期限切れにならないようにするには、*iTimeout* を 0x7FFFFFFF に設定します。新規ユーザの場合、タイムアウトは 0（パスワードが常に必要）になります。

cPassword は、タイムアウト値の設定に必要な Self-Sign セキュリティパスワードです。

ユーザが PPKLite 署名ハンドラにログインしていない場合、*setPasswordTimeout* で例外が発生します。

例：この例では、PPKLite 署名ハンドラにログインし、パスワードのタイムアウトを 30 秒に設定します。パスワードのタイムアウト時間（この場合は 30 秒）が経過すると、署名者は次に署名をする際、パスワードを入力する必要があります。パスワードが切れていなければ、パスワードを入力する必要はありません。

```
var ppklite= security.getHandler( "Adobe.PPKLite" );
ppklite.login( "dps017", "/d/profiles/DPSmith.apf" );
ppklite.setPasswordTimeout( "dps017", 30 );
```

Report オブジェクト



Report オブジェクトを使用すると、レポートに適した PDF ドキュメントを生成することができます。Report オブジェクトを作成するには、次のように [Report](#) コンストラクタを使用します。

```
var rep = new Report();
```

Report オブジェクトの作成後、レポートの作成やフォーマットに関連するプロパティおよびメソッドを使用することができます。

Report オブジェクトのプロパティ

size

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

型：数値


アクセス：R/W

このプロパティは、[writeText](#) で作成するテキストのサイズを示す乗数です。テキストサイズは、*size* プロパティに定義済スタイルのデフォルトサイズを掛けたものになります。

例：

```
var rep = new Report();
rep.size = 1.2;
rep.writeText("Hello World!");
rep.open("xxx.pdf");
```

absIndent

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



型：数値

アクセス：R/W

このプロパティは、インデントレベルを絶対位置で示します。インデントのオーバーフローを避けるために、なるべく *indent* や *outdent* を使用するようになしてください。

レポートのページの中央を越えるインデントを指定した場合、ページの中央が有効なインデント位置になります。Report.[divide](#) を実行して短い波線が表示される場合、インデントが大きすぎることを示しています。

color

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

型：色

アクセス：R/W

レポートに書き込むテキストや区切り線の色を取得あるいは設定します。

例：

```
var rep = new Report();
rep.size = 1.2;
rep.color = color.blue;
rep.writeText("Hello World!");
rep.open("xxx.pdf");
```

レポートにテキストを書き込むには [writeText](#) を使用し、区切り線（水平線）を書き込むには [divide](#) を使用します。

Report オブジェクトのメソッド



breakPage

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

パラメータ：なし

現在のページを終了し、新しいページを開始します。



divide

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

パラメータ：[nWidth]

ページを横切る水平方向の線を、現在の行に指定の線幅で作成します。線は現在のインデント位置からバウンディングボックスの右端まで引かれます。線に短い波形が表示される場合、インデントがバウンディングボックスの中央を越えていることを示しています。

indent

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


パラメータ：[nPoints]

現在のインデント位置を *nPoints* またはデフォルトの量だけ右に移動します。

レポートのページの中央を越えるインデントを指定した場合、ページの中央が有効なインデント位置になります。Report.[divide](#)() を実行して短い破線が表示される場合、インデントが大きすぎることを示しています。

使用例については、[writeText](#) を参照してください。

outdent

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

パラメータ : *nPoints*

これはインデントの逆で、現在のインデント位置を *nPoints* またはデフォルトの量だけ左に移動します。

使用例については、[writeText](#) を参照してください。

open

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

パラメータ : *cTitle*

戻り値 : *Doc* オブジェクト

レポートの生成を終了して Acrobat 上に開き、[Doc オブジェクト](#)を返します。[Doc オブジェクト](#)はレポートの追加処理を行う場合に使用できます。

例 :

```
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";
```

詳しい例については、[writeText](#) を参照してください。

save

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

パラメータ : *cDIPath*、*[cFS]*

レポートの生成を終了し、デバイスに依存しないパス *cDIPath* にレポートを保存します。


例 :

```
rep.save("/c/myReports/myreport.pdf");
```


オプションの *cFS* には、ファイルシステムを指定します。*cFS* に有効な値は「CHTTP」のみで、この指定をした場合、*cDIPath* パラメータには URL を指定する必要があります。*cFS* は、Web サーバが WebDAV をサポートしている場合のみ使用することができます。

例 :

```
rep.save("http://www.mycompany/reports/myreport.pdf", "CHTTP");
```

セキュリティ : このメソッドは、バッチまたはコンソールイベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。



mail

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

パラメータ: [bUI]、[cTo]、[cCc]、[cBcc]、[cSubject]、[cMsg]

レポートの生成を終了し、レポートをメールで送信します。パラメータは、[mailDoc](#) の場合と同じです。

Report

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

パラメータ: [aMedia]、[aBBox]

これはコンストラクタです。指定のメディアボックスとバウンディングボックス（値の単位はポイント、つまり 1/72 インチ）を使用して、新規の Report オブジェクトを作成します。デフォルトではメディアボックスが 8.5 x 11 インチで、バウンディングボックスはメディアボックスのすべてのサイドから .5 インチだけインデントした大きさになります。

writeText

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

パラメータ: 文字列

テキストのブロックをレポートに書き込みます。書き込みは必ず現在のインデン位置から開始され、Roman、CJK、WGL4 のテキストは正しくワードラップされます。

例:

```
// 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";
```

Report オブジェクトの他の例については、*Annots.js* を参照してください。

Search オブジェクト

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

Search オブジェクトは、Acrobat Search プラグインの機能にアクセスするための静的なオブジェクトです。Search オブジェクトにアクセスするには、このプラグインがインストールされている必要があります ([available](#) プロパティを参照)。

関連オブジェクトには、Search オブジェクトの一部のメソッドから返される [Index オブジェクト](#) があります。

Search オブジェクトのプロパティ

available

型：ブーリアン

アクセス：R

このプロパティは、Search プラグインがロード済でクエリの実行が可能である場合、*true* になります。スクリプトの作成者は、クエリや他の Search オブジェクト操作を行う前に、このブーリアンをチェックしてください。

例：

```
// Make sure the search object exists and is available.
if (typeof search != "undefined" && search.available) {
    search.query("Cucumber");
}
```

indexes

型：配列

アクセス：R

このプロパティは、検索エンジンで現在アクセス可能なすべての [Index オブジェクト](#) を配列にして返します。

例：

```
// 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);
}
```

matchCase

型：ブーリアン

アクセス：R/W

このプロパティは、検索クエリで大文字と小文字が区別されるかどうかを示します。デフォルトは *false* です。

maxDocs

型：整数

アクセス：R/W

このプロパティは、検索クエリの結果として返されるドキュメントの最大数を示します。デフォルトは 100 個のドキュメントです。

proximity

型：ブーリアン

アクセス：R/W

このプロパティは、AND ブーリアン句を含む検索を実行する場合、単語の近似性を検索クエリの結果ランキングに反映するかどうかを示します。デフォルトは *false* です。近似性について詳しくは、『Acrobat Online Guide』の検索機能に関する節を参照してください。

refine

型：ブーリアン

アクセス：R/W

このプロパティは、前回のクエリ結果に対して新しいクエリを適用し、検索結果を絞り込むかどうかを示します。デフォルトは *false* です。絞り込み検索について詳しくは、『Acrobat Online Guide』の検索機能に関する節を参照してください。

soundex

型：ブーリアン

アクセス：R/W

このプロパティは、単語の発音（MacMillan、McMillan、McMilon など）を考慮して検索クエリを実行するかどうかを示します。デフォルトは *false* です。soundex について詳しくは、『Acrobat Online Guide』の検索機能に関する節を参照してください。

stem

型：ブーリアン

アクセス：R/W

このプロパティは、単語の語幹（run、runs、running など）を考慮して検索クエリを実行するかどうかを示します。デフォルトは *false* です。語幹について詳しくは、『Acrobat Online Guide』の検索機能に関する節を参照してください。

thesaurus

型：ブーリアン

アクセス：R/W

このプロパティは、検索クエリで類似した単語を検索するかどうかを示します。例えば、「embellish」を検索すると、「enhanced」、「gracefully」、あるいは「beautiful」などの単語にもヒットします。デフォルトは *false* です。類義語オプションについて詳しくは、『Acrobat Online Guide』の検索機能に関する節を参照してください。

Search オブジェクトのメソッド

addIndex

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

パラメータ：*cDIPath*、*[bSelect]*

戻り値：*Index* オブジェクト

このメソッドは、指定したパスのインデックスを、検索可能インデックスのリストに追加します。

cDIPath には、ユーザのハードドライブにあるインデックスファイルのパスを、デバイスに依存しない形式で指定します。パスの正確な構文について詳しくは、『[PDF Reference](#)』の第 3.10.1 節「File Specification Strings」を参照してください。

オプションパラメータである *bSelect* には、インデックスを検索用に選択するかどうかを指定します。

```
// 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

パラメータ：*cDIPath*

戻り値：*Index* オブジェクト

インデックスリストを検索し、指定のパスに対応するパスを持つ [Index オブジェクト](#) を返します。

cDIPath には、ユーザのハードドライブにあるインデックスファイルのパスを、デバイスに依存しない形式で指定します。パスの正確な構文について詳しくは、『[PDF Reference](#)』の第 3.10.1 節「File Specification Strings」を参照してください。

query

パラメータ：*cQuery*

戻り値：数値

このメソッドは、*cQuery* に指定されたテキストをインデックスリストから検索するよう検索エンジンに指示し、検索によって見つかったドキュメントの数を返します。Search オブジェクトに関連するプロパティ (*stem*、*soundex*、*thesaurus*、*proximity*、*refine*、*maxDocs* など) により、結果が影響を受ける可能性があります。

例 :

```
var nDocs = search.query("Acrobat");
app.alert("You found " + nDocs + " documents that match your query.");
```

removeIndex

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


パラメータ : インデックス
戻り値 : なし

このメソッドは、[Index オブジェクト](#)であるインデックスを、インデックスリストから削除します。

Security オブジェクト

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

Security オブジェクトは、Acrobat の暗号化機能や電子署名機能をカプセル化する、静的な JavaScript オブジェクトです。

セキュリティ  : Security オブジェクトのメソッドおよびプロパティは、バッチ、コンソール、メニュー、またはアプリケーション初期化イベントでのみ実行可能です。Acrobat JavaScript イベントについて詳しくは、[Event オブジェクト](#)を参照してください。

Security オブジェクトのプロパティ

handlers

型 : 配列

アクセス : R

使用可能な署名ハンドラの、言語に依存しない名前を、配列にして返します。[getHandler](#) メソッドも参照してください。

validateSignaturesOnOpen

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

型 : ブーリアン

アクセス : R/W

ドキュメントを開いたときに署名を検証するかどうかを示す、ユーザレベルの設定値を取得または設定します。

Security オブジェクトのメソッド

getHandler

パラメータ : *cName*、*[bUIEngine]*

戻り値 : 署名ハンドラオブジェクト

cName で指定される署名ハンドラオブジェクトを返します。署名ハンドラが存在しない場合、*null* オブジェクトが返されます。[handlers](#) プロパティも参照してください。

2 番目のオプションパラメータ *bUIEngine* はブーリアンで、デフォルトは *false* です。*true* の場合、UI を備えた (UI を介してログインできるなど) 既存の署名ハンドラのエンジンが返されます。*false* (デフォルト) の場合は、新規エンジンが返されます。新規エンジンはいくつでも作成可能で、*getHandler* が呼び出されるたびに新規エンジンが作成されますが、UI エンジンは 1 つだけです。

例：このコードでは、[PPKLite 署名ハンドラオブジェクト](#)を選択しています。

```
// 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("Adobe.PPKLite", true);
// login
ppklite.login("dps017", "/C/signatures/DPSmith.apf");
```

この例の続きは、[signatureSign](#) の例を参照してください。

Sound オブジェクト

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

Sound オブジェクトは、ドキュメントに保持されているサウンドを表します。すべての Sound オブジェクトの配列は、Doc.[sounds](#) プロパティから取得することができます。

[Docオブジェクト](#)の[getSound](#)、[importSound](#)、および[deleteSound](#)メソッドも参照してください。

Sound オブジェクトのプロパティ

name

型：文字列

アクセス：R

このプロパティは、Sound オブジェクトに関連付けられている名前を示します。

例：

```
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 オブジェクトのメソッド

play

パラメータ：なし

戻り値：なし

このメソッドは、サウンドを非同期で再生します。

pause

パラメータ：なし

戻り値：なし

このメソッドは、現在再生中のサウンドを一時停止します。サウンドを既に一時停止していた場合は、サウンドが再開されます。

stop

パラメータ：なし

戻り値：なし

このメソッドは、現在再生中のサウンドを停止します。

Spell オブジェクト

5.0			⊗
-----	--	--	---

JavaScript の Spell オブジェクトを使用すると、フォームのテキストフィールドや注釈などでスペルチェックを行うことができます。Spell オブジェクトを使用するには、Acrobat Spelling プラグインとスペルチェック用の辞書をインストールしておく必要があります。

Spell オブジェクトのプロパティ

available

5.0			⊗
-----	--	--	---

型：ブーリアン

アクセス：R

このプロパティは、Spell オブジェクトが使用可能な場合、`true` になります。

例：

```
console.println("Spell checking available: " + spell.available);
```

dictionaryNames

5.0			⊗
-----	--	--	---

型：配列

アクセス：R

このプロパティは、使用可能な辞書名の配列を返します。この配列のサブセットを [check](#)、[checkText](#)、[checkWord](#) メソッドや、[Doc.spellDictionaryOrder](#) プロパティに渡すと、特定の辞書を使用したり、辞書の検索順を指定したりすることができます。

ユーザのインストールによって有効となる辞書名には「英語：米」、「ドイツ語：旧字体」、「フランス語」、「スペイン語」、「イタリア語」、「英語：英」、「スウェーデン語」、「デンマーク語」、「ノルウェー語」、「オランダ語」、「ポルトガル語」、「ポルトガル語：ブラジル」、「フランス語：カナダ」、「ドイツ語：スイス」、「ノルウェー語：ニーノシク」、「フィンランド語」、「カタロニア語」、「ロシア語」、「ウクライナ語」、「チェコ語」、「ポーランド語」、「英語：英法律」、「英語：英医学」、「ドイツ語：改良綴字法」、「ドイツ語：旧スイス」、「英語：米法律」、「英語：米医学」、「英語：米化学技術」、および「英語：米地理／生物」が含まれます。

dictionaryOrder

5.0			⊗
-----	--	--	---

型：配列

アクセス：R

このプロパティは、ユーザによって「スペルチェック」パネルで検索順が指定されている辞書を、配列にして返します。Spelling プラグインは、最初に [Doc.spellDictionaryOrder](#) 配列（ドキュメントに設定されている場合）の単語を検索し、その後でこのプロパティの配列を検索します。

domainNames

5.0			⊗
-----	--	--	---

型：配列

アクセス：R

このプロパティは、他のプラグインによって Spelling プラグインに登録されているスペルチェック範囲名の配列を返します。この配列のサブセットを [check](#) メソッドに渡すと、スペルチェックの範囲を限定することができます。

ユーザのインストールによって有効となる範囲名には「すべて」、「Form Field」、「All Form Fields」、「注釈」、「すべての注釈」があります。

Spell オブジェクトのメソッド

addDictionary

5.0			⊗
-----	-----------------------------------------------------------------------------------	--	---

パラメータ：cFile、cName、[bShow]

戻り値：ブーリアン

このメソッドは、利用可能な辞書のリストに辞書を追加します。必須の cFile パラメータには、辞書ファイルのパスをデバイスに依存しない形式で指定します。

必須の cName パラメータには、スペルチェックダイアログで使用する辞書の名前を指定します。この名前は、[check](#)、[checkText](#) および [checkWord](#) メソッドの入力パラメータとして使用することができます。

オプションの bShow パラメータが true の場合（デフォルト）、cName パラメータに「ユーザ：」という文字列が付加され、辞書のリストやメニューに表示されます。例えば、cName が「Test」の場合、「ユーザ：Test」がリストやメニューに追加されます。bShow が false の場合、このカスタム辞書の名前はどのリストまたはメニューにも表示されません。

addDictionary メソッドは、成功すると true を返し、それ以外の場合は false を返します。

辞書は、実際には、DDDxxxxx.hyp、DDDxxxxx.lex、DDDxxxxx.clx、および DDDxxxxx.env の 4 ファイルで構成されています。このうち、cFile パラメータには、.hyp ファイルのパスをデバイスに依存しない形式で指定してください。例えば「/c/temp/testdict/TST.hyp」というファイルを指定した場合、Spelling プラグインは TST.hyp ファイルが置かれているフォルダ（testdict）から、他の 3 ファイルを探します。ファイル名の先頭 3 文字は 4 つのファイルで共通にし、かつ他の辞書とは違うものにする必要があります。こうすることにより 4 つのファイルが関連付けられます。Macintosh の場合であっても、ファイル名の最後には、ドットの後に上記の拡張子を付ける必要があります。

例

```
/* Get dictionary path and name from the user */
var dictPath = this.getField("dictPath");
var dictName = this.getField("dictName");

/* now add this dictionary, if possible */
if ( spell.available ) {
    spell.addDictionary(dictPath.value, dictName.value);
}
```

addWord

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

パラメータ : *cWord*、*cName*
戻り値 : ブーリアン

このメソッドは、新しい単語 *cWord* を辞書に追加します。このメソッドは成功すると *true* を返し、それ以外の場合は *false* を返します。

必須の *cName* パラメータは、辞書名です。現在インストールされている辞書の配列は、[dictionaryNames](#) メソッドで取得することができます。

[removeWord](#) メソッドも参照してください。

注意： 内部的には、*Spell Check* オブジェクトによってユーザ辞書「Not-A-Word」がスキャンされ、そこに指定した単語が含まれている場合は単語が削除されます。単語がこの辞書に含まれていなければ、ユーザ辞書に追加されます。実際の辞書は変更されません。

check

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

パラメータ : [*aDomain*]、[*aDictionary*]
戻り値 : ブーリアン

このメソッドは、フォームフィールド、注釈、または他のオブジェクトで、スペルミスのある単語を訂正するためのスペルチェックダイアログを表示します。このメソッドは、フラグの付いたすべての単語をユーザが変更するか無視した場合、*true* を返します。すべての単語をチェックし終える前にユーザがダイアログを終了すると、*false* を返します。

オプションの *aDomain* パラメータは、フォームフィールドや注釈など、*Spelling* プラグインでチェックするドキュメントオブジェクトの配列です。オブジェクトの配列を指定しない場合、「すべて」の範囲がチェックされます。登録済のチェック範囲名の配列は、[domainNames](#) メソッドで取得することができます。

オプションの *aDictionary* パラメータは、スペルチェッカで使用する辞書名の配列です。配列内の辞書の順序は、スペルミスのある単語のチェックに使用される順序です。現在インストールされている辞書の配列は、[dictionaryNames](#) メソッドで取得することができます。このパラメータを省略すると、[Doc.spellDictionaryOrder](#) リスト、[spell.dictionaryOrder](#) リストの順に検索されます。

例：

```
var dictionaries = [" 英語：米法律 ", " 英語：米 "];
var domains = ["All Form Fields", " すべての注釈 "];
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			⊗
-----	--	--	---

パラメータ：*cText*、*[aDictionary]*
戻り値：文字列

このメソッドは、文字列 *cText* 内でスペルミスのある単語を訂正するためのスペルチェックダイアログを表示します。このメソッドは、スペルチェックダイアログの結果を新しい文字列で返します。

オプションの *aDictionary* パラメータは、スペルチェッカで使用する辞書名の配列です。配列内の辞書の順序は、スペルミスのある単語のチェックに使用される順序です。現在インストールされている辞書の配列は、[dictionaryNames](#) メソッドで取得することができます。このパラメータを省略すると、[Doc.spellDictionaryOrder](#) リスト、[spell.dictionaryOrder](#) リストの順に検索されます。

例：

```
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			⊗
-----	--	--	---

パラメータ：*cWord*、*[aDictionary]*
戻り値：*null* または配列

このメソッドは、単語 *cWord* のスペリングをチェックします。スペリングが正しければ *null* オブジェクトが返され、そうでない場合は、置き換える単語の候補が配列で返されます。

オプションの *aDictionary* パラメータは、使用する辞書名の配列です。配列内の辞書の順序は、スペルミスのある単語のチェックに使用される順序です。現在インストールされてい

る辞書の配列は、[dictionaryNames](#) メソッドで取得することができます。このパラメータを省略すると、[Doc.spellDictionaryOrder](#) リスト、[spell.dictionaryOrder](#) リストの順に検索されます。

例：

```
var word = "subpinna"; /* misspelling of "subpoena" */
var dictionaries = [" 英語：米法律 ", " 英語：米 "];
var f = this.getField("Alternatives") /* alternative spellings listbox */
f.clearItems();
f.setItems(spell.checkWord(word, dictionaries));
```

例：次のスクリプトは、ドキュメント全体を調べて、スペルミスのある単語に波形の注釈マークを付けます。波形の注釈の内容には、置き換える単語の候補が書き込まれます。このスクリプトは、コンソールでも、ドキュメント内のマウスボタンを放すアクションとしても、あるいはバッチシーケンスからも実行できます。

```
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()
            });
        }
    }
}
```

removeDictionary

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

パラメータ：cName
戻り値：ブーリアン

このメソッドは、[addDictionary](#) を使用して追加したユーザ辞書を削除します。cName は、[addDictionary](#) で使用した辞書名と同じ名前であればなりません。

`removeDictionary` メソッドは成功すると `true` を返し、それ以外の場合は `false` を返します。

removeWord

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

パラメータ : *cWord*、*cName*
戻り値 : ブーリアン

このメソッドは、単語 *cWord* を辞書から削除します。このメソッドは成功すると *true* を返し、それ以外の場合は *false* を返します。

必須の *cName* パラメータは、辞書名です。現在インストールされている辞書の配列は、[dictionaryNames](#) メソッドで取得することができます。

[addWord](#) メソッドも参照してください。

注意： 内部的には、*Spell Check* オブジェクトによってユーザ辞書がスキャンされ、以前に追加された単語が含まれている場合は単語が削除されます。単語がこの辞書に含まれていなければ、ユーザ辞書「Not-A-Word」に追加されます。実際の辞書は変更されません。

userWords

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

パラメータ : *cName*、*bAdded*
戻り値 : 配列

このメソッドは、ユーザが辞書に追加した単語の配列、あるいは辞書から削除した単語の配列を返します。[addWord](#) および [checkWord](#) メソッドも参照してください。

必須の *cName* パラメータは、辞書名です。現在インストールされている辞書名の配列は、[dictionaryNames](#) プロパティで取得することができます。

必須の *bAdded* パラメータは、2 種類の配列のうちどちらを返すかを示します。*true* の場合はユーザが追加した単語の配列を返し、*false* の場合は削除した単語の配列を返します。

Statement オブジェクト

5.0			ⓧ
-----	--	--	---

Statement オブジェクトは、ADBC の中核です。SQL による更新や照会、およびこれらの結果の取得は、Statement オブジェクトを介して行うことができます。Statement オブジェクトは、[newStatement](#) を使用して作成できます。

注意： 以下のいずれかのメソッドで列を取得した後、再度同じ列を取得しようとする、失敗する可能性があります。

Statement オブジェクトのプロパティ

columnCount

型：数値

アクセス：R

columnCount プロパティは、クエリで返される行に含まれている列数です。更新操作の場合は未定義になります。

rowCount

型：数値

アクセス：R

rowCount プロパティは、更新によって影響を受ける行数です。クエリで返される行数ではありません。クエリコンテキストでは値が未定義になります。

Statement オブジェクトのメソッド

execute

パラメータ：cSQL

戻り値：ブーリアン

execute メソッドは、Statement オブジェクトのコンテキストで SQL ステートメントを実行します。このメソッドは成功すると *true* を返し、失敗すると *false* を返します。

例：

```
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. */
statement.execute("Select firstname, lastname, ssn from ¥\"Employee Info¥\"");
statement.execute("Select ¥\"First Name¥\" from ¥\"Client Data¥\"");
```



```

/* 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. */
statement.execute(' Select  "First Name","Second Name" from "Client Data" ');

```

詳しい例については、[getRow](#) および [nextRow](#) メソッドを参照してください。

注意： `execute` が失敗せず、すべてのデータが返されてしまうようなことがないとしても、それはステートメントが正しいという保証にはなりません。

getColumn

パラメータ : `nColumn`、`[nDesiredType]`
 戻り値 : `Column` オブジェクトまたは `null`

`getColumn` メソッドは、`nColumn` パラメータで識別される列データを表す [Column オブジェクト](#) を返します。`nColumn` パラメータには、数値または文字列を指定できます。数値の場合は、列番号に基づいた `Column` オブジェクトが返され、文字列の場合は、列名に基づいた `Column` オブジェクトが返されます。オプションの `nDesiredType` パラメータには、列データの表現に最も適した [JavaScript 型](#) を指定します。`getColumn` メソッドは、失敗すると `null` を返します。

次の表に、[Column オブジェクト](#) のプロパティを示します。

Column オブジェクト			
Column オブジェクトは、1 列にすべての行のデータを含む汎用オブジェクトです。Column オブジェクトは、 getColumn または getColumnArray で取得することができます。			
プロパティ	型	アクセス	説明
<code>columnNum</code>	数値	R	取得した Column オブジェクトの列を、Statement オブジェクト内で識別する数値。
<code>name</code>	文字列	R	列の名前です。 ColumnInfo オブジェクト の <code>name</code> プロパティに似ています。
<code>type</code>	数値	R	列に含まれるデータの SQL 型 です。 ColumnInfo オブジェクト の <code>type</code> プロパティに似ています。
<code>typeName</code>	文字列	R	列のデータ型を示す名前です。 ColumnInfo オブジェクト の <code>typeName</code> プロパティに似ています。
<code>value</code>	各種	R/W	列のデータ値。データ型はデータが取得されたときの本来の形式になります

getColumnArray

パラメータ：なし

戻り値：Column オブジェクトの配列または null

`getColumnArray` メソッドは、結果セットに含まれる各列の `Column` オブジェクトを配列にして返します。列の最適な [JavaScript 型](#) を決定するために、「最良の推測」が行われます。このメソッドは失敗すると、長さゼロの配列のほかに `null` を返すことがあります。

[getColumn](#) メソッドの説明の最後に、[Column オブジェクト](#)のプロパティ一覧が記載されています。

getRow

パラメータ：なし

戻り値：Row オブジェクト

`getRow` メソッドは、現在の行を表す [Row オブジェクト](#) を返します。このオブジェクトには、各列の情報が含まれています。[getColumnArray](#) と同様、列のデータ型は「最良の推測」によって決定されます。

以下の表およびその後の説明に、[Row オブジェクト](#)のプロパティを示します。

Row オブジェクト			
Row オブジェクトは、1 行にすべての列のデータを含む汎用オブジェクトです。Row オブジェクトは、 getRow で取得することができます。			
プロパティ	型	アクセス	説明
<code>columnArray</code>	配列	R	Row オブジェクトの作成と getColumnArray の呼び出しを、同一の Statement オブジェクトで同時に行った場合、 getColumnArray から返される配列と <code>columnArray</code> プロパティは同じになります。

`columnArray` プロパティに加え、[Row オブジェクト](#)には、クエリで選択された各列を表すプロパティがあります。

すべての Row オブジェクトには、データ行の各列のプロパティが含まれています。次に例を示します。

```
statement.execute("SELECT firstname, lastname, ssn FROM ¥\"Employee Info¥\"");
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);
```

例：列名にスペースが含まれている場合、上記の構文（row.firstname.value など）では行プロパティにアクセスすることはできません。代わりに、次のようにします。

```
Connect = ADBC.newConnection("Test Database");
statement = Connect.newStatement();
statement.execute(' Select  "First Name","Second Name"  from "Client Data" ');
statement.nextRow();
row=statement.getRow();

// Populate this PDF file
this.getField("name.first").value = row["First Name"].value;
this.getField("name.last").value = row["Second Name"].value;
```

nextRow

パラメータ：なし

戻り値：なし

`nextRow` メソッドは、クエリで生成されたデータの次の行のデータを取得します。[execute](#) で結果セットを得た後で最初の行を取得するには、このメソッドを呼び出す必要があります。このメソッドが（次の行が存在しないなどの理由で）失敗すると、例外が発生します。

例：次の例では、2つのボタンと文書レベルの JavaScript を作成し、データベースから取得したデータを PDF フォームに入力する簡単な方法を示しています。

以下で定義している `getNextRow` ボタンでは、（次の行が存在しないことを示す）例外が発生しない限り、`nextRow` を使用してデータベースから次の行を取り出します。例外が発生した場合はデータベースに再接続し、`nextRow` を使用してデータの最初の行を（再度）取り出します。

```
/* 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;
}

```

Template オブジェクト

Template オブジェクトは、ドキュメント内の名前付きのページです。これらのテンプレートページは表示／非表示を切り替えることが可能で、そのコピーを作成したり、他のページに適用したりすることができます。Template オブジェクトは、コンテンツの動的な作成に使用するのが一般的です（請求書のページが足りなくなったときにページを追加するなど）。

Template オブジェクトのプロパティ

hidden

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

型：ブーリアン

アクセス：R/W

このプロパティは、テンプレートを非表示にするかどうかを示します。非表示にされたテンプレートは、そのコピーが作成されるか非表示が解除されるまで、ユーザから確認することはできません。非表示のテンプレートを表示させた場合、ドキュメントの最後に追加されます。

[templates](#) プロパティ、[Doc オブジェクト](#)の [createTemplate](#)、[getTemplate](#)、[removeTemplate](#) メソッド、および [Template オブジェクト](#) も参照してください。

注意： *Acrobat Reader* では、このプロパティを取得することはできますが、設定しようとする例外が発生します。

name

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

型：文字列

アクセス：R

このプロパティは、テンプレートの作成時に指定されたテンプレート名を返します。

[templates](#) プロパティ、[Doc オブジェクト](#)の [createTemplate](#)、[getTemplate](#)、[removeTemplate](#) メソッド、および [Template オブジェクト](#) も参照してください。

Template オブジェクトのメソッド

spawn

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

パラメータ：[nPage]、[bRename]、[bOverlay]
戻り値：なし

テンプレートに基づいて、ドキュメントに新しいページを作成します。

nPage には、テンプレートを適用（オーバーレイ）するページ番号（0 ベース）を指定します。*nPage* のデフォルトは 0 です。

bRename は、ページ上のフォームフィールド名を変更すべきかどうかを示すブーリアンです。*bRename* のデフォルトは *true* です。

bOverlay は、テンプレートをページに適用するのか、指定ページの前に新規ページとして挿入するのかを示すブーリアンです。*bOverlay* のデフォルトは *true* です。ページをドキュメントの最後に追加する場合は、*bOverlay* を *false* に設定し、*nPage* をドキュメントのページ数に設定します。

例：

```
var a = this.templates;
for (i = 0; i < a.length; i++)
    a[i].spawn(this.numPages, false, false);
```

この例では、すべてのテンプレートのコピーを作成し、それを 1 つずつドキュメントの最後に追加しています。

[templates](#) プロパティ、[Doc オブジェクト](#)の [createTemplate](#)、[getTemplate](#)、[removeTemplate](#) メソッド、および [Template オブジェクト](#)も参照してください。

TTS オブジェクト

4.05			
------	--	--	--

JavaScript の TTS オブジェクトを使用すると、テキストを音声に変換することができます。TTS オブジェクトを使用するには、ユーザのマシンに Text-To-Speech エンジンがインストールされている必要があります。Text-To-Speech エンジンとは、テキストをデジタルオーディオとしてレンダリングすることで「話をします」。ほとんどの場合、Text-To-Speech エンジンはアクセシビリティを意識して実装されていますが、PDF ドキュメントを活性化するために、その他さまざまなアプリケーションでの利用も考えられます。

これは現時点では Windows 専用の機能で、[Microsoft Text to Speech](#) エンジンがオペレーティングシステムにインストールされている必要があります。

TTS オブジェクトは JavaScript オブジェクトなので、Windows および Macintosh のどちらのプラットフォームにも存在しますが、Macintosh では使用不可になっています。

注意： Acrobat 5.0 では仕様が大幅に変更され、障害のあるユーザにアクセシビリティを提供するために、一般的なスクリーンリーダーを直接統合する方法が採用されました。このため、バージョン 4.05 で定義されていた一部の TTS オブジェクトの機能およびメソッドは、スクリーンリーダーと競合するようになったため、使用が推奨されなくなりました。ただし、それらのオブジェクトも、マルチメディアドキュメントで評判の良い便利な機能を備えているため、当面残されています。

TTS オブジェクトのプロパティ

available

型：ブーリアン

アクセス：R

このプロパティは、TTS オブジェクトと Text-To-Speech エンジンが使用可能である場合、true を返します。

```
console.println("Text to speech available: " + tts.available);
```

numSpeakers

型：整数

アクセス：R

現在の Text-To-Speech エンジンで使用可能なスピーカの数 را返します。[speaker](#) プロパティおよび [getNthSpeakerName](#) メソッドも参照してください。

pitch

型：整数

アクセス：R/W

このプロパティは、スピーカの音声のベースラインピッチを設定します。ピッチの有効範囲は 0 ～ 10 で、デフォルトは 5 です。

soundCues

☹			
---	--	--	--

型：ブーリアン

アクセス：R/W

このプロパティの使用は推奨されていません。現時点では *false* のみを返します。

speaker

型：文字列

アクセス：R/W

このプロパティを使用すると、Text-To-Speech を実行する際に、ユーザが異なる音質のスピーカを指定することができます。[numSpeakers](#) プロパティおよび [getNthSpeakerName](#) メソッドも参照してください。

speechCues

☹			
---	--	--	--

型：ブーリアン

アクセス：R/W

このプロパティの使用は推奨されていません。現時点では *false* のみを返します。

speechRate

型：整数

アクセス：R/W

このプロパティは、Text-To-Speech エンジンによる音声の速度を示します。[speechRate](#) の値は、1 分あたりの単語数で表されます。

volume

型：整数

アクセス：R/W

このプロパティは、音量を示します。有効範囲は、0（ミュート）～ 10（最大）です。

TTS オブジェクトのメソッド

getNthSpeakerName

パラメータ : *nIndex*
戻り値 : *cName*

このメソッドは、現在インストールされている Text-To-Speech エンジンで利用可能な、*n* 番目のスピーカの名前を取得します ([numSpeakers](#) および [speaker](#) プロパティも参照)。

例 :

```
// 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

パラメータ : なし
戻り値 : なし

このメソッドは、TTS オブジェクトの Text-To-Speech の出力を一時停止します。待ち状態になっている残りのテキストを再生するには、[resume](#) メソッドを使用します。

qSilence

パラメータ : *nDuration*
戻り値 : なし

このメソッドは、一定時間の無音をテキストのキューに入れます。*nDuration* には、無音時間をミリ秒単位で指定します。

qSound

パラメータ : *cSound*
戻り値 : なし

このメソッドは、サウンドをキューに入れて、[talk](#) で再生できるようにします。パラメータ *cSound* には、有効なサウンドキューの名前のリストから 1 つを指定します。これらの名前は、SoundCues フォルダに保存されているサウンドファイル（存在する場合）に直接マッピングされます。

```
tts.qSound("DocPrint");           // Plays DocPrint.wav
```

SoundCues フォルダは、C:\Program Files\Adobe\Acrobat 5.0\SoundCues というように、ビューアのプログラムレベルに配置する必要があります。

注意： Windows 専用 `--qSound` は、22KHz16 ビットの PCM.wav ファイルのみ扱うことができます。また、MS SAPI のキューの遅延問題を避けるため、サウンドに 1 秒以上の長さが必要です。1 秒未満の場合は、サウンドを編集して最後に無音を追加する必要があります。

qText

パラメータ：cText
戻り値：なし

このメソッドは、テキストをキューに入れて、[talk](#) で再生できるようにします。

cText には、音声に変換するテキストを指定します。

```
tts.qText("Hello, how are you?");
```

reset

パラメータ：なし
戻り値：なし

このメソッドは、現在キューに入っているテキストの再生を停止し、キューをフラッシュします。[resume](#) メソッドでテキストの再生を再開することはできません。また、TTS オブジェクトのプロパティはすべてデフォルト値にリセットされます。

resume

パラメータ：なし
戻り値：なし

このメソッドは、一時停止した TTS オブジェクトのテキストの再生を再開します。

stop

パラメータ：なし
戻り値：なし

このメソッドは、現在キューに入っているテキストの再生を停止し、キューをフラッシュします。[resume](#) メソッドでテキストの再生を再開することはできません。

talk

パラメータ：なし

戻り値：なし

このメソッドは、キューの内容を音声にするため Text-To-Speech エンジンに送ります。テキストの音声出力が一時停止されていた場合は、[talk](#) によってキューのテキストの再生が再開されます。

```
tts.qText("Hello there!");  
tts.talk();
```

this オブジェクト

JavaScript の特殊なキーワードである *this* は、現在のオブジェクトを指します。Acrobat では、現在のオブジェクトは次のように定義されます。

- オブジェクトのメソッドでは、メソッドが属すオブジェクト。
- コンストラクタ関数では、構築中のオブジェクト。
- [フォルダレベルの JavaScript](#) ファイルで定義される関数では、未定義。これらの関数でドキュメントオブジェクトを必要とする場合、呼び出し側の関数からドキュメントオブジェクトを渡してください。
- [文書レベル](#)または[フィールドレベル](#)のスクリプトでは、ドキュメントオブジェクト。これを使用して、ドキュメントプロパティの取得や設定、メソッドの呼び出しをすることができます。

例えば、次の関数が Plug-in フォルダレベルで定義されているとします。

```
function PrintPageNum(doc)
{ /* Print the current page number to the console. */
  console.println("Page=" + doc.pageNum);
}
```

次のスクリプトは、現在のページ番号をコンソールに 2 回出力し、その後ページを印刷します。

```
PrintPageNum(this); /* Must pass the document object. */
console.println("Page=" + this.pageNum); /* Same as the previous call. */
this.print(false, this.pageNum, this.pageNum); /* Prints the current page. */
```

変数名と関数名の衝突

スクリプトで定義される変数と関数は、*this* オブジェクトを親とします。次に例を示します。

```
var f = this.getField("Hello");
```

これは、以下と同じことです。

```
this.f = this.getField("Hello");
```

ただし、変数の *f* は、スクリプトの実行後にガーベジコレクトされ得るという違いはあります。

Acrobat JavaScript プログラマは、[Doc オブジェクト](#)のプロパティ名やメソッド名を、変数名に使用しないでください。次のように、予約語「var」の後にメソッド名を使用すると、例外が発生します。

```
var getField = 1; // TypeError: redeclaration of function getField
```

プロパティ名を使用しても例外は発生しませんが、プロパティがオブジェクトを参照している場合、プロパティの値を変更できないことがあります。

```
// "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]
```

次の例は [signatureInfo オブジェクト](#) の表の後のコードから引用したもので、変数名の衝突を避けるための 1 例を示しています。

```
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 オブジェクト

Util オブジェクトは静的な JavaScript オブジェクトで、数多くのユーティリティメソッド、および文字列と日付のフォーマットや解析を行う便利な関数を定義しています。

Util オブジェクトのメソッド

printf

パラメータ : *cFormat*、*
戻り値 : *cResult*

このメソッドは、1 つ以上の値を、フォーマット文字列に従って文字列としてフォーマットします。同名の C 関数に似ています。このメソッドは、フォーマット文字列 (*cFormat*) に従って入力引数を変換およびフォーマットし、その結果を文字列 (*cResult*) として返します。フォーマット文字列は、2 種類のオブジェクトで構成されます。1 つは通常の文字で、結果文字列にコピーされます。もう 1 つは変換仕様で、各仕様が printf の 2 番目の引数に対応し、変換およびフォーマットを行います。各変換仕様は、次のように構成されています。

`%[,nDecSep][cFlags][nWidth][nPrecision]cConvChar`

cDecSep には、小数点／区切り文字のフォーマットを、コンマ (,) に続く 0 ～ 3 の数値で指定します。

- 0 - 区切り文字にコンマ、小数点にピリオドを使用します。
- 1 - 区切り文字はなく、小数点にピリオドを使用します。
- 2 - 区切り文字にピリオド、小数点にコンマを使用します。
- 3 - 区切り文字はなく、小数点にコンマを使用します。

cFlags は数値変換にのみ有効で、仕様を変更するいくつかの文字で構成されています（順序は任意）。

- + - 数値を常に符号付きでフォーマットします。
- space* - 最初の文字が符号でない場合は、スペースを前に置きます。
- 0 - フィールドを先行ゼロで埋めます。
- # - 代替出力形式を指定します。f の場合、常に出力に小数点を付けます。

nWidth は、フィールドの最小幅を指定する数値です。変換後の引数は、符号と小数点を含めて最低でもここで指定した文字数にフォーマットされ、必要に応じてそれ以上の文字数になることもあります。変換後の引数の文字数がフィールド幅より少ない場合は、値の左側を埋めてフィールド幅になるようにします。埋め込み文字には通常はスペースが使用されますが、ゼロの埋め込みフラグが指定されている場合は 0 を使用します。

nPrecision には、浮動小数点数に変換をする場合の小数点以下の桁数を、ピリオド (.) に続く数値で指定します。

cConvChar は、次のいずれかになります。

d - 整数。引数を整数として解釈します（必要に応じて切り捨てられます）。

f - 浮動小数点数。引数を数値として解釈します。

s - 文字列。引数を文字列として解釈します。

x - 16 進数。引数を整数として解釈し（必要に応じて切り捨てられます）、符号なしの 16 進表記にフォーマットします。

例：

```
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));
```

出力：

```
Decimal format: 314
Hex format: 13A
Float format: 314.16
String format: 314.159265358979
```

printf

パラメータ：*cFormat*、*oDate*
戻り値：文字列

このメソッドは、フォーマットに従って日付をフォーマットします。*cFormat* は、文字列または数値のどちらかで、*date* は日付オブジェクトでなければなりません。

cFormat パラメータで使える文字列フォーマットの値は、次の通りです。

文字列	説明	例
mmmm	月の名称	September
mmm	月の略称	Sept
mm	月を表す先行ゼロ付きの数値	09
m	月を表す先行ゼロのない数値	9
dddd	曜日の名称	Wednesday

文字列	説明	例
ddd	曜日の略称	Wed
dd	先行ゼロ付きの日	03
d	先行ゼロのない日	3
yyyy	4桁の年	1997
yy	2桁の年	97
HH	先行ゼロ付きの24時間制の時間	09
H	先行ゼロのない24時間制の時間	9
hh	先行ゼロ付きの12時間制の時間	09
h	先行ゼロのない12時間制の時間	9
MM	先行ゼロ付きの分	08
M	先行ゼロのない分	8
ss	先行ゼロ付きの秒	05
s	先行ゼロのない秒	5
tt	am/pm	am
t	1文字で表した am/pm	a
¥	エスケープ文字として使用	

例えば、現在の日付を省略しない形式でフォーマットするには、次のようにします。

```
var d = new Date();
console.println(util.printd("mmmm dd, yyyy", d));
```

5.0 追記事項

数値で「簡単に」指定できるフォーマットがいくつか追加されました。

値	説明	例
0	PDF 日付フォーマット	D:20000801145605+07'00'
1	Universal	2000.08.01 14:56:05 +07'00'
2	ローカル文字列	2000/08/01 14:56:05

例：

```
// display date in a local format
var d = new Date();
console.println(util.printd(2, d));
```


printx

パラメータ : *cFormat*、*[cSource]*
戻り値 : 文字列

このメソッドは、フォーマット文字列 *cFormat* に従って、ソース文字列 *cSource* をフォーマットします。フォーマット文字列には、特殊なマスク文字を含めて、どんな文字列でも使用することができます。

値	説明
?	次の文字をコピーします。
X	次の英数字をコピーし、他の文字はスキップします。
A	次の英字をコピーし、他の文字はスキップします。
9	次の数値をコピーし、他の文字はスキップします。
*	これ以降の残りのソース文字列をコピーします。
¥	エスケープ文字です。
>	次の指示があるまで、大文字に変換します。
<	次の指示があるまで、小文字に変換します。
=	次の指示があるまで、大文字／小文字を保持します（デフォルト）。

例えば、文字列を米国の電話番号としてフォーマットするには、次のようにします。

```
var v = "aaa14159697489zzz";  
v = util.printx("9 (999) 999-9999", v);  
console.println(v);
```

scand

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

パラメータ : *cFormat*、*cDate*
戻り値 : 日付オブジェクト

このメソッドは、*cFormat* に指定したフォーマット文字列に従って、指定の日付 *cDate* を JavaScript の日付オブジェクトに変換します。*cFormat* では、[printd](#) の場合と同じ構文が使用されます。以下のコードは、日付コンストラクタを直接使用する場合に比べ、はるかに柔軟性に富んでいます。

```
/* 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 mmm dd", d));
```

注意： [scand](#) に 2 桁の年が与えられた場合、あいまいさを解決する処理が行われます。つまり、年が 50 未満の場合は 21 世紀（2000 を加える）であるとみなされ、50 以上の場合は 20 世紀（1900 を加える）とみなされます。この手法は Date Horizon と呼ばれることがあります。

Acrobat JavaScript に関する簡単な FAQ¹

JavaScript はどこにあり、どのように使用するのですか？

JavaScript は Acrobat の様々なレベルで動作します。それはフォルダレベル、文書レベル、フィールドレベル、バッチレベルです（[JavaScript の使用場所](#)の節も参照）。それぞれのレベルで可能な処理は異なり、ロードされるタイミングも違います。

フォルダレベルの JavaScript

拡張子を ".js" とした個々のファイルに JavaScript を配置することができます。ビューアの起動時にこれらのファイルを読み込むには、Acrobat アプリケーションの *JavaScripts* フォルダまたはユーザの *JavaScripts* フォルダにファイルを入れておく必要があります。アプリケーションの起動時にファイルを読み込む順序については、[App / Init](#) を参照してください。

アプリケーションで使用する便利な変数や関数は、これらのフォルダに入れてください。JavaScript メソッドの中には、App.[addItem](#) や App.[addSubMenuItem](#) のように、起動時にどれか 1 つの JavaScript ファイルからしか実行できないものもあるので注意してください。

JavaScript ファイルを作成する場合、いくつかの制約がありますが、特に [this オブジェクト](#) の使用に関して注意が必要です。

Acrobat JavaScript の標準の実装には、3 つの JavaScript ファイルが含まれています。組み込みの関数を含む *Aform.js*、注釈プラグインで使用する *Annotations.js*、そして ADBC プラグインで使用する *ADBC.js* です。これらのファイルは Acrobat の *JavaScripts* フォルダに入っています。

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

glob.js はプログラムによって生成されるファイルで、グローバルオブジェクトの [setPersistent](#) メソッドによってセッション間のアプリケーション設定が保存されます。

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

Config.js ファイルは、ビューアのツールバーボタンやメニュー項目を削除して、外観をカスタマイズするために使用されます（アプリケーションメソッドの [hideMenuItem](#) および [hideToolbarButton](#) を参照）。

¹ よくある質問

文書レベル

Adobe Acrobat メニューの ツール / JavaScript / 文書レベル JavaScript の編集を選択して文書レベルのスクリプトを追加、変更、削除することができます。ここにはドキュメント外に適用する関数ではなく、ドキュメント内全般から使用するのに便利な関数を定義します。文書レベルのスクリプトは、ドキュメントが開かれてプラグインレベルのスクリプトが読み込まれた後に実行されます。文書レベルのスクリプトは PDF ドキュメント内に保存されるので、フォームを作成するプログラマは、スクリプトをできるだけ効果的にパッケージする必要があります。

フィールドレベル

フォームの編集ツールにあるダイアログボックスを使用して、スクリプトをフォームフィールド定義に追加できます。これらのスクリプトは、イベント（マウスボタンを放す、または計算など）が発生すると実行されます。フィールド固有のスクリプトはこのレベルで作成する必要があります。これらのスクリプトは通常、フィールドの値を検証、フォーマット、あるいは計算します。

文書レベルおよびフィールドレベルのスクリプトは、プラグインフォルダスクリプトとは異なり、PDF ドキュメント内に保存されます。したがって、パフォーマンスやファイルサイズの理由により、フォームを作成するプログラマはスクリプトをさまざまなレベルで、出来るだけ効果的（コードの再利用など）にパッケージする必要があります。

フォームフィールドにはどのように名前を付けるのですか？

フォームフィールドには、*FirstName* や *LastName* というような名前を付けるのが一般的です。このような形式の名前をフラットネームと呼びます。多くのフォームアプリケーションでは、この名前の階層だけで十分であり、問題なく動作します。フラットネームの問題点は、フィールド間の関連付けが無いということです。

階層構造を作成すると、フォームフィールド名をさらに便利に使えるようになります。例えば *FirstName* を *Name.First* に変更し、*LastName* を *Name.Last* に変更することで、フィールドのツリーを形成することができます。ピリオド（「.」）は、Acrobat Forms で階層の区切りを表すために使用されるセパレータです。これらのフィールドの *Name* の部分が階層の親で、*First* および *Last* が子になります。名前の階層の深さに制限はありませんが、管理可能な範囲にしておくことが重要です。

この階層はさまざまな用途に便利に使えます。たとえば作成作業の効率を上げたり、JavaScript でフィールドのグループを簡単に操作することができます。また、ドキュメントに多数のフィールドが含まれている場合、フォームフィールドの階層を使用することで、フォームアプリケーションのパフォーマンスを向上させることができます。

フラットネームである *FirstName*、*MiddleName*、*LastName* を使用して、複数のフィールドの背景色を黄色に変更するとします（データが欠落していることを示したり、重要な部分を強調するため）。この場合、各フィールドの処理ごとに 2 行の JavaScript コードを必要とします。

```
var name = this.getField("FirstName");  
name.fillColor = color.yellow;
```

```
name = this.getField("MiddleName");
name.fillColor = color.yellow;
name = this.getField("LastName");
name.fillColor = color.yellow;
```

これを上記の *Name.First*、*Name.Middle*、*Name.Last*（他に *Name.Title* などを付け足すこともできます）という階層を使用して処理すると、6 行のコードをわずか 2 行にして背景色を変更することができます。

```
var name = this.getField("Name");
name.fillColor = color.yellow
```

この階層を JavaScript で操作する場合、変更できるのは親フィールドの外観に関するプロパティだけであり、外観に加えた変更はすべての子にも適用されることに注意してください。フィールドの値は変更できません。例えば、次のようなスクリプトを試してみます。

```
var name = this.getField("Name");
name.value = "Lincoln";
```

このスクリプトは失敗します。なぜなら *Name* は親フィールドだからです。値を変更できるのは、階層の最後のフィールド（*Last* や *First* など、子を持たないフィールド）だけです。次のスクリプトは正しく実行されます。

```
var first = this.getField("Name.First");
var last = this.getField("Name.Last");
first.value = "Abraham";
last.value = "Lincoln";
```

日付オブジェクトはどのように使用するのですか？

ここでは、Acrobat の Date オブジェクトについて説明します。このマニュアルでは、JavaScript の Date オブジェクトおよび日付を処理する [Util オブジェクト](#) 関数を十分に理解していることが前提となります。JavaScript の Date オブジェクトは、実際には日付と時刻の両方を含むオブジェクトです。ですから、ここで「日付」と言う場合、日付と時刻の組み合わせを意味しています。

注意： このマニュアルに記載されているメソッドも含め、JavaScript の日付処理はすべて 2000 年問題（Y2K）に対応済です。

ヒント： Date オブジェクトを使用する場合、現在の年から 1900 を引いた値を返す *getYear()* メソッドを使用するのではなく、常に 4 桁の年を返す *getFullYear()* メソッドを使用してください。

日付から文字列への変換

JavaScript の Date オブジェクトに加えて、Acrobat Forms には日付に関連するメソッドがいくつか用意されています。Date オブジェクトと文字列との間の変換には、これらのメソッドを使用することもできます。Acrobat Forms は数多くの日付フォーマットを扱っているため、Date オブジェクトではこれらを正しく変換できないことがあります。

Date オブジェクトを文字列に変換するには、[Util オブジェクト](#)の [printd](#) メソッドを使用します。組み込みの変換と異なり、[printd](#) では日付を指定通りにフォーマットすることができます。

```
/* Example of util.printd */
var d = new Date(); /* Create a Date object containing the current date. */
/* Create some strings from the Date object with various formats with
** util.printd */
var s = [ "mm/dd/yy", "yy/m/d", "mmmm dd, yyyy", "dd-mmm-yyyy" ];
for (var i = 0; i < s.length; i++) {
    /* print these strings to the console */
    console.println("Format " + s[i] + " looks like: " + util.printd(s[i], d));
}
```

このスクリプトの出力は、次のようになります。

```
Format mm/dd/yy looks like: 01/15/99
Format yy/mm/dd looks like: 99/1/15
Format mmmm dd, yyyy looks like: January 15, 1999
Format dd-mmm-yyyy looks like: 15-Jan-1999
```

ヒント： 人間の寿命は延びていますし、Y2K の教訓もあります。4 桁の年で日付を出力するようにして、あいまいさを無くすようにしてください。

文字列から日付への変換

文字列を Date オブジェクトに変換するには、[Util オブジェクト](#)の [scand](#) メソッドを使用します。このメソッドは、文字列に含まれる年、月、日を取得するための手がかりとして、フォーマット文字列を引数に取ります。

```
/* Example of util.scand */
/* Create some strings containing the same date in differing formats. */
var s1 = "03/12/97";
var s2 = "80/06/01";
var s3 = "December 6, 1948";
var s4 = "Saturday 04/11/76";
var s5 = "Tue. 02/01/30";
var s6 = "Friday, Jan. the 15th, 1999";
```

```
/* Convert the strings into Date objects using util.scand */
var d1 = util.scand("mm/dd/yy", s1);
var d2 = util.scand("yy/mm/dd", s2);
var d3 = util.scand("mmmm dd, yyyy", s3);
var d4 = util.scand("mm/dd/yy", s4);
var d5 = util.scand("yy/mm/dd", s5);
var d6 = util.scand("mmmm dd, yyyy", s6);
/* Print the dates to the console using util.printd */
console.println(util.printd("mm/dd/yyyy", d1));
console.println(util.printd("mm/dd/yyyy", d2));
console.println(util.printd("mm/dd/yyyy", d3));
console.println(util.printd("mm/dd/yyyy", d4));
console.println(util.printd("mm/dd/yyyy", d5));
console.println(util.printd("mm/dd/yyyy", d6));
```

このスクリプトの出力は、次のようになります。

```
03/12/1997
06/01/1980
12/06/1948
04/11/1976
01/30/2002
01/15/1999
```

[scand](#) は日付コンストラクタ（`new Date(...)`）と異なり、渡される文字列に関する規則が厳密ではありません。

注意： [scand](#) に 2 桁の年が与えられた場合、あいまいさを解決する処理が行われます。つまり、年が 50 未満の場合は 21 世紀（2000 を加える）であるとみなされ、50 以上の場合は 20 世紀（1900 を加える）とみなされます。この手法は Date Horizon と呼ばれることがあります。

日付の演算

日付の演算を行って、2 つの日付の間の経過時間や、数日後あるいは数週間後の日付を求めたいことがあります。JavaScript の Date オブジェクトには、この処理を行う方法がいくつか用意されています。最も単純で分かりやすい方法は、日付を数値表現によって処理する方法です。JavaScript では、日付は固定日時からのミリ秒（1000 ミリ秒は 1 秒）の数値で内部的に保持されており、Date オブジェクトの `valueOf` メソッドによって取得することができます。Date コンストラクタを使用して、この数値から新しい日付を構築することもできます。

```

/* Example of date arithmetic. */
/* Create a Date object with a definite date. */
var d1 = util.scand("mm/dd/yy", "4/11/76");
/* Create a date object containing the current date. */
var d2 = new Date();
/* Number of seconds difference. */
var diff = (d2.valueOf() - d1.valueOf()) / 1000;
/* Print some interesting stuff to the console. */
console.println("It has been " + diff + " seconds since 4/11/1976");
console.println("It has been " + diff / 60 + " minutes since 4/11/1976");
console.println("It has been " + (diff / 60) / 60 + " hours since 4/11/1976");
console.println("It has been " +
    ((diff / 60) / 60) / 24 + " days since 4/11/1976");
console.println("It has been " +
    (((diff / 60) / 60) / 24) / 365 + " years since 4/11/1976");

```

このスクリプトの出力は、次のようになります。

```

It has been 718329600 seconds since 4/11/1976
It has been 11972160 minutes since 4/11/1976
It has been 199536 hours since 4/11/1976
It has been 8314 days since 4/11/1976
It has been 22.7780821917808 years since 4/11/1976

```

次に日付を加算する例を示します。

```

/* Example of date arithmetic. */
/* Create a date object containing the current date. */
var d1 = new Date();
/* num contains the numeric representation of the current date. */
var num = d1.valueOf();
/* Add thirteen days to today's date. */
/* 1000 ms / sec; 60 sec / min; 60 min / hour; 24 hours / day; 13 days */
num += 1000 * 60 * 60 * 24 * 13;
/* Create our new date that is 13 days ahead of the current date. */
var d2 = new Date(num);
/* Print out the current date and our new date using util.printd */
console.println("It is currently: " + util.printd("mm/dd/yyyy", d1));
console.println("In 13 days, it will be: " + util.printd("mm/dd/yyyy", d2));

```

このスクリプトの出力は、次のようになります。

```

It is currently: 01/15/1999
In 13 days, it will be: 01/28/1999

```


どうすればドキュメントを保護できますか？

Acrobat が持つセキュリティ機能としては、まずドキュメントに対するアクセス制限、ドキュメントが開かれた後はフォームに対する権限の制限、そして電子署名があります。

ドキュメントに対するアクセス制限

フォームへのアクセスを完全に制限したい場合、Acrobat の標準セキュリティモデルを使用して、ユーザにパスワード入力を義務付けることができます。サードパーティからプラグインとして提供されているセキュリティハンドラの中には、便利なものもあります。例えば、公開／秘密鍵のインフラストラクチャを使用して、特定のユーザに対してフォームをロックしたり、一定期間が経過した時点でフォームを無効にすることもできます。

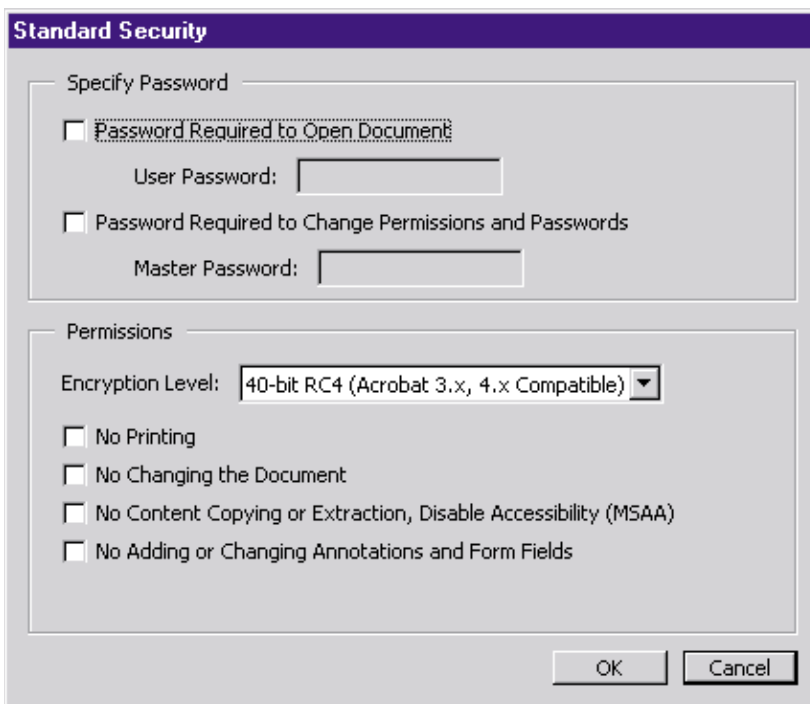
ユーザパスワードを設定するには、Acrobat メニューのファイル／文書のセキュリティを選択し、「セキュリティオプション」というタイトルのドロップダウンメニューから *Acrobat 標準セキュリティ* などの適切なセキュリティハンドラを選択します。するとパスワードおよび権限を、希望通り設定することができるようになります。

権限の制限

Acrobat の標準セキュリティモデルへのアクセスはドキュメントの保存時に行われ、このときドキュメントに対する制限が設定されます。制限を設定できるものには、ドキュメントの印刷、ドキュメントの変更、テキストおよびグラフィックの選択、注釈およびフォームフィールドの追加 / 変更があります。

フォームを作成した後は、データの入力はできてもフォームツールで改ざんされないようにするため、フォームをロックしておくのが便利です。例えば、フォームの作成後に Web サイトで公開するような場合がそうです。フォームの信頼性を保つには、フォーム内の式やデータ関数に変更されないように保護する必要があります。

文書の変更を許可しないというオプションを選択すると、ユーザはフォームにデータを入力したり注釈を追加したりすることはできませんが、フォームフィールドの作成や変更をしたり、TouchUp プラグインで背景のテキストを変更したりすることはできなくなります。



また、フォームにデータを入力し終えた後は、ドキュメントを変更できないようにするため、ドキュメント全体をロックするのが望ましい場合もあります。税金申告や機密文書にデータを入力した場合は、ドキュメントを保存した後で他の変更を加えることができないようにすべきです。データの入力と作成を禁止するには、文書の変更を許可しないおよび注釈とフォームフィールドの追加や変更を許可しないというオプションを選択してください。

電子署名

電子署名フォームフィールドは、アクセスや権限を制限することはしませんが、フォーム作成者またはユーザは、ドキュメントに署名が付けられた後で変更が加えられていないかを検証することができます。

フォーム作成者はデータ入力用のフォームをリリースする前に、フォームに電子署名を付けることができます。ユーザは、フォームが改ざんされておらず「公式」なものであることを署名によって確認できます。隠し署名（表示されない署名）はこのような場合に便利で、電子署名メニューのサブメニューから作成することができます。

ユーザはフォームにデータを入力した後、電子署名ツールまたは事前に用意されている署名フィールドを使用してドキュメントに署名を付けることができるので、知らない間にフォームが変更されるのを防ぐことができます。

電子署名フィールドをプログラムから作成および署名する方法について詳しくは、署名フィールドの説明も参照してください。

署名フィールドに署名した後、どうすればドキュメントをロックできますか？

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

署名フィールドを使用すると、ドキュメントに電子署名を付けることができます。ドキュメントに署名を付けた後でドキュメントに変更を加えると、「署名後にドキュメントが変更された」ことが署名に示されます。

署名フィールドの値は読み取り専用で、署名されていなければヌル値、署名した場合は非ヌル値になります。

署名フィールドに署名がされたときに動作させるカスタムスクリプトを作成する場合、フォームがリセットされる（フィールドの値がヌル値に設定されるなど）可能性も必ず考慮してください。例えば、署名フィールドに署名がされた瞬間、名前が「A」で始まるすべてのフィールドをロックし、「B」で始まるすべてのフィールドをアンロックしたいとします。この場合、次のようなスクリプトが考えられます。

```
/* This example is incorrect. */
var f = this.getField("A");
/* Lock all A fields. */
f.readonly = true;
f = this.getField("B");
/* Unlock all B fields. */
f.readonly = false;
```

これは多くのフォームで使用可能な汎用的な処理ですが、このスクリプトには**間違い**があります。これではフォームがリセットされたときに、間違ってフィールドをロックおよびアンロックしてしまいます。次のように署名イベントの値をチェックし、それに応じて対処するようにしてください。

```
var bLock = (event.value != "");
var f = this.getField("A");
/* Lock A on sign, unlock on reset. */
f.readonly = bLock;
f = this.getField("B");
/* Unlock B on sign, lock on reset. */
f.readonly = !bLock;
```

AForm.js には AFSignatureLock() という便利な関数があり ([JavaScript はどこにあり、どのように使用するのですか？](#)を参照)、署名のプロパティダイアログで行う単純なロック操作を、プログラムから実行できます。この関数を使用すると、全フィールド、特定のフィールド、あるいは特定のフィールド以外の全フィールドを簡単にロックおよびアンロックできます。以下はこの関数を使用して書き直した例です。

```
var bLock = (event.value != "");
AFSignatureLock(this, "THESE", "A", bLock);
AFSignatureLock(this, "THESE", "B", !bLock);
```

詳しくは、AForm.js のコメントを参照してください。

どうすればドキュメントへのアクセスを簡単にできますか？

4.05			
------	--	--	--

電子情報のアクセシビリティは、重要性がますます高まっている問題です。アクセシビリティに関する以下のヒントに従ってフォームを作成すれば、すべてのユーザにとってより使いやすいフォームになるでしょう。Acrobat のリリース 4.05 およびそれ以降では、身体や視覚に障害のあるユーザでも Acrobat Forms に入力できるよう設計されています。Acrobat の今後のバージョンでは、音声認識機能がさらに充実する予定です。

Acrobat 4.05 およびそれ以降で、標準機能のまま、最小限の手順でフォームにアクセスできるようにするには、次のガイドラインに従ってください。

ドキュメントのメタデータ

ファイル／文書のプロパティ／概要ダイアログを使用して、ドキュメントのメタデータを設定できます。

ドキュメントのオープン、保存、印刷、クローズといった操作を行うと、ドキュメントのタイトルがユーザに告げられます。タイトルが指定されていない場合はファイル名が告げられますが、ファイル名は短縮されたり変更されたりすることがよくあるので、ドキュメントの作成者はタイトルを設定するようにしてください。例えば、「IRS1040.pdf」というファイル名では意味が分かりません。この場合、「Form 1040: U.S. Individual Income Tax Return for 1998」というようにタイトルを付けるのが良いでしょう。

ドキュメントに関連するすべてのメタデータ（作成者、サブタイトル、キーワード）を入力すると、Acrobat Search やインターネット検索エンジンを使用してドキュメントを簡単に検索できるようになります。

フィールドの説明

ユーザに操作されるフィールド（読み取り専用フィールドでも隠しフィールドでない場合）であるにもかかわらず、フィールド名の意味が分かりにくい場合、そのフィールドに簡単な説明を含めるべきです。この説明はユーザがフィールドのフォーカスを取得したときに告げられるので、フィールドの目的を表すものでなければなりません。例えば、フィールド名が「name.first」であれば、「First Name」というような説明が適しています。この説明は、ユーザがマウスポインタをフィールドに置いたときにも「ツールヒント」として表示されるので、フォームがさらに使いやすくなります。

タブ順序の設定

ドキュメントの内容を効率的にたどっていくには、フィールドのタブ順序が論理的に設定されている必要があります。ほとんどのユーザはタブを使用してドキュメント内を移動するので、その順序は特に重要です。視覚障害のあるユーザの場合は、マウスポインタを目的的位置に移動させたり視覚的な合図を確認したりできないので、これは必須条件となります。

TTS オブジェクトの使用

フォームフィールドの現在の状態に関する説明や指示を Acrobat の標準機能で十分に伝えることができない場合、[TTS オブジェクト](#)を使用してそれを補うことができます。

標準の動作

アクセシビリティに関する Acrobat 4.05 およびそれ以降の標準の動作は、次の通りです。

1. Tab キー：キーボードフォーカスがフォームフィールドに存在しないとき Tab (Shift-Tab) キーを押すと、現在のページでタブオーダーの最初（最後）にあるフィールドがアクティブになります。ページにフォームフィールドが存在しない場合、このことが音声によって通知されます。

フィールドにフォーカスがあるときに Tab (Shift-Tab) キーを押すと、タブオーダーに従って次（前）のフィールドに移動します。ページの最後（最初）のフィールドで Tab (Shift-Tab) キーを押すと、次（前）のページ（存在する場合）の最初（最後）のフィールドにフォーカスが移ります。次（前）のフィールドが存在しない場合、フォーカスは現在のページで「ループ」して、最初（最後）のフィールドに移動します。

2. サウンドによる通知：サウンドはユーザによる設定が可能で、特定のイベントが発生したときに再生され、ユーザに状況を通知します。次のアクションではサウンドの再生が可能です。

- ドキュメントのオープン、クローズ、アクティブ化、保存、印刷
 - ページの移動
 - フィールドでのキー入力
3. 音声による通知：音声の内容はユーザによる設定が可能で、テキストを音声エンジンに与えることで、障害のあるユーザに音声で状況を知らせることができます。次のアクションでは音声による通知が可能です。
- アプリケーションの初期化
 - ドキュメントのオープン、クローズ、アクティブ化、保存、印刷
 - ページの移動
 - フィールドへのフォーカスの移動、フォーカスの消失
 - 警告ダイアログ
 - フィールドでのキー入力

どうすれば JavaScript でグローバル変数を定義できますか？

Acrobat には JavaScript 用の拡張機能として Global オブジェクトが定義されており、これにグローバル変数をプロパティとして格納することができます。「myVariable」という新しいグローバル変数を定義し、それにヌル文字列を代入するには、次のようにします。

```
global.myVariable = "";
```

ドキュメント内のどのスクリプトからでもこの変数を参照することができます。

グローバル変数の永続化

通常、グローバル変数のグローバルデータは、ユーザセッション間では維持されませんが、Global オブジェクトに備わっているメソッドを使うと、セッション間でデータを維持することができます。「myVariable」という変数をセッション間で維持するには、次のようにします。

```
global.setPersistent("myVariable", true);
```

このようにすると、以降のセッションでも引き続き変数は存在するようになります（前の値は保持されます）。

どうすればフォームデータを電子メールアドレスに送信できますか？

これには、[Field オブジェクト](#)の [submitForm](#) メソッドを使用します。

```
var url = "mailto:johndoe@doe.net";  
this.submitForm(url, false);
```


この場合、フォームの内容は変数 `url` が示すアドレスに送信されます。`submitForm()` の 2 つ目の引数は、フォーム内容を URL エンコードしたデータとして POST メソッドで送信するのか、あるいは FDF (Forms Data Format) として送信するのかを示します。「false」は URL エンコード形式で送信することを示します。

どうすれば他のフィールドの値に応じてフィールドを非表示にできますか？

[Field オブジェクト](#) の [display](#) プロパティを使用します。

```
var title = this.getField("title");
if (this.getField("showTitle").value == "Off")
    title.display = display.hidden;
else
    title.display = display.visible;
```

どうすれば別に関われているフォームのフィールド値を現在のフォームから参照できますか？

[Global オブジェクト](#) の [subscribe](#) メソッドを使用して、目的のフィールドを他のフィールドから参照することができます。例えば、目的のフィールドのグローバル値を設定する文書レベルのスクリプト (ドキュメントを初めて開いたときに起動される) をフォーム (ドキュメント B) に設定します。

```
function PublishValue( xyzForm_fieldValue_of_interest ) {
    global.xyz_value = xyzForm_fieldValue_of_interest;
}
```

そして、現在のドキュメント (ドキュメント A) からドキュメント B の目的の値にアクセスするには、その変数を subscribe します。

```
global.subscribe("xyz_value", ValueUpdate);
```

`ValueUpdate` は、`xyz_value` の内容が変更されるたびに自動的に呼び出されるユーザ定義関数です。ドキュメント A の `MyField` というフィールドで `xyz_value` の値を使用している場合、コールバック関数を次のように定義します。

```
function ValueUpdate( newValue ) {
    getField("MyField").value = newValue;}

```

どうすればキー入力を 1 つずつインターセプトできますか？

カスタムキーストロークスクリプトを作成し (テキストフィールドまたはコンボボックスフィールドで、「フィールドのプロパティ」ダイアログの「フォーマット」タブを参照)、`event.change` の値を調べます。この値を変更すると、ユーザ入力の内容を入力時に変更することができます。

どうすればネストされたポップアップメニューを作成できますか？

`app.popUpMenu()` メソッドを使用します。メニュー選択の配列を作成し、フィールドの「マウスボタンを押す」または「マウスボタンを放す」イベントから `app.popUpMenu(arrayName)` を呼び出し、メニューをポップアップさせます。

例：

```
var cChoice = app.popUpMenu("one", "two", "-",  
    [ "three", "three.one", "three.two" ] );  
app.alert("You chose " + cChoice);
```

どうすれば独自の色を作成できますか？

色は配列オブジェクトであり、配列の最初の項目はカラースペースを表す文字列（「G」はグレースケール、「RGB」は RGB、「CMYK」は CMYK）で、その後の項目はそれぞれのカラースペースのコンポーネントを表す数値です。したがって、次のようになります。

```
color.blue = new Array("RGB", 0, 0, 1);  
color.cyan = new Array("CMYK", 1, 0, 0, 0);
```

カスタムカラーを作成するには、使用するカラースペースのタイプとチャンネル値の配列を宣言するだけです。

どうすればユーザ入力を促すダイアログを表示できますか？

[App オブジェクト](#) クラスに定義された [response](#) メソッドを使用します。このメソッドで表示されるダイアログには、質問と、それに応答するための入力フィールドが含まれます（オプションとして、ダイアログタイトルや、応答のデフォルト値を表示することもできます）。戻り値はユーザの応答を含む文字列です。ユーザがダイアログの「キャンセル」ボタンをクリックすると、応答は null オブジェクトになります。

```
var dialogTitle = "Please Confirm";  
var defaultAnswer = "No.";  
var reply = app.response("Did you really mean to type that?",  
    dialogTitle, defaultAnswer);
```

どうすれば JavaScript で URL を取り出せますか？

[Data オブジェクト](#) クラスの [getURL](#) メソッドを使用します。このメソッドは、GET を使用して指定の URL をインターネットから取り出します。現在のドキュメントをブラウザ内で表示しているか、Acrobat Web Capture が使用できない場合は、Weblink プラグインを使用して目的の URL を取り出します。

どうすればフィールドのホットヘルプテキストを動的に変更できますか？

[Field オブジェクト](#)の [userName](#) プロパティを使用して、フィールドの「説明」を文字列として取得 / 設定できます。このプロパティは、マウスカーソルをフィールドに置いたときにツールヒントまたはホットヘルプを表示するためにあります。便利なアドバイス（あるいは何らかの説明）を [userName](#) プロパティに設定すると非常に効果的です。

どうすればズーム倍率をプログラムで変更できますか？

[Data オブジェクト](#)クラスの [zoom](#) メソッドを使用します。例えば、次のコードでは、現在のページのズーム倍率を 2 通りの方法で設定します。

```
// This zooms in to twice the current zoom level:
this.zoom *= 2;

// This sets the zoom to 73%:
this.zoom = 73;
```

どうすればマウスが特定の領域に入った（あるいは出た）かを判断できますか？

マウスの出入りを調べたい場所に、非表示の読み取り専用テキストフィールドを作成します。次に、フィールドの「ポインタを範囲内に入れる」および「ポインタを範囲外に出す」アクションに JavaScript を割り当てます。

回転ユーザスペースおよびデフォルトユーザスペースとは何ですか？

このマニュアルでは、*回転ユーザスペース*および*デフォルトユーザスペース*という 2 つの用語を頻繁に使用します。これらは、それぞれフォームと注釈に関係しています。フォームでは常に回転ユーザスペースを使用し、注釈では常にデフォルトユーザスペースを使用します。

回転ユーザスペース

（座標系である）回転ユーザスペースでは、ページを回転しても原点（または参照点）はクロップボックスの左下隅になります。原点の右が正の X 軸で、原点の上が正の Y 軸です。X 軸および Y 軸の 1 単位の長さは 1/72 インチです。

デフォルトユーザスペース

デフォルトユーザスペースとは、『[PDF Reference](#)』の 126 ページで記載している座標系のことです。回転していないページの場合、原点はメディアボックスの左下隅になり、正の X 軸は右に水平に延び、正の Y 軸は上に垂直に延びます。X 軸および Y 軸の 1 単位の長さは 1/72 インチです。しかしページを回転すると座標系も回転するので、X 軸と Y 軸の方向が上記の標準のものと異なることがあります。

座標の関係

以下の図は、トリミングしてから時計回りに 90 度回転したページを示しています。回転ユーザスペースでは、原点はクロップボックスの左下隅であり、軸の方向は標準座標と変わりません。ページを時計回りに 90 度回転したことは記憶されており、デフォルトユーザスペースの原点は左上隅となり、正の X 軸は下を指し、正の Y 軸は右を指すようになります。

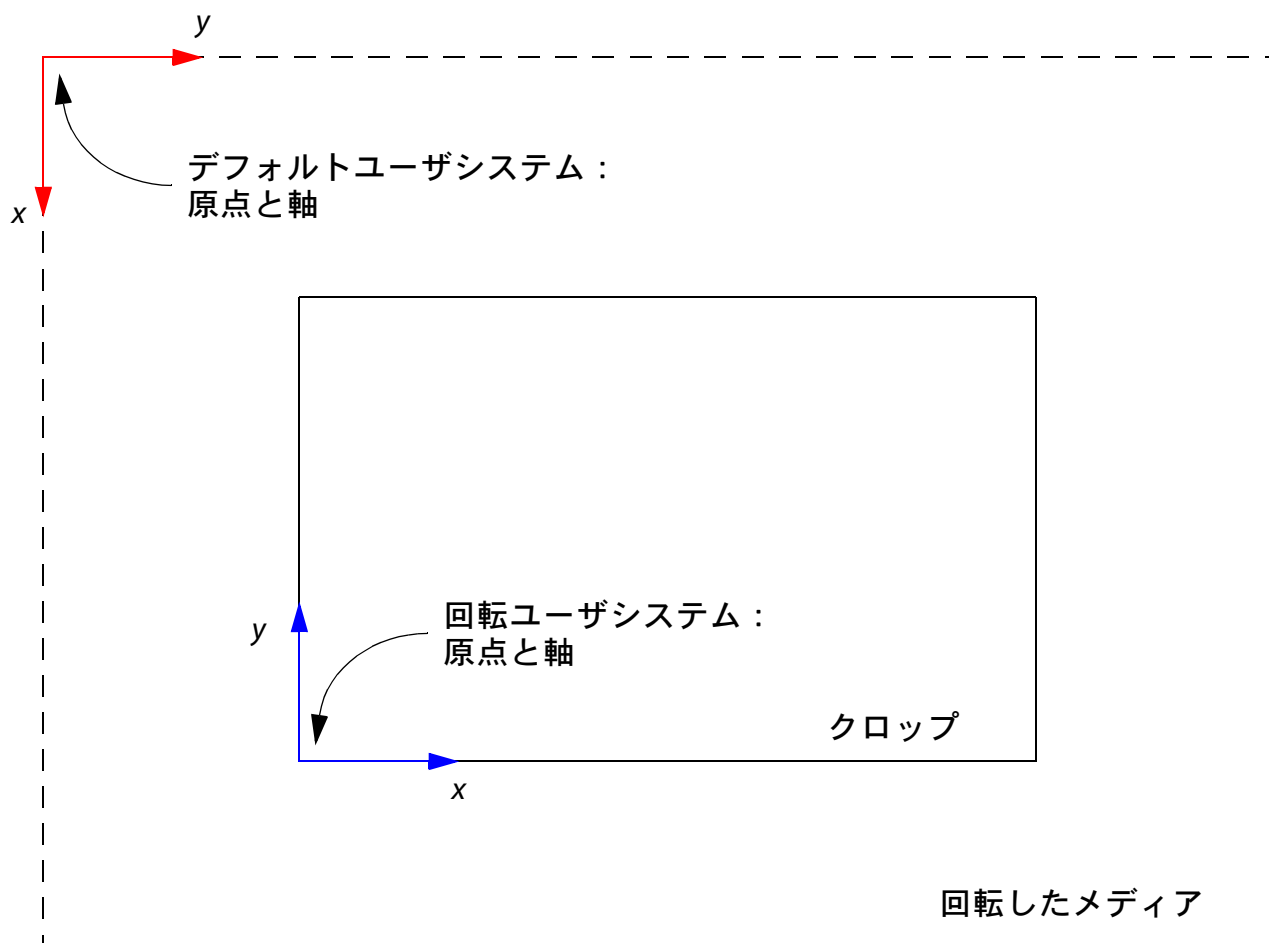
(その境界を示す矩形が) 同じ座標を持つ注釈およびフォーム要素が、必ずしもページ上の同じ位置に配置されるとは限りません。例えば、時計回りに 90 度回転された先頭ページに対して、以下を実行するとします。

```
this.addField("myButton", "button", 0, [0,0,20,20] )
```

その後、次のスクリプトを実行します。

```
this.addAnnot({ page: 0, type: "Square", rect: [0,0,20,20] });
```

この結果、ボタンはクロップボックスの左下隅に配置され、「Square」タイプの注釈はメディアボックスの左上隅に配置されます。フォームオブジェクトや注釈オブジェクトをページに配置する場合は、常にこのことを念頭に置いてください。



例：次のコードでは、まずボタンフィールドを画面ページ（トリミングされているページ）の左下隅に配置し、次に「Square」タイプの注釈をまったく同じ場所に配置しています。このスクリプトは、ページがトリミングされたり回転されたりしていても、正しく動作します。このスクリプトでは、回転ユーザスペースをデフォルトユーザスペースに変換するため、マニュアルに記載されていない JavaScript メソッドを使用しています。

```
var f = this.addField("myButton", "button", 0, [36,36,72,72] );
f.strokeColor = color.black;
var m = (new Matrix2D).fromRotated(this,0);
r = m.transform(f.rect);
this.addAnnot( { type:"Square", rect: r, page: 0 } );
```

どうすればフォームフィールドをプログラムで作成できますか？

Acrobat にはフォームフィールドを作成するための JavaScript プロパティおよびメソッドが数多く用意されており、フォームフィールドの外観および関連するアクションを設定することもできます。ここでは、新規（および従来）のプロパティとメソッドをすべて整理し、Acrobat のユーザインタフェース項目に対応する JavaScript プロパティおよびメソッドを解説します。

フォームフィールドには次の 7 種類があります。

- ・ [ボタン](#)
- ・ [ラジオボタン](#)
- ・ [チェックボックス](#)
- ・ [署名](#)
- ・ [コンボボックス](#)
- ・ [テキスト](#)
- ・ [リストボックス](#)

以上について順に説明していきます。

ボタン

フォームフィールドの作成には、Acrobat の UI または [Data オブジェクト](#) の [addField](#) メソッドを使用できます。ボタンフィールドをプログラムで作成するには、次のようにします。

```
var f = this.addField("myButton", "button", 0, [400, 436, 472, 400]);
```

このようにすると、ページ 0 にボタンが作成され、[400, 436, 472, 400] に配置されます。ボタンの幅は 1 インチ（72 ポイント）、高さは 0.5 インチ（36 ポイント）で、フィールドにはデフォルトの外観が与えられます。このメソッドの戻り値は Field オブジェクトであり、この節全体を通して使用されます。

既存のフィールドの Field オブジェクトは、Doc オブジェクトの [getField](#) メソッドを使用して取得することができます。

```
var f = this.getField("myButton");
```

フィールドのプロパティ

UIの一番上(タブパレットの上)には、フィールドの概要を示す3つのプロパティがあります。

フィールドのプロパティの概要		
UI の名前	使用するメソッド / プロパティ	例
名前	name	console.println(f.name);
種類	type	console.println(f.type);
説明	userName	f.userName = "Submit Button"

表の注意事項

- ・ **名前**および**種類**は読み取り専用のプロパティで、UIでフィールドを作成するときに設定するか、[addField](#) を使用してプログラムから設定します。前述の例を参照してください。

Acrobat のボタンフィールドの UI には、フィールドの表示方法、オプション、およびアクションを設定するタブがあり、これらのプロパティにはすべてフィールドレベルの JavaScript プロパティおよびメソッドからアクセスすることができます。

表示方法

「表示方法」タブでは、フィールドの基本的な外観を設定できます。下の表に、JavaScript を使用して外観をプログラムから設定する方法をまとめます。この表では、変数 *f* を Field オブジェクトとしています。

「表示方法」タブ		
カテゴリ	使用するメソッド / プロパティ	例
境界線		
境界線の色 背景色 幅 スタイル	strokeColor fillColor lineWidth borderStyle	f.strokeColor = color.black; f.fillColor = color.ltGray; f.lineWidth = 1; f.borderStyle = border.b
テキスト		
文字の色 フォント サイズ	textColor textFont textSize	f.textColor = color.blue; f.textFont = font.Times; f.textSize = 16;
一般プロパティ		
読み取り専用 表示と印刷 向き	readonly display	f.readonly = true; f.display = display.visible 表の注意事項を参照

表の注意事項

- ・ フォームフィールドの方向を設定するプロパティはありませんが、ページを回転してからボタンを作成し、またページの回転を戻すことで、回転したボタンを作成することができます。

```
this.setPageRotations(this.pageNum, this.pageNum, 90);  
var f = this.addField("actionField", "button", 0, [200, 250, 300, 200]);  
f.delay = true;  
f.strokeColor = color.black;  
f.fillColor = color.ltGray;  
f.borderStyle = border.b;  
f.delay=false;  
this.setPageRotations(this.pageNum, this.pageNum);
```

オプション

「オプション」タブでは、強調表示、レイアウト、ボタン表示の外観を設定できます。

「オプション」タブ		
強調表示	highlight	f.hightlight = hightlight.p
レイアウト	buttonPosition	f.buttonPosition = position.iconOnly;
レイアウトの詳細設定		
条件 変更方法 ボタン	buttonScaleWhen buttonScaleHow buttonAlignX および buttonAlignY	f.buttonScaleWhen = scaleHow.always f.buttonScaleHow = scaleHow.proportional f.buttonAlign = 50; f.buttonAlign = 50;
ボタンの属性		
テキスト	buttonSetCaption および	f.buttonSetCaption("Push Me");
アイコンの選択	buttonGetCaption buttonSetIcon	表の注意事項を参照

表の注意事項

- ・ 「ボタンの属性」の「アイコンの選択」に対応する基本メソッドは [buttonSetIcon](#) です。このメソッドはアイコンをボタンの表面に関連付けますが、名前付きアイコンがあらかじめ PDF ファイルに含まれている必要があります。ボタンの表面にアイコンに関連付ける完全な例については、Doc オブジェクトの [addIcon](#) を参照してください。
- ・ 名前付きアイコンをドキュメントに含めるには、[buttonImportIcon](#) を使用します。

アクション

ボタンフィールドのアクションを設定するには、フィールドレベルの [setAction](#) メソッドに、「MouseUp」、「MouseDown」、「MouseEnter」、「MouseExit」、「OnFocus」、「OnBlur」というトリガー名を指定します。次に例を示します。

```
f.setAction("MouseUp", "app.beep(0);")
```

チェックボックス

フォームフィールドの作成には、Acrobat の UI または [Data オブジェクト](#) の [addField](#) メソッドを使用できます。チェックボックスフィールドをプログラムで作成するには、次のようにします。

```
var f = this.addField("myCheck", "checkbox", 0, [400, 412, 412, 400]);
```

このようにすると、ページ 0 にチェックボックスが作成され、[400, 412, 412, 400] に配置されます。チェックボックスの幅および高さはどちらも 12 ポイントになり、フィールドにはデフォルトの外観が与えられます。このメソッドの戻り値は Field オブジェクトであり、この節全体を通して使用されます。

既存のフィールドの Field オブジェクトは、Doc オブジェクトの [getField](#) メソッドを使用して取得することができます。

```
var f = this.getField("myCheck");
```

フィールドのプロパティ

UI の一番上（タブパレットの上）には、フィールドの概要を示す 3 つのプロパティがあります。プロパティはボタンの場合と同じです。ボタンの [フィールドのプロパティ](#) を参照してください。

表示方法

[表示方法](#) の属性はボタンの場合と同じですが、異なる点が 2 つあります。まず、チェックボックスの場合は `textFont` プロパティに選択肢がなく、フォントは常に *Zapf Dingbats* になります。そして「表示方法」タブにある **一般プロパティ** の必須チェックボックスが使用可能になります。

「表示方法」タブ		
カテゴリ	使用するメソッド / プロパティ	例
一般プロパティ		
必須	required	f.required = true;

オプション

「オプション」タブではフィールドで使用するチェックマークのスタイルや、書き出し値を設定することができます。次の表では、UI と JavaScript の対応を示しています。

「オプション」タブ		
チェックのスタイル	style	f.style = style.ci;
書き出し値	exportValues	f.setExportValues(["buy"]);
デフォルトでチェックする	defaultIsChecked	f.defaultIsChecked(0); // see table notes

表の注意事項

- デフォルトでチェックする：[defaultIsChecked](#) を使用しただけでは、フィールドは必ずしもチェックされません。フィールドをチェックするには、`this.resetForm([f.name])` を使用してフィールドをリセットするか、[checkThisBox](#) メソッドで `f.checkThisBox(0)` ; というようにします。
- デフォルトでチェックする：「オプション」タブの「デフォルトでチェックする」チェックボックスがチェックされているかどうかを判断するには、[isDefaultChecked](#) を使用します。
- チェックボックスフィールドがチェックされているかどうかを判断するには、[isBoxChecked](#) を使用します。

アクション

チェックボックスフィールドのアクションを設定するには、フィールドレベルの [setAction](#) メソッドに、「MouseUp」、「MouseDown」、「MouseEnter」、「MouseExit」、「OnFocus」、「OnBlur」というトリガー名を指定します。次に例を示します。

```
f.setAction("MouseUp", "app.beep(0);")
```

コンボボックス

フォームフィールドの作成には、Acrobat の UI または [Doc オブジェクト](#) の [addField](#) メソッドを使用できます。コンボボックスフィールドをプログラムで作成するには、次のようにします。

```
var f = this.addField("myCombo", "combobox", 0, [200, 436, 400, 400]);
```

このようにすると、ページ 0 にコンボボックスが作成され、[200, 436, 400, 400] に配置されます。コンボボックスの幅は 200 ポイント、高さは 36 ポイントになります。フィールドには、デフォルトの外観が与えられます。このメソッドの戻り値は Field オブジェクトであり、この節全体を通して使用されます。

既存のフィールドの Field オブジェクトは、Doc オブジェクトの [getField](#) メソッドを使用して取得することができます。

```
var f = this.getField("myCombo");
```

フィールドのプロパティ

UI の一番上（タブパレットの上）には、フィールドの概要を示す 3 つのプロパティがあります。プロパティはボタンの場合と同じです。ボタンの [フィールドのプロパティ](#) を参照してください。

表示方法

「表示方法」タブでは、フィールドの基本的な外観を設定できます。下の表に、JavaScriptを使用して外観をプログラムから設定する方法をまとめます。この表では、変数 `f` を `Field` オブジェクトとしています。

「表示方法」タブ		
カテゴリ	使用するメソッド / プロパティ	例
境界線		
境界線の色 背景色 幅 スタイル	strokeColor fillColor lineWidth borderStyle	<code>f.strokeColor = color.black;</code> <code>f.fillColor = color.ltGray;</code> <code>f.lineWidth = 1;</code> <code>f.borderStyle = border.b</code>
テキスト		
文字の色 フォント サイズ	textColor textFont textSize	<code>f.textColor = color.blue;</code> <code>f.textFont = font.Times;</code> <code>f.textSize = 16;</code>
一般プロパティ		
読み取り専用 必須 表示と印刷 向き	readonly required display	<code>f.readonly = true;</code> <code>f.required = true;</code> <code>f.display = display.visible</code> 表の注意事項を参照

表の注意事項

- フィールドの方向は、ボタンフィールドに関する節の「[表の注意事項](#)」に記載した方法で再設定することができます。

オプション

「オプション」タブでは、項目名とそれに対応する書き出し値を設定できます。

「オプション」タブ		
項目	setItems	表の注意事項を参照
書き出し値	setItems	表の注意事項を参照
項目の並べ替え		表の注意事項を参照
編集可能	editable	<code>f.editable = true;</code>
スペルチェックしない	doNotSpellCheck	<code>f.doNotSpellCheck = true;</code>

表の注意事項

- 項目および書き出し値：[addField](#) でコンボボックスを作成後、[setItems](#) を使用して、項目名とその書き出し値を簡単に設定することができます。次に例を示します。

```
f.setItems([ ["California", "CA"], ["Ohio", "OH"], ["Arizona", "AZ"] ] );
```


- ・ソート:「オプション」タブのこのチェックボックスを直接フックすることはできません。UIで項目が追加された際にリストをソートするため、このチェックボックスはAcrobatで管理されています。上の [setItems](#) の例では項目はアルファベット順に入力されませんが、配列オブジェクトのソートメソッドを使用すると、リストをプログラムでソートすることができます。次に例を示します。

```
function compare (a,b) {                                // define a compare function
    if (a[0] < b[0] ) return -1;
    if (a[0] > b[0] ) return 1;
    return 0;
}
var tmp = new Array();
var f = this.getField("myCombo");
for (var i = 0; i < f.numItems; i++)    // load [item, exportvalue]
    tmp[i] = [f.getItemAt(i,false), f.getItemAt(i)];
tmp.sort(compare);                                // sort of first component
f.clearItems();                                    // out with the old
f.setItems(tmp);                                    // in with the new
```

アクション

コンボボックスフィールドのアクションを設定するには、フィールドレベルの [setAction](#) メソッドに、「MouseUp」、「MouseDown」、「MouseEnter」、「MouseExit」、「OnFocus」、「OnBlur」というトリガー名を指定します。次に例を示します。

```
f.setAction("MouseEnter", "app.beep(0);")
```

フォーマット

コンボボックスフィールドのアクションは、フィールドレベルの [setAction](#) メソッドに「Format」というトリガー名を指定して設定することもできます。UIにはいくつかのフォーマットのカテゴリがあります。次の表にJavaScriptとの対応を示します。カスタムを除くすべてのフォーマットは、*Aform.js*に含まれている関数で実現することができます。

「フォーマット」タブ		
数値	AFNumber_Format	f.setAction("Format", 'AFNumber_Format(2, 0, 0, 0, "¥240", true);
パーセント	AFPercent_Format	
日付	AFDate_FormatEx	
時間	AFTime_Format	
特殊	AFSpecial_Format	
カスタム		表の注意事項を参照

表の注意事項

- ・ 数値：表の例は、コンマで区切った小数点以下 2 桁のユーロ通貨を表しており、UI にも同じフォーマットがあります。
- ・ カスタム：上に示すフォーマット関数を使用しないフォーマットスクリプトは、すべてカスタムフォーマットスクリプトとして分類されます。カスタムキーストロークスクリプトを設定するには、[setAction](#) メソッドに「Keystroke」というトリガー名を指定します。

検証

コンボボックスフィールドのアクションは、フィールドレベルの [setAction](#) メソッドに「Validate」というトリガー名を指定して設定することもできます。UI にはいくつかの検証のカテゴリがあります。次の表に JavaScript との対応を示します。カスタムを除くすべての検証は、*Aform.js* に含まれている関数で実現することができます。

「検証」タブ		
値の範囲指定（最小値、最大値）	AFRange_Validate	f.setAction("Validate", 'AFRange_Validate(true, 0, true, 100)'); /* value between 0 and 100, inclusive */
カスタム		表の注意事項を参照

表の注意事項

- ・ カスタム：AFRange_Validate 関数を使用しない検証スクリプトは、すべてカスタムとして分類されます。

計算

コンボボックスフィールドのアクションは、フィールドレベルの [setAction](#) メソッドに「Calculate」というトリガー名を指定して設定することもできます。UI にはいくつかの計算のカテゴリがあります。次の表に、JavaScript との対応を示します。カスタムを除くすべての計算は、*Aform.js* に含まれている関数で実現することができます。

「計算」タブ		
次のフィールドの和（積、平均、最小、最大）の値を指定	AFSimple_Calculate	f.setAction("Calculate", 'AFSimple_Calculate("SUM", new Array ("line.1", "line.3"))');
カスタム		表の注意事項を参照

表の注意事項

- ・ カスタム：AFSimple_Calculate 関数を使用しない計算スクリプトは、すべてカスタムとして分類されます。

プログラミングに関するその他の注意

- ・ コンボ（またはリスト）ボックスにある項目数を取得するには、[numItems](#) プロパティを使用します。
- ・ 項目の表示名（項目名）や書き出し値を取得するには、[getItemAt](#) を使用します。
- ・ コンボ（またはリスト）ボックスに新規の項目を追加するには、[insertItemAt](#) を使用します。
- ・ コンボ（またはリスト）ボックスから項目を削除するには、[deleteItemAt](#) を使用します。
- ・ コンボ（またはリスト）ボックスから項目すべてを削除するには、[clearItems](#) を使用します。
- ・ コンボ（またはリスト）ボックスの現在値を変更するには、[currentValueIndices](#) を使用します。例えば、`f.currentValueIndices = 2` を指定すると、（0 ベースなので）3 つ目の項目がコンボボックスの現在値になります（フォームを送信した場合、その書き出し値がエクスポートされます）。

リストボックス

フォームフィールドの作成には、Acrobat の UI または [Doc オブジェクト](#) の [addField](#) メソッドを使用できます。リストボックスフィールドをプログラムで作成するには、次のようにします。

```
var f = this.addField("myList", "listbox", 0, [400, 445, 544, 400]);
```

このようにすると、ページ 0 にリストボックスが作成され、`[400, 445, 544, 400]` に配置されます。リストボックスの幅は 2 インチ（144 ポイント）で、12 ポイントの文字 3 行分の高さになります。フィールドには、デフォルトの外観が与えられます。このメソッドの戻り値は Field オブジェクトであり、この節全体を通して使用されます。

既存のフィールドの Field オブジェクトは、Doc オブジェクトの [getField](#) メソッドを使用して取得することができます。

```
var f = this.getField("myList");
```

フィールドのプロパティ

UI の一番上（タブパレットの上）には、フィールドの概要を示す 3 つのプロパティがあります。プロパティはボタンの場合と同じです。ボタンの [フィールドのプロパティ](#) を参照してください。

表示方法

[表示方法](#) はコンボボックスの場合と同じです。

オプション

[オプション](#) はコンボボックスの場合と殆ど同じですが、例外があります。

「オプション」タブ		
項目	setItems	表の注意事項を参照
書き出し値	setItems	表の注意事項を参照

「オプション」タブ		
項目の並べ替え		表の注意事項を参照
複数選択	multipleSelection	f.multipleSelection = true;

表の注意事項

- ・ 項目および書き出し値：コンボボックスの[表の注意事項](#)を参照してください。
- ・ 項目の並べ替え：コンボボックスの[表の注意事項](#)を参照してください。

アクション

リストボックスフィールドのアクションを設定するには、フィールドレベルの [setAction](#) メソッドに、「MouseUp」、「MouseDown」、「MouseEnter」、「MouseExit」、「OnFocus」、「OnBlur」というトリガー名を指定します。次に例を示します。

```
f.setAction("MouseUp", "app.beep(0);");
```

選択の変更

リストボックスフィールドのアクションは、フィールドレベルの [setAction](#) メソッドに「Keystroke」というトリガー名を指定して設定することもできます。UI にはいくつかの選択の変更のカテゴリがあります。カスタムを除くすべての選択の変更は、*Aform.js* に含まれている関数で実現することができます。

例：

```
f.setAction("Keystroke", "ProcessSelection();");
```

プログラミングに関するその他の注意

コンボボックスの[プログラミングに関するその他の注意](#)を参照してください。

ラジオボタン

フォームフィールドの作成には、Acrobat の UI または [Doc オブジェクト](#) の [addField](#) メソッドを使用できます。ラジオボタンフィールドをプログラムで作成するには、次のようにします。

```
this.addField("myRadio", "radiobutton", 0, [400, 442, 412, 430]);
this.addField("myRadio", "radiobutton", 0, [400, 427, 412, 415]);
var f = this.addField("myRadio", "radiobutton", 0, [400, 412, 412, 400]);
f.setExportValues(["Yes", "No", "Sometimes"]);
```

このようにすると、ページ 0 に 3 つのラジオボタンが作成されます。各ラジオボタンの幅は 12 ポイント、高さは 12 ポイントになり、フィールドにはデフォルトの外観が与えられます。このメソッドの戻り値は Field オブジェクトであり、この節全体を通して使用されます。各ボタンの書き出し値を定義するには、[exportValues](#) を使用します。

既存のフィールドの Field オブジェクトは、Doc オブジェクトの [getDataObject](#) メソッドを使用して取得することができます。

```
var f = this.getField("myRadio");
```

ラジオボタンの UI はチェックボックスとまったく同じです。ラジオボタンフィールドの外観の変更方法や、オプションおよびアクションの設定方法については、[チェックボックス](#)の節を参照してください。

フィールドのプロパティ

UI の一番上（タブパレットの上）には、フィールドの概要を示す 3 つのプロパティがあります。プロパティはボタンの場合と同じです。ボタンの[フィールドのプロパティ](#)を参照してください。

表示方法

[表示方法](#)はチェックボックスの場合と同じです。

オプション

ラジオボタンの[オプション](#)はチェックボックスの場合と同じです。[表の注意事項](#)も参照してください。

アクション

ラジオボタンフィールドのアクションを設定するには、フィールドレベルの [setAction](#) メソッドに、「MouseUp」、「MouseDown」、「MouseEnter」、「MouseExit」、「OnFocus」、「OnBlur」というトリガー名を指定します。次に例を示します。

```
f.setAction("MouseUp", "app.beep(0);")
```

署名

フォームフィールドの作成には、Acrobat の UI または [Doc オブジェクト](#)の [addField](#) メソッドを使用できます。署名フィールドをプログラムで作成するには、次のようにします。

```
var f = this.addField("mySignature", "signature", 0, [200, 500, 500, 400]);
```

このようにすると、ページ 0 に署名フィールドが作成され、[200, 500, 500, 400] に配置されます。署名フィールドの幅は 300 ポイント、高さは 100 ポイントになります。フィールドには、デフォルトの外観が与えられます。このメソッドの戻り値は Field オブジェクトであり、この節全体を通して使用されます。

既存のフィールドの Field オブジェクトは、[Doc オブジェクト](#)の [getField](#) メソッドを使用して取得することができます。

```
var f = this.getField("mySignature");
```

フィールドのプロパティ

UI の一番上（タブパレットの上）には、フィールドの概要を示す 3 つのプロパティがあります。プロパティはボタンの場合と同じです。ボタンの [フィールドのプロパティ](#) を参照してください。

表示方法

[表示方法](#) はコンボボックスの場合と同じです。

アクション

署名フィールドのアクションを設定するには、フィールドレベルの [setAction](#) メソッドに、「MouseUp」、「MouseDown」、「MouseEnter」、「MouseExit」、「OnFocus」、「OnBlur」というトリガー名を指定します。次に例を示します。

```
f.setAction("MouseUp", "app.beep(0);")
```

署名

署名フィールドのアクションは、フィールドレベルの [setAction](#) メソッドに「Format」というトリガー名を指定して設定することもできます。UI にはいくつかの署名のカテゴリがあります。次の表に JavaScript との対応を示します。カスタムを除くすべての署名は、*Aform.js* に含まれている関数で実現することができます。

「署名」タブ		
読み取り専用にする：すべてのフィールド（選択したフィールドのみ、選択したフィールド以外）	AFSignature_Format	f.setAction("Format", 'AFSignature_Format("THESE", new Array ("mySignature"));');
カスタム		表の注意事項を参照

表の注意事項

- ・ カスタム：AFSignature_Format を使用しないスクリプトは、すべてカスタムとして分類されます。

例

以下は、JavaScript を使用して署名フィールドを作成、署名、ロックする完全な例です。

```
// Create signature field dynamically
var f = this.addField("mySignature", "signature", 0, [200, 500, 500, 400]);
f.strokeColor = color.black;

// set it to lock when signed
f.setAction("Format",
  'AFSignature_Format("THESE", new Array ("mySignature"));' );
```

```
var ppklite = security.getHandler("Adobe.PPKLite"); // choose handler
ppklite.login("dps017", "/C/signatures/DPSmith.apf"); // login
f.signatureSign(ppklite, // sign it
  { password: "dps017",
    location: "San Jose, CA",
    reason: "I am approving this document",
    contactInfo: "dpsmith@adobe.com",
    appearance: "Fancy"});
ppklite.logout(); // logout
```

テキスト

フォームフィールドの作成には、Acrobat の UI または [Doc オブジェクト](#) の [addField](#) メソッドを使用できます。テキストフィールドをプログラムで作成するには、次のようにします。

```
var f = this.addField("myText", "text", 0, [200,516,344,500])
```

このようにすると、ページ 0 にテキストフィールドが作成され、[200, 516, 344, 500] に配置されます。テキストフィールドの幅は 144 ポイント、高さは 16 ポイントになります。フィールドには、デフォルトの外観が与えられます。このメソッドの戻り値は Field オブジェクトであり、この節全体を通して使用されます。

既存のフィールドの Field オブジェクトは、[Doc オブジェクト](#) の [getField](#) メソッドを使用して取得することができます。

```
var f = this.getField("myText");
```

フィールドのプロパティ

UI の一番上（タブパレットの上）には、フィールドの概要を示す 3 つのプロパティがあります。プロパティはボタンの場合と同じです。ボタンの [フィールドのプロパティ](#) を参照してください。

表示方法

[表示方法](#) はコンボボックスの場合と同じです。

オプション

「オプション」タブでは、さまざまな属性を使用してデフォルトテキストを設定できます。

「オプション」タブ		
デフォルト	defaultValue	f.defaultValue = "Name: ";
整列	alignment	f.alignment = "center";
複数行	multiline	f.multiline = true;

「オプション」タブ		
最大文字数 # 文字	charLimit	f.charLimit = 40;
パスワード	password	f.password = true;
ファイルの選択に使用する	exportValues	f.fileSelect = false;
スペルチェックしない	doNotSpellCheck	f.doNotSpellCheck = true;

アクション

テキストフィールドのアクションを設定するには、フィールドレベルの [setAction](#) メソッドに、「MouseUp」、「MouseDown」、「MouseEnter」、「MouseExit」、「OnFocus」、「OnBlur」というトリガー名を指定します。次に例を示します。

```
f.setAction("MouseEnter", "app.beep(0);")
```

フォーマット

「[フォーマット](#)」タブはコンボボックスの場合と同じです。

検証

「[検証](#)」タブはコンボボックスの場合と同じです。

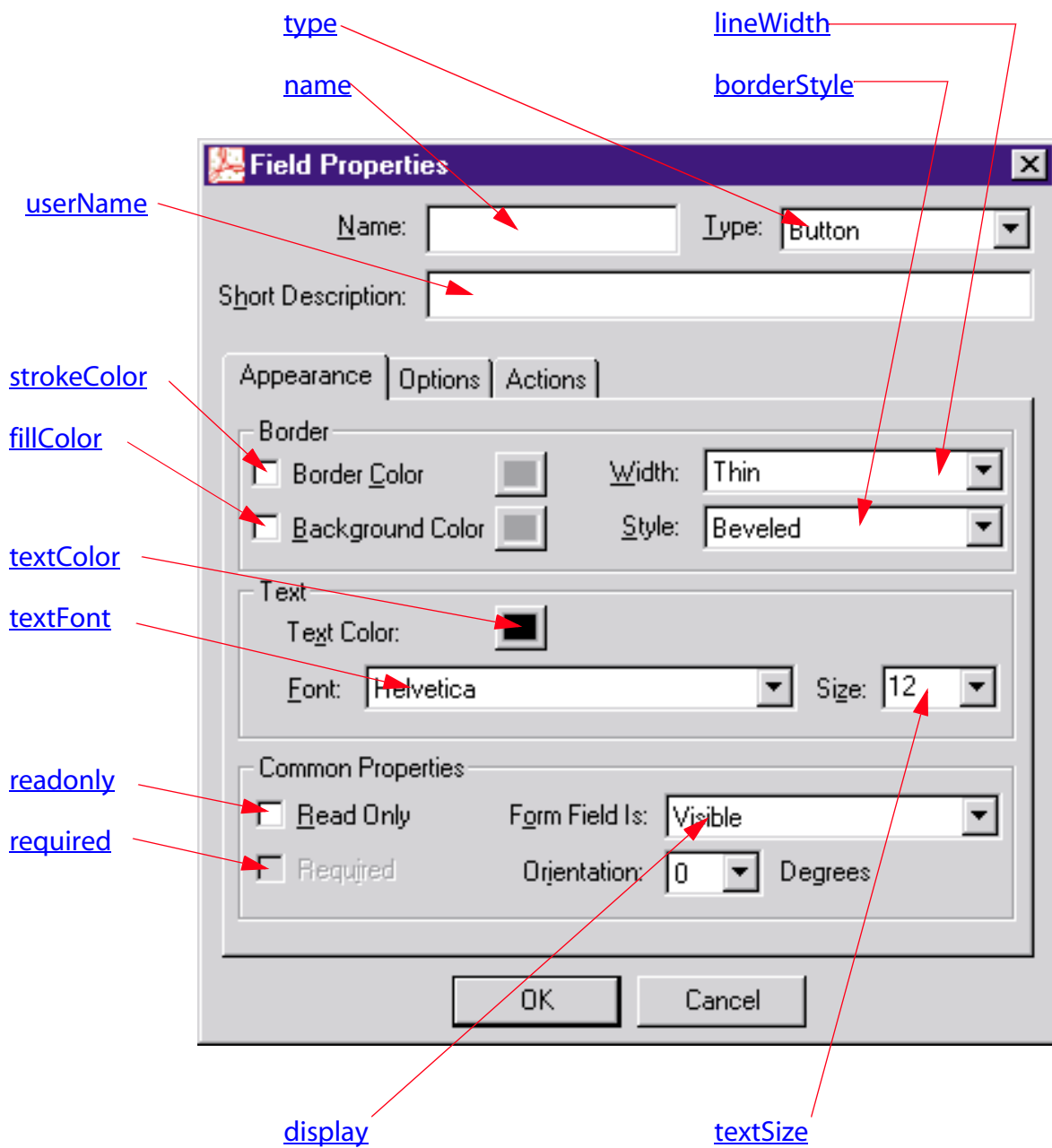
計算

「[計算](#)」タブはコンボボックスの場合と同じです。

クイックリファレンス：フォーム

表示方法：全フィールド

「表示方法」タブは、すべての種類のフィールドでほぼ同じです。

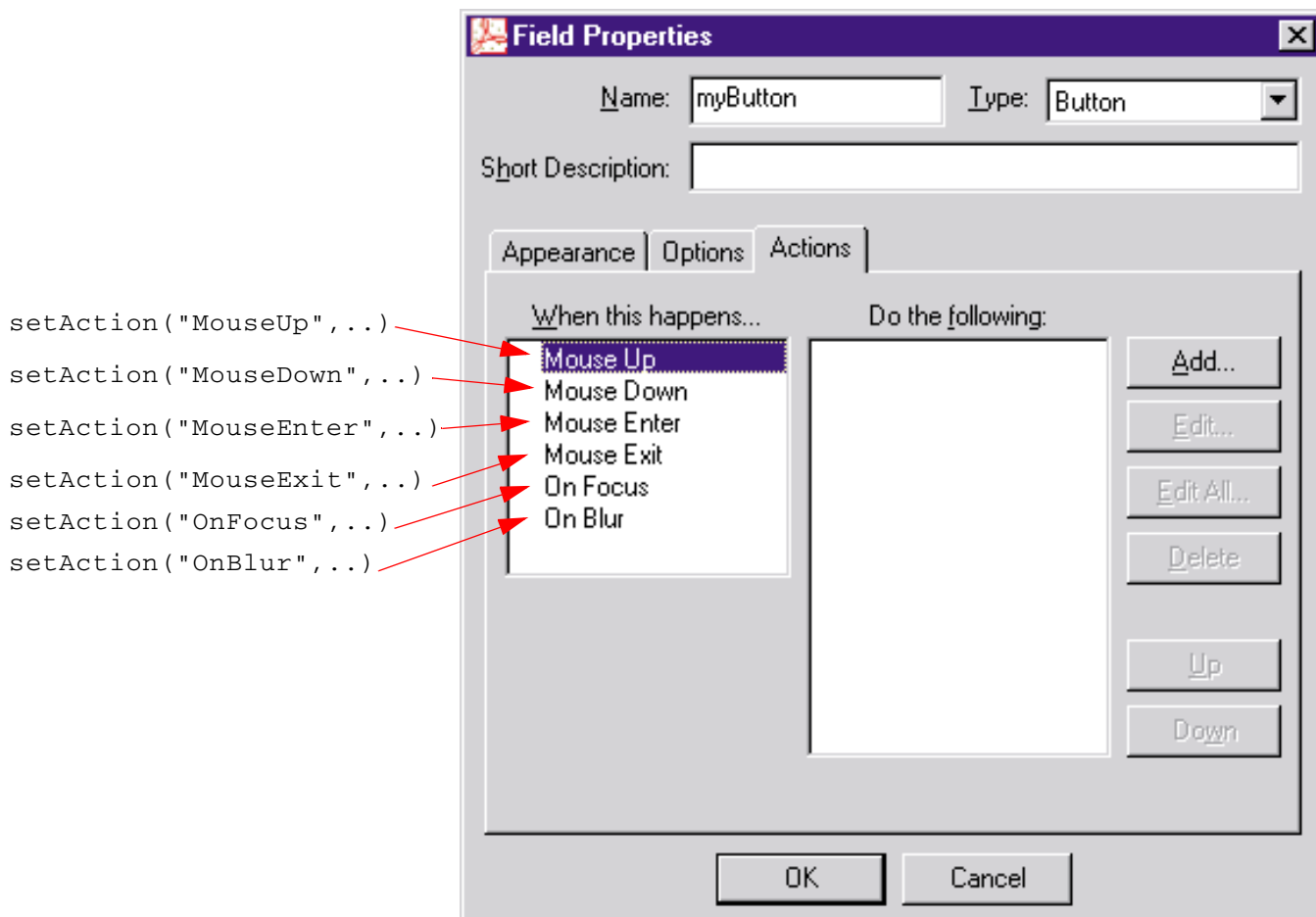


アクション：全フィールド

アクションを設定するには、次のようにフィールドレベルの [setAction](#) メソッドを使用することができます。

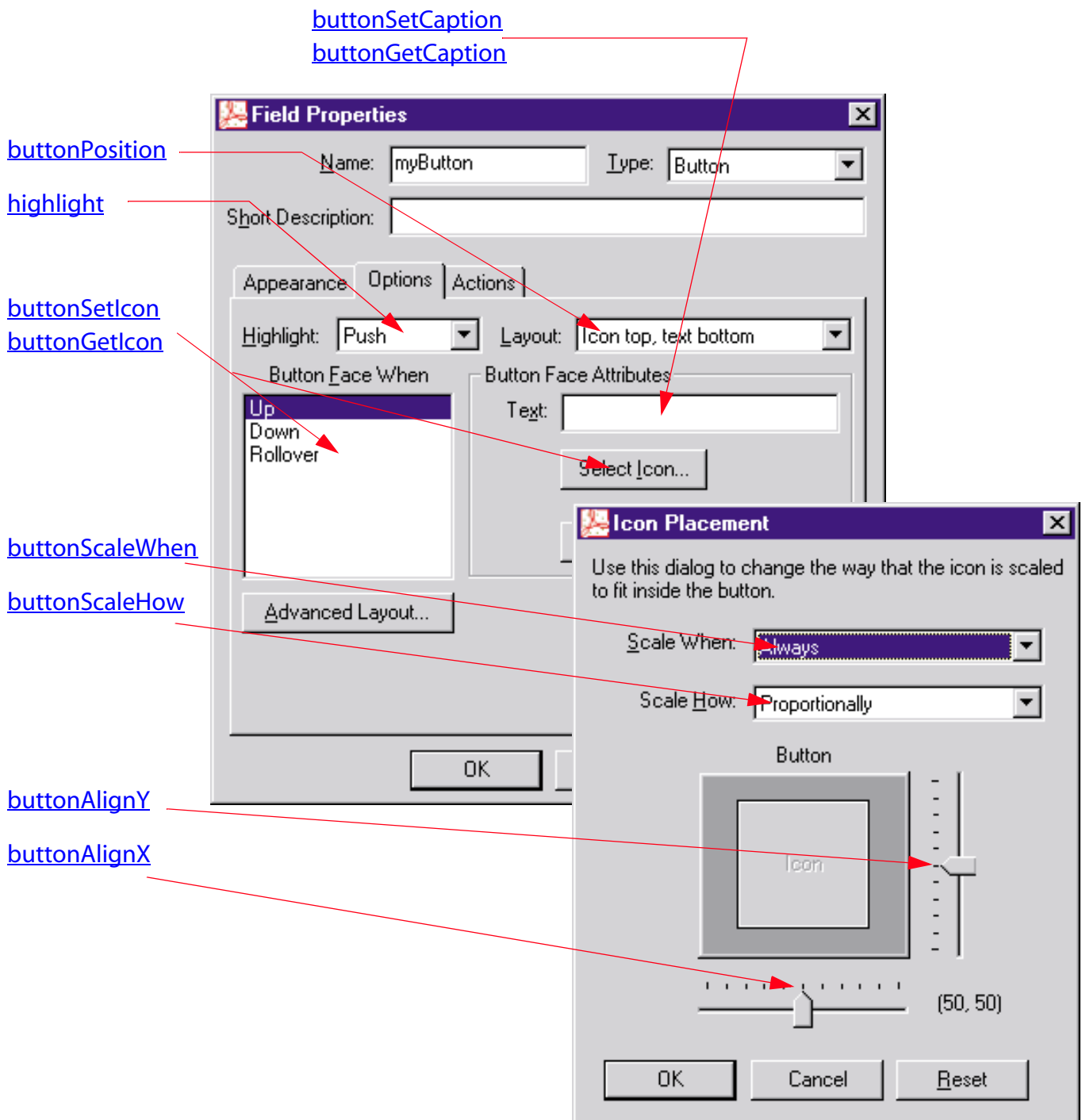
```
f.setAction("MouseUp", "app.beep(0);");
```

最初の引数はアクションのトリガーで、2つ目の引数はトリガーイベントの発生時に実行される JavaScript です。



オプション：ボタン

詳細および例については、[ボタン](#)フィールドの「[オプション](#)」タブを参照してください。



オプション：チェックボックスとラジオボタン

詳細および例については、[チェックボックス](#)フィールドの「[オプション](#)」タブおよび[ラジオボタン](#)フィールドの「[オプション](#)」タブを参照してください。

[style](#)

[setFocus](#)

[defaultIsChecked](#)

The image shows a 'Field Properties' dialog box for a 'Check Box' field. The 'Name' is 'myCheckbox' and the 'Type' is 'Check Box'. The 'Short Description' field is empty. The 'Options' tab is selected, showing 'Check Style' set to 'Check', 'Export Value' set to 'Yes', and 'Default Is Checked' as an unchecked checkbox. A hint at the bottom states: 'Hint: check boxes should be used to create lists of items where zero or more items are selectable simultaneously. To make a list of items where only one item can be selected, use a radio button field.' Red arrows point from the text labels on the left to the 'Check Style', 'Export Value', and 'Default Is Checked' controls.

Field Properties

Name: myCheckbox Type: Check Box

Short Description:

Appearance Options Actions

Check Style: Check

Export Value: Yes

☐ Default Is Checked

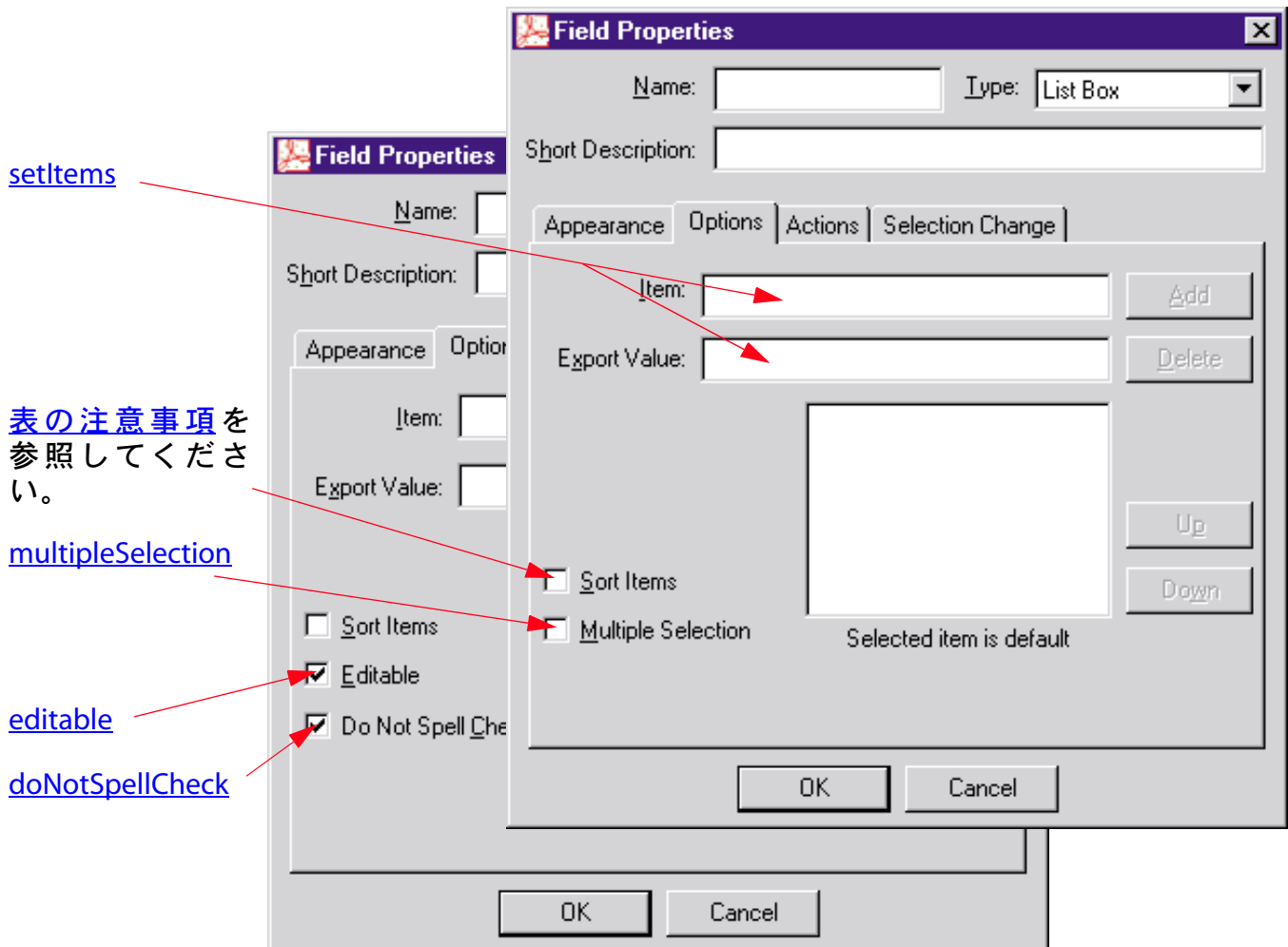
Hint: check boxes should be used to create lists of items where zero or more items are selectable simultaneously. To make a list of items where only one item can be selected, use a radio button field.

OK Cancel

オプション：コンボボックスとリストボックス

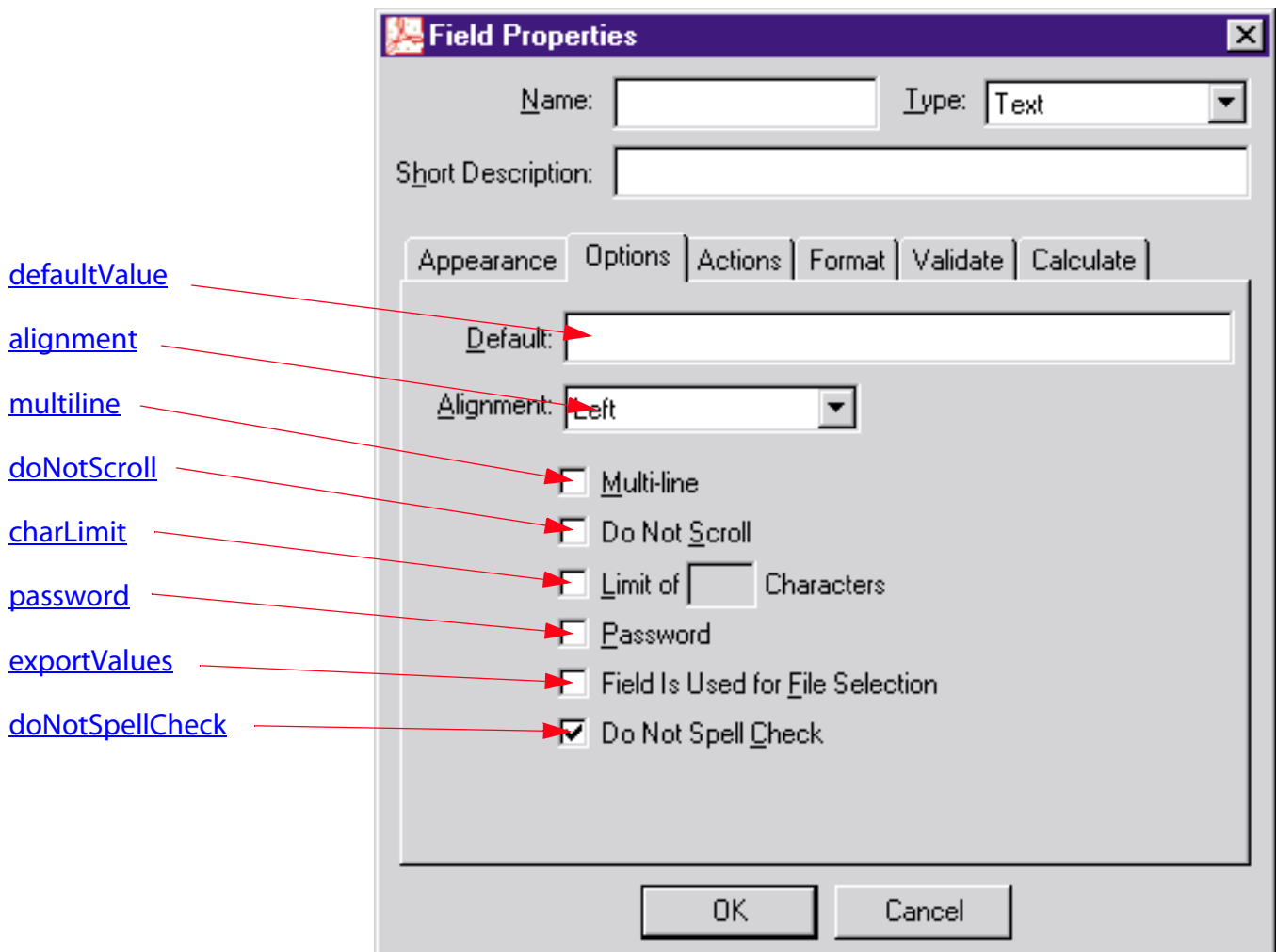
詳細および例については、コンボボックスフィールドの「[オプション](#)」タブおよびリストボックスフィールドの「[オプション](#)」タブを参照してください。

注意：上に表示されているのはリストボックスの「オプション」タブで、下はコンボボックスの「オプション」タブです。



オプション：テキストフィールド

詳細および例については、[テキスト](#)フィールドの「[オプション](#)」タブを参照してください。



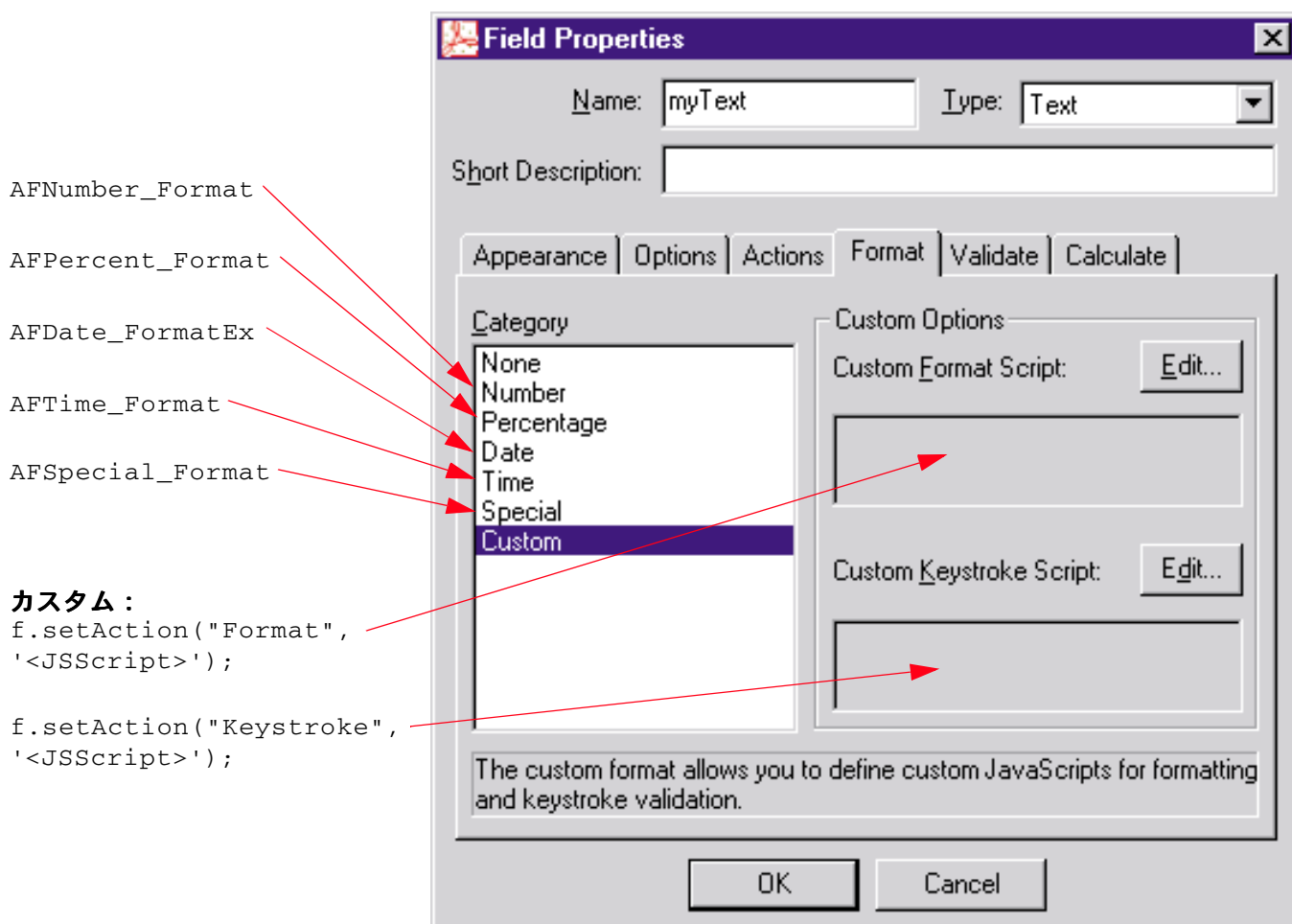
フォーマット：コンボボックスとテキスト

詳細および例については、[コンボボックス](#)フィールドの「[フォーマット](#)」タブおよび[テキスト](#)フィールドの「[フォーマット](#)」タブを参照してください。

カスタムキーストロースクリプトを除いて、フォーマットのすべてのカテゴリは、フィールドメソッドの [setAction](#) により設定可能です。

```
f.setAction("Format", '<JSScript>');
```

JavaScript でアクションを設定する場合は、aform.js に定義されている次の JavaScript 関数を使用してください。



検証：コンボボックスとテキスト

詳細および例については、[コンボボックス](#)フィールドの「[検証](#)」タブおよび[テキスト](#)フィールドの「[検証](#)」タブを参照してください。

検証スクリプトを設定するには、フィールドメソッドの [setAction](#) を使用します。

```
f.setAction("Validate", '<JScript>');
```

```
f.setAction("Validate",  
'AFSimple_Calculate(..)');
```

```
f.setAction("Validate",  
'<JScript>');
```

The screenshot shows the 'Field Properties' dialog box with the 'Validate' tab selected. The 'Name' field is 'myText' and the 'Type' is 'Text'. The 'Short Description' field is empty. The 'Validate' tab contains three radio button options: 'Value is not validated.', 'Value must be greater than or equal to' (with a text input field), and 'Custom validate script' (with a large text area and an 'Edit...' button). Red arrows point from the code snippets on the left to the corresponding options in the dialog.

計算：コンボボックスとテキスト

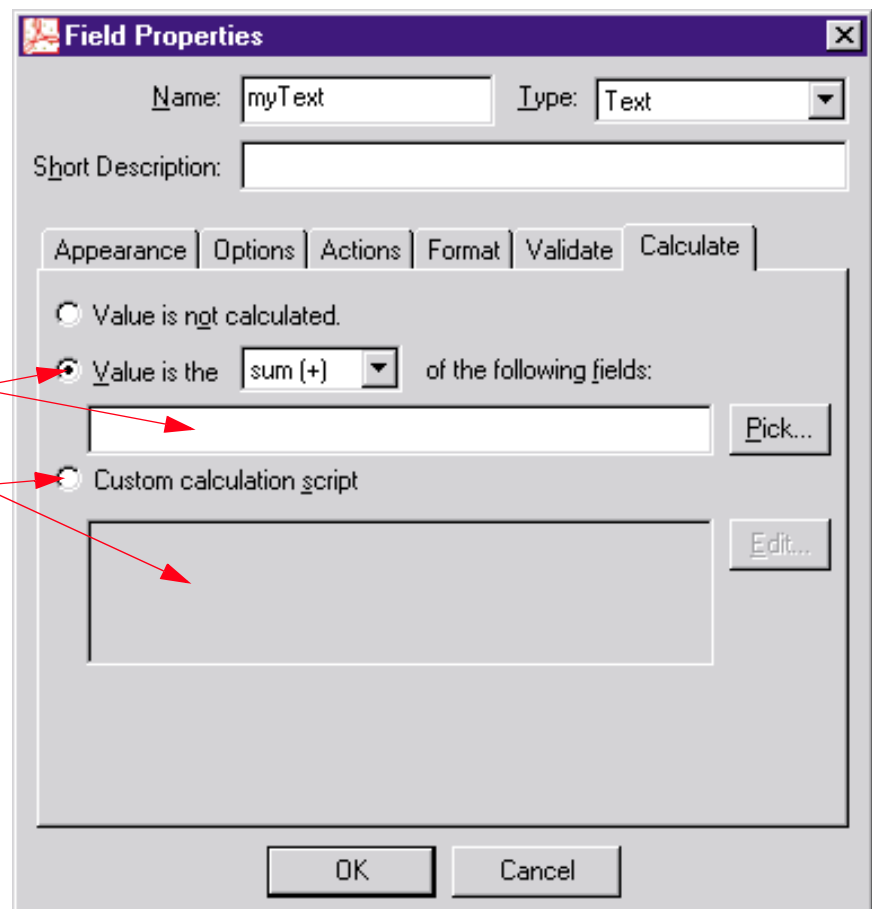
詳細および例については、[コンボボックス](#)フィールドの「[計算](#)」タブおよび[テキスト](#)フィールドの「[計算](#)」タブを参照してください。

計算スクリプトを設定するには、フィールドメソッドの [setAction](#) を使用します。

```
f.setAction("Calculate", '<JScript>');
```

```
f.setAction("Calculate",  
'AFSimple_Calculate(...)');
```

```
f.setAction("Calculate",  
'<JScript>');
```



署名

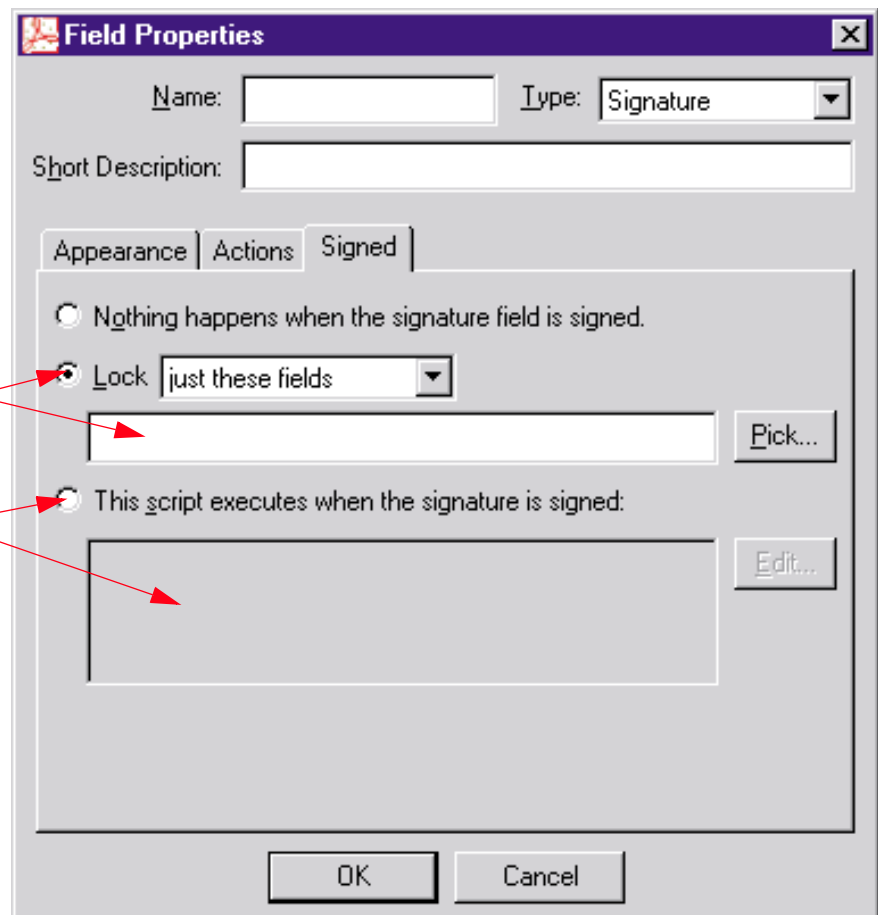
詳細および例については、[署名](#)フィールドの「[署名](#)」タブを参照してください。

署名スクリプトを設定するには、フィールドメソッドの [setAction](#) を使用します。

```
f.setAction("Format", '<JScript>');
```

```
f.setAction("Format",  
'AFSignature_Format(...)');
```

```
f.setAction("Format",  
'AFSignature_Format(...)');
```



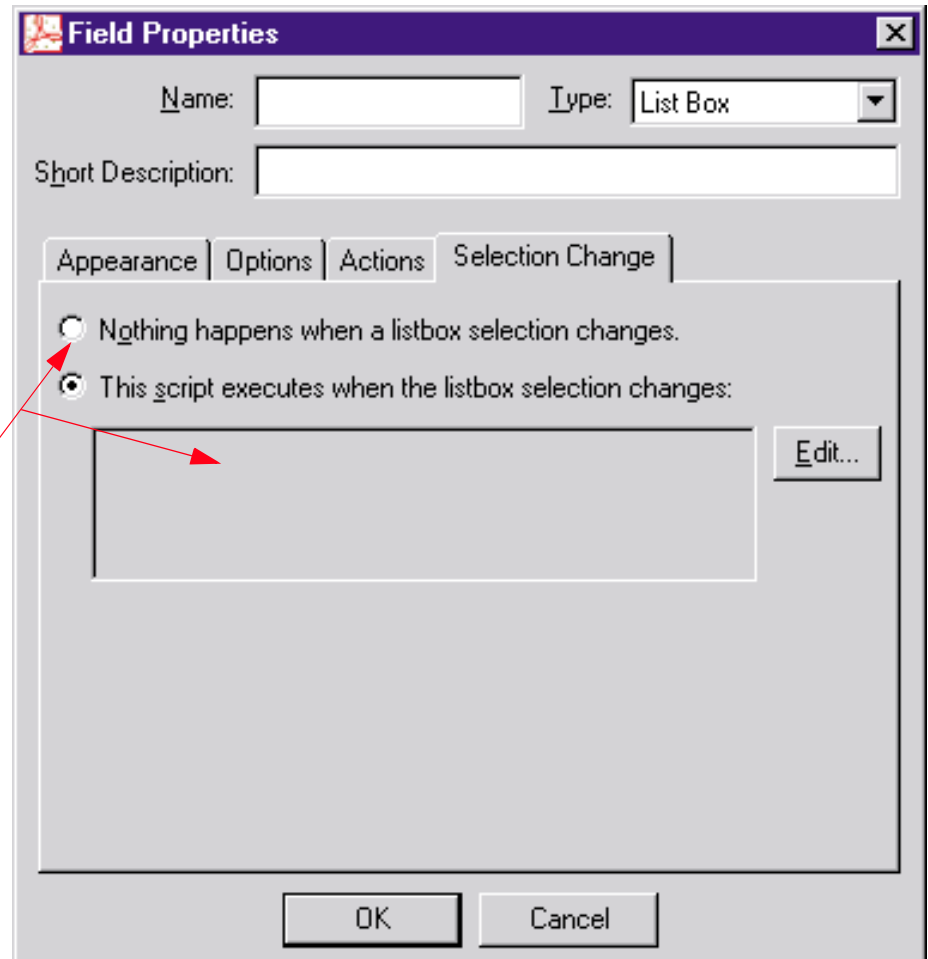
選択の変更

詳細および例については、[リストボックス](#) フィールドの「[選択の変更](#)」タブを参照してください。

選択の変更スクリプトを設定するには、フィールドメソッドの [setAction](#) を使用します。

```
f.setAction("Keystroke", '<JScript>');
```

```
f.setAction("Keystroke",  
'<JScript>');
```



どうすれば注釈をプログラムで作成できますか？

Acrobat には、注釈を作成するための JavaScript プロパティおよび JavaScript メソッドが数多く用意されています。以下に、これらのメソッドのアウトラインを簡単に示します。

注釈には 13 種類のタイプがあり（ここでは、そのうちの 11 種類のみを取り上げます）、これらは類似したプロパティに分類することができます。

- [Circle および Square 注釈](#)
- [Text 注釈](#)
- [Line 注釈](#)
- [Ink 注釈](#)
- [Stamp 注釈](#)
- [Highlight、Strikeout、Underline、Squiggle](#)
- [FreeText 注釈](#)

Circle および Square 注釈

円の注釈を作成するには、[addAnnot](#) を使用します。

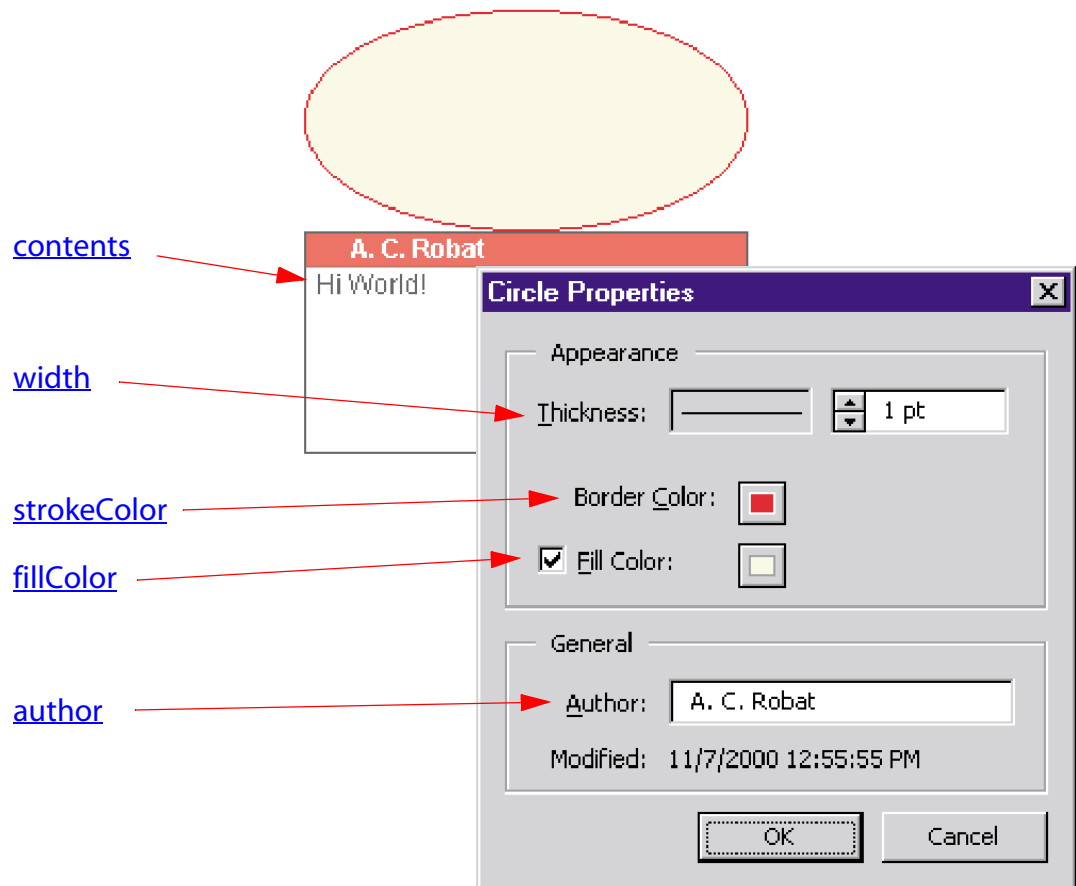
例：

```
var annot = this.addAnnot(  
  {  
    type: "Circle",  
    page: 0,  
    rect: [200,200,400,300],  
    author: "A. C. Robat",  
    name: "myCircle",  
    popupOpen: true,  
    popupRect: [200,100,400,200],  
    contents: "Hi World!",  
    strokeColor: color.red,  
    fillColor: ["RGB",1,1,.855]  
  });
```

次に、上の例を少し変えて、[addAnnot](#) の他に [setProps](#) も使用してみます。

例：

```
var annot = this.addAnnot  
  ({ type: "Square" });  
annot.setProps  
  ({  
    page: 0,  
    rect: [200,200,400,400],  
    author: "A. C. Robat",  
    name: "mySquare",  
    popupOpen: true,  
    popupRect: [200,100,400,200],  
    contents: "Hi World!",  
    strokeColor: color.red  
  });
```

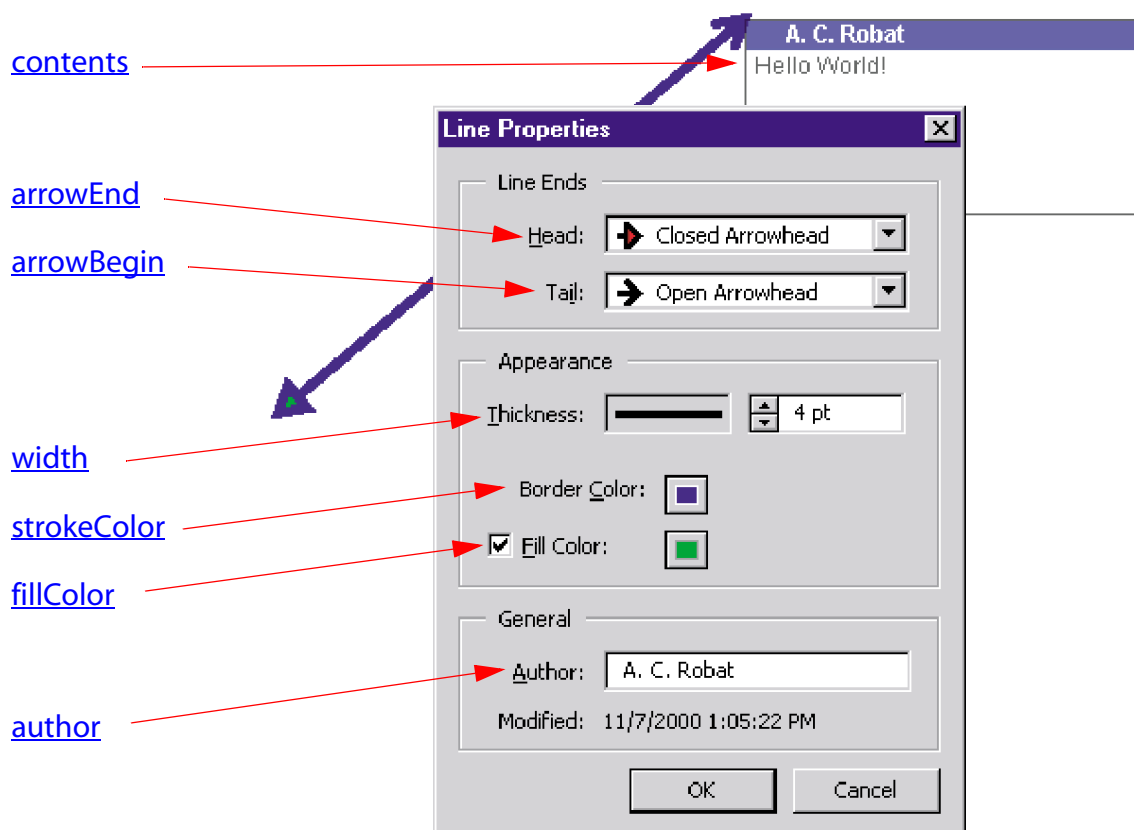


Line 注釈

線の注釈を作成するには、[addAnnot](#) を使用するか、[addAnnot](#) と [setProps](#) を組み合わせて使
用します（例については [Circle および Square 注釈](#) を参照）。

例：

```
var annot = this.addAnnot
({
  type: "Line",
  page: 0,
  points: [[10,40],[200,200]],
  author: "A. C. Robot",
  name: "myLine",
  popupOpen: true,
  popupRect: [200, 100, 400, 200],
  arrowBegin: "ClosedArrow",
  arrowEnd: "OpenArrow",
  width: 4,
  contents: "Hello World!",
  strokeColor: color.red,           // border and text color
  fillColor: color.green           // background color
});
```

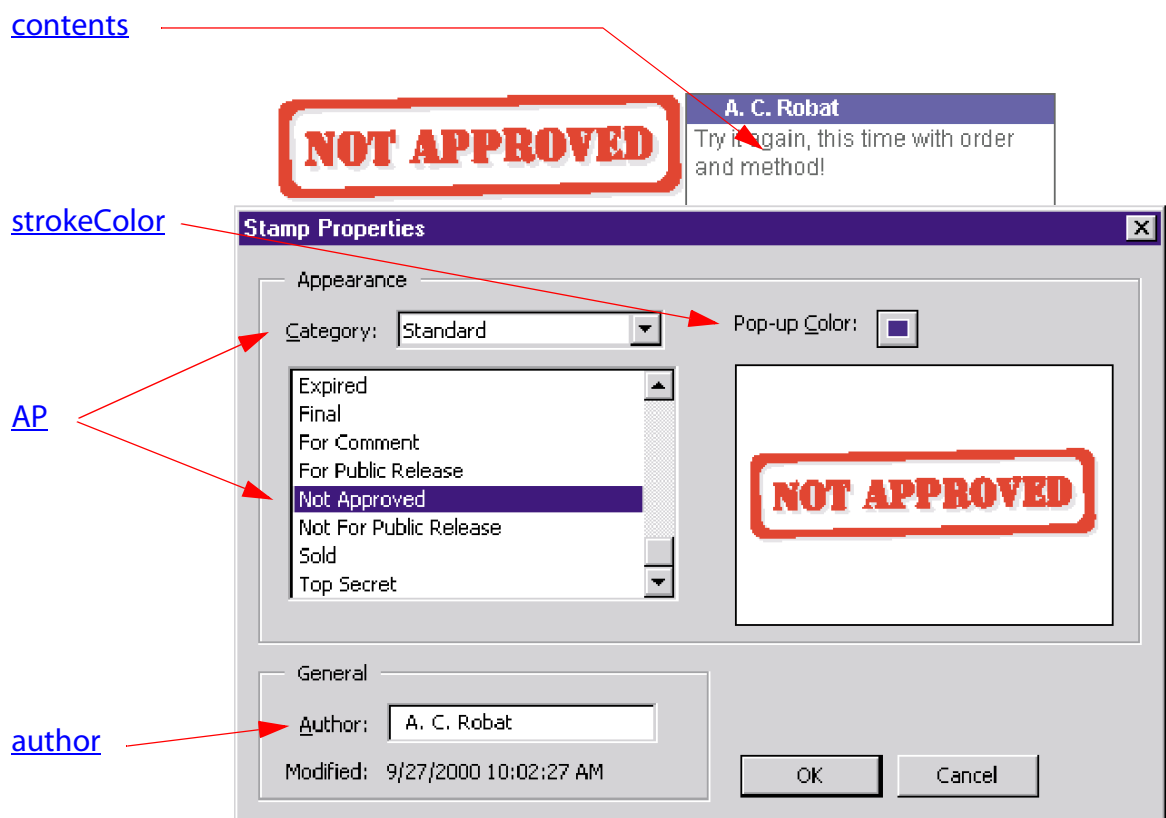


Stamp 注釈

スタンプの注釈を作成するには、[addAnnot](#) を使用するか、[addAnnot](#) と [setProps](#) を組み合わせて使用します（例については [Circle および Square 注釈](#) を参照）。

例：

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

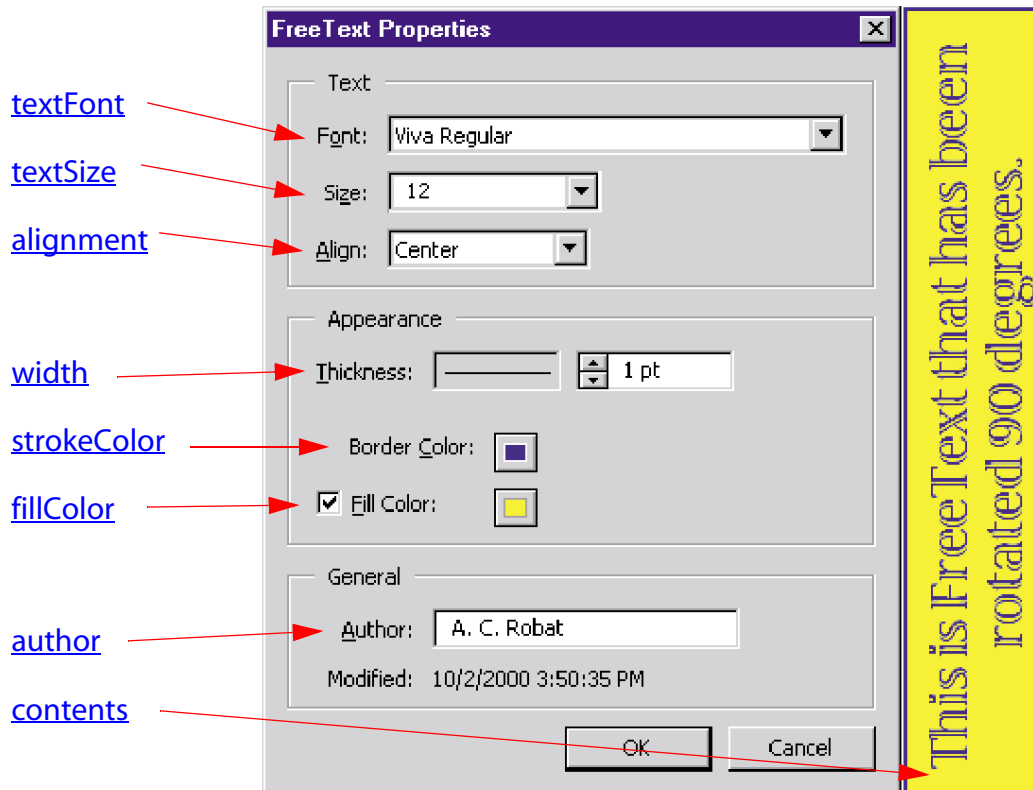


FreeText 注釈

テキスト注釈を作成するには、[addAnnot](#) を使用するか、[addAnnot](#) と [setProps](#) を組み合わせて使用します（例については [Circle および Square 注釈](#) を参照）。

例：

```
var annot = this.addAnnot
({
  page: 0,
  type: "FreeText",
  author: "A. C. Robat",
  textFont: "Viva-Regular",
  textSize: 12,
  alignment: 1,
  rect: [10, 10, 42, 200],
  fillColor: ["RGB", 1, 1, 0],
  strokeColor: color.blue,
  name: "myFreeText",
  contents: "This is FreeText that has ¥
           been rotated 90 degrees.",
  rotate: 90 // no GUI for rotation
});
```

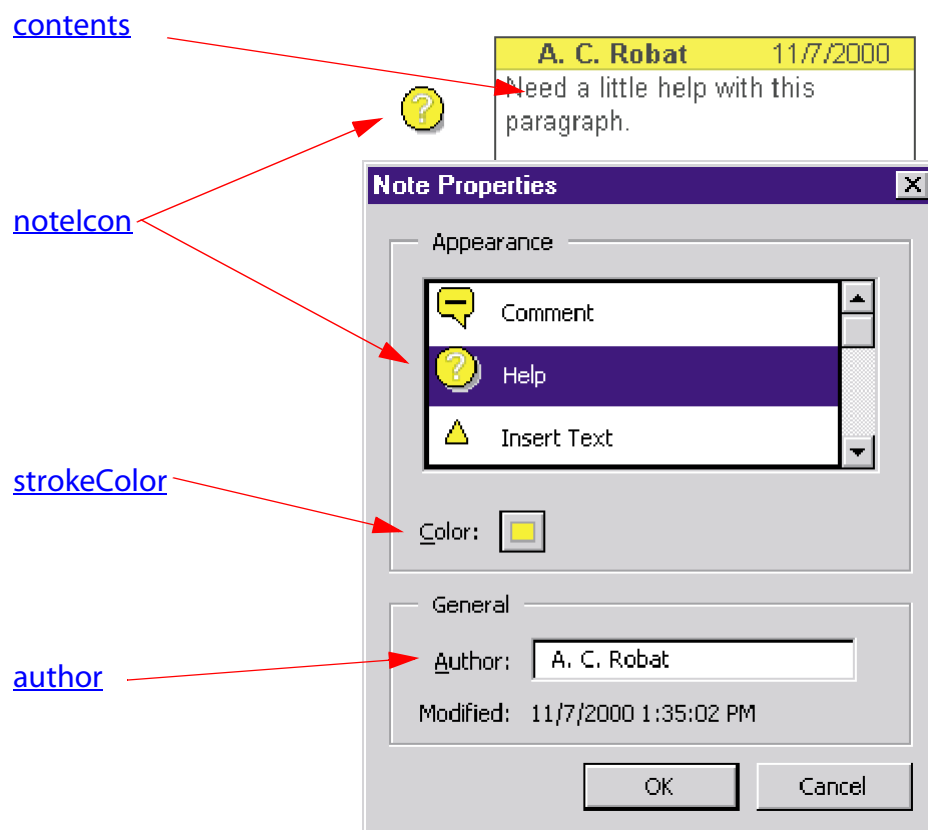


Text 注釈

ノート注釈を作成するには、[addAnnot](#) を使用するか、[addAnnot](#) と [setProps](#) を組み合わせて使用します（例については [Circle および Square 注釈](#) を参照）。

例：

```
var annot = this.addAnnot
({
  page: 0,
  type: "Text",
  author: "A. C. Robat",
  point: [300,400],
  strokeColor: color.yellow,
  name: "myHelp",
  contents: "Need a little help with this paragraph.",
  noteIcon: "Help"
});
```

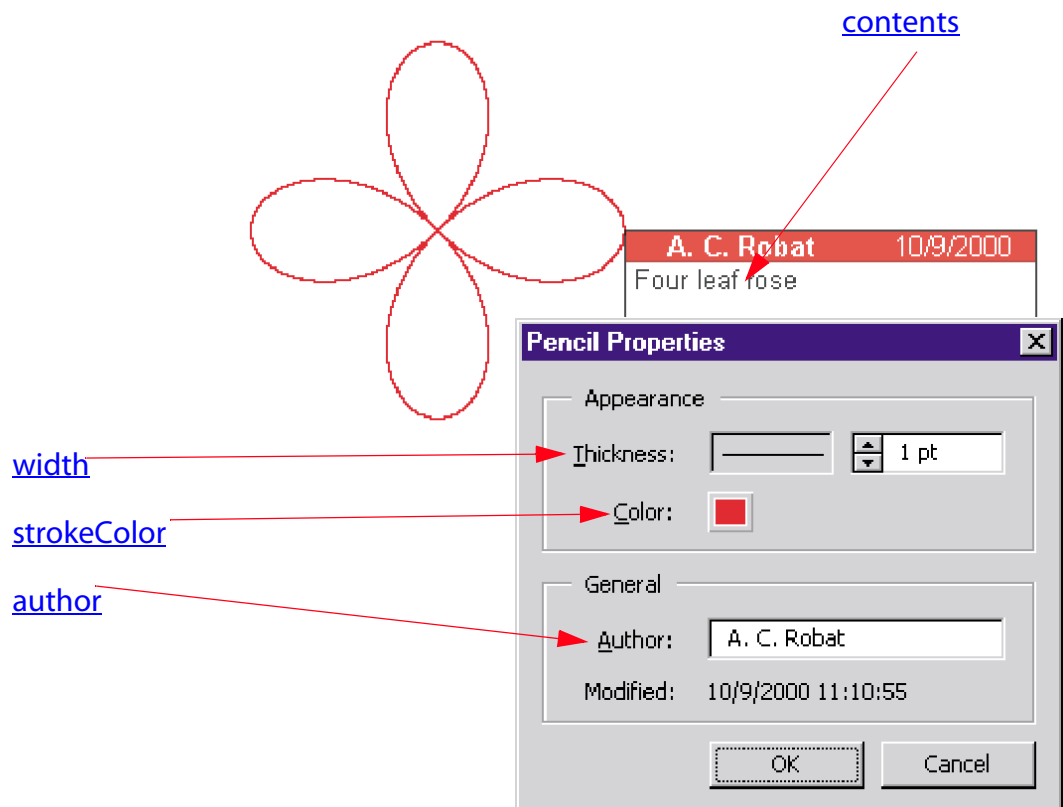


Ink 注釈

Ink 注釈を作成するには、[addAnnot](#) を使用するか、[addAnnot](#) と [setProps](#) を組み合わせて使
用します（例については [Circle および Square 注釈](#) を参照）。

例：

```
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(2*Theta)*Math.cos(Theta)*scaledInch,
        y0 + 2*Math.cos(2*Theta)*Math.sin(Theta)*scaledInch];
}
var annot = this.addAnnot({
    type: "Ink",
    page: 0,
    name: "myRose",
    author: "A. C. Robat",
    contents: "Four leaf rose",
    gestures: [points],
    strokeColor: color.red,
    width: 1
});
```



Highlight, Strikeout, Underline, Squiggle

マークアップ注釈を作成するには、[addAnnot](#) を使用するか、[addAnnot](#) と [setProps](#) を組み合わせて使用します（例については [Circle および Square 注釈](#) を参照）。

例：以下のコードはページ中の「mark up annotations」という 3 つの単語を強調表示します。このスクリプトは現在のページで最初に出現する「mark」という単語を検索し、次にこの単語を囲む矩形とその 2 つ後の「annotations」という単語の矩形を取得します。そして「mark」と「annotations」の間にあるすべての単語を包むようにして 2 つの矩形を結合し、1 つの矩形を形成します。これは実際的な例ではありませんが、矩形を扱う際のテクニックを示しています。

```
var thisPage = this.pageNum;
var numWords = this.getPageNumWords(thisPage);
for ( var j = 0; j < numWords; j++) {
    nthWord = this.getPageNthWord(thisPage,j)
    if ( nthWord == "mark" ) {
        aQuadsFirst = this.getPageNthWordQuads(thisPage,j);
        aQuadsLast  = this.getPageNthWordQuads(thisPage,j+2);
        annot = this.addAnnot({
            page: thisPage,
            type: "Highlight",      // "Underline", "StrikeOut", or "Squiggly"
            strokeColor: color.yellow,
            quads: [[ aQuadsFirst[0][0], aQuadsFirst[0][1],
                    aQuadsLast[0][2], aQuadsLast[0][3],
                    aQuadsFirst[0][4], aQuadsFirst[0][5],
                    aQuadsLast[0][6], aQuadsLast[0][7]
                    ]],
            author: "A. C. Acrobat",
            contents: "Highlight, Underline,StrikeOut, and Squiggly"
        });
        break;
    }
}
```

お試しください：このコードをコピーして JavaScript コンソールに貼り付けてください。そして貼り付けたコードをすべて選択し、Ctrl-Enter キーまたは数値パッドの Enter キーを押してコードを実行します。上記の「mark up annotations」というフレーズが黄色で強調表示されるはずです。スペルチェックの例については、[checkWord](#) を参照してください。

[contents](#)

[strokeColor](#)

[author](#)

