Acrobat JavaScript Scripting Reference

Technical Note #5431
Version : Acrobat 6.0

February 2004
# Contents

## Preface

- Introduction .......................... 19
- What's In This Document ................. 19
- Document Conventions .................... 19
- Font Conventions Used in This Book .... 19
- Quick Bars ............................. 21
- Other Sources of Information ............. 23
- Online Help ............................ 23
- References ............................. 23

## Acrobat JavaScript Scripting Reference

- ADBC Object .................................. 25
  - ADBC Properties ....................... 26
    - SQL Types ............................. 26
    - JavaScript Types ................... 27
  - ADBC Methods ......................... 27
    - getDataSourceList .................. 27
    - newConnection .................... 28
- AlternatePresentation Object .......... 29
  - AlternatePresentation Properties .... 29
    - active ............................. 29
    - type ............................. 29
  - AlternatePresentation Methods ....... 30
    - start ............................. 30
    - stop ............................. 30
- Annot Object ............................ 31
- Annotation Types ....................... 31
- Annotation Access from JavaScript .... 33
  - Annot Properties ..................... 33
    - alignment .......................... 33
    - AP ................................. 33
    - arrowBegin ......................... 34
    - arrowEnd .......................... 35
    - attachIcon .......................... 35
    - author ............................. 35
    - borderEffectIntensity ............. 36
    - borderEffectStyle ................. 36
    - contents .......................... 36
    - doc ............................... 37
fillColor ................................................................. 37
gestures ............................................................... 38
hidden ................................................................. 38
inReplyTo ............................................................. 38
modDate ............................................................... 38
name ................................................................. 39
noteIcon ............................................................. 39
noView ............................................................... 39
page ................................................................. 40
point ................................................................. 40
points ............................................................... 40
popupOpen .......................................................... 41
popupRect .......................................................... 41
print ............................................................... 42
quads ............................................................... 42
rect ................................................................. 42
readOnly ........................................................... 42
richContents ....................................................... 43
rotate ............................................................... 44
strokeColor ........................................................ 44
textFont ........................................................... 44
textSize ............................................................ 45
toggleNoView .................................................... 45
type ............................................................... 46
soundIcon .......................................................... 46
width .............................................................. 46
Annot Methods .................................................. 47
destroy ............................................................. 47
getProps ........................................................... 47
getStateInModel .................................................. 48
setProps ........................................................... 48
transitionToState ................................................ 49
App Object .......................................................... 50
App Properties ..................................................... 50
activeDocs ........................................................ 50
calculate ........................................................... 51
focusRect ........................................................... 51
formsVersion ....................................................... 51
fromPDFConverters ............................................... 52
fs ................................................................. 52
fullscreen ........................................................ 52
language ........................................................... 53
numPlugins ........................................................ 54
openInPlace ....................................................... 54
platform ........................................................... 54
plugins ............................................................ 55
printColorProfiles ............................................... 55
printerNames ........................................................ 55
runtimeHighlight ............................................... 56
runtimeHighlightColor ......................................... 56
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>addMenuItem</td>
<td>58</td>
</tr>
<tr>
<td>addSubMenu</td>
<td>60</td>
</tr>
<tr>
<td>alert</td>
<td>61</td>
</tr>
<tr>
<td>beep</td>
<td>63</td>
</tr>
<tr>
<td>clearInterval</td>
<td>65</td>
</tr>
<tr>
<td>clearTimeOut</td>
<td>66</td>
</tr>
<tr>
<td>execMenuItem</td>
<td>66</td>
</tr>
<tr>
<td>getNthPlugInName</td>
<td>67</td>
</tr>
<tr>
<td>getPath</td>
<td>68</td>
</tr>
<tr>
<td>goBack</td>
<td>68</td>
</tr>
<tr>
<td>goForward</td>
<td>69</td>
</tr>
<tr>
<td>hideMenuItem</td>
<td>69</td>
</tr>
<tr>
<td>hideToolBarButton</td>
<td>70</td>
</tr>
<tr>
<td>listMenuItems</td>
<td>71</td>
</tr>
<tr>
<td>listToolBarButtons</td>
<td>71</td>
</tr>
<tr>
<td>mailGetAddr</td>
<td>71</td>
</tr>
<tr>
<td>mailMsg</td>
<td>73</td>
</tr>
<tr>
<td>newDoc</td>
<td>73</td>
</tr>
<tr>
<td>newFDF</td>
<td>74</td>
</tr>
<tr>
<td>openDoc</td>
<td>75</td>
</tr>
<tr>
<td>openFDF</td>
<td>76</td>
</tr>
<tr>
<td>popUpMenu</td>
<td>77</td>
</tr>
<tr>
<td>popUpMenuEx</td>
<td>77</td>
</tr>
<tr>
<td>removeToolBarButton</td>
<td>79</td>
</tr>
<tr>
<td>response</td>
<td>79</td>
</tr>
<tr>
<td>setInterval</td>
<td>80</td>
</tr>
<tr>
<td>setTimeout</td>
<td>81</td>
</tr>
<tr>
<td>execute</td>
<td>85</td>
</tr>
<tr>
<td>insertChild</td>
<td>85</td>
</tr>
<tr>
<td>remove</td>
<td>85</td>
</tr>
<tr>
<td>thermometer</td>
<td>56</td>
</tr>
<tr>
<td>toolbar</td>
<td>57</td>
</tr>
<tr>
<td>toolbarHorizontal</td>
<td>57</td>
</tr>
<tr>
<td>toolbarVertical</td>
<td>57</td>
</tr>
<tr>
<td>viewerType</td>
<td>58</td>
</tr>
<tr>
<td>viewerVariation</td>
<td>58</td>
</tr>
<tr>
<td>viewerVersion</td>
<td>58</td>
</tr>
<tr>
<td>bookmarkProperties</td>
<td>82</td>
</tr>
<tr>
<td>bookmarkMethods</td>
<td>84</td>
</tr>
<tr>
<td>createChild</td>
<td>84</td>
</tr>
<tr>
<td>execute</td>
<td>85</td>
</tr>
<tr>
<td>insertChild</td>
<td>85</td>
</tr>
<tr>
<td>remove</td>
<td>86</td>
</tr>
<tr>
<td>Class</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Catalog Object</td>
<td>87</td>
</tr>
<tr>
<td>Catalog Properties</td>
<td>87</td>
</tr>
<tr>
<td>isIdle</td>
<td>87</td>
</tr>
<tr>
<td>jobs</td>
<td>87</td>
</tr>
<tr>
<td>Catalog Methods</td>
<td>88</td>
</tr>
<tr>
<td>getIndex</td>
<td>88</td>
</tr>
<tr>
<td>remove</td>
<td>88</td>
</tr>
<tr>
<td>CatalogJob Generic Object</td>
<td>89</td>
</tr>
<tr>
<td>Certificate Object</td>
<td>89</td>
</tr>
<tr>
<td>Certificate Properties</td>
<td>90</td>
</tr>
<tr>
<td>binary</td>
<td>90</td>
</tr>
<tr>
<td>issuerDN</td>
<td>90</td>
</tr>
<tr>
<td>keyUsage</td>
<td>90</td>
</tr>
<tr>
<td>MD5Hash</td>
<td>90</td>
</tr>
<tr>
<td>SHA1Hash</td>
<td>91</td>
</tr>
<tr>
<td>serialNumber</td>
<td>91</td>
</tr>
<tr>
<td>subjectCN</td>
<td>91</td>
</tr>
<tr>
<td>subjectDN</td>
<td>91</td>
</tr>
<tr>
<td>usage</td>
<td>91</td>
</tr>
<tr>
<td>Collab Object</td>
<td>93</td>
</tr>
<tr>
<td>Collab Methods</td>
<td>93</td>
</tr>
<tr>
<td>addStateModel</td>
<td>93</td>
</tr>
<tr>
<td>removeStateModel</td>
<td>94</td>
</tr>
<tr>
<td>Color Object</td>
<td>95</td>
</tr>
<tr>
<td>Color Arrays</td>
<td>95</td>
</tr>
<tr>
<td>Color Properties</td>
<td>96</td>
</tr>
<tr>
<td>Color Methods</td>
<td>97</td>
</tr>
<tr>
<td>convert</td>
<td>97</td>
</tr>
<tr>
<td>equal</td>
<td>98</td>
</tr>
<tr>
<td>Column Generic Object</td>
<td>98</td>
</tr>
<tr>
<td>ColumnInfo Generic Object</td>
<td>99</td>
</tr>
<tr>
<td>Connection Object</td>
<td>99</td>
</tr>
<tr>
<td>Connection Methods</td>
<td>100</td>
</tr>
<tr>
<td>close</td>
<td>100</td>
</tr>
<tr>
<td>newStatement</td>
<td>100</td>
</tr>
<tr>
<td>getTableList</td>
<td>100</td>
</tr>
<tr>
<td>getColumnList</td>
<td>101</td>
</tr>
<tr>
<td>Console Object</td>
<td>102</td>
</tr>
<tr>
<td>Console Methods</td>
<td>102</td>
</tr>
<tr>
<td>show</td>
<td>102</td>
</tr>
<tr>
<td>hide</td>
<td>102</td>
</tr>
<tr>
<td>println</td>
<td>102</td>
</tr>
<tr>
<td>clear</td>
<td>103</td>
</tr>
<tr>
<td>Data Object</td>
<td>103</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>DataSourceInfo Generic Object</td>
<td>105</td>
</tr>
<tr>
<td>DirConnection Object</td>
<td>114</td>
</tr>
<tr>
<td>Doc Object</td>
<td>119</td>
</tr>
<tr>
<td>Doc Access from JavaScript</td>
<td>120</td>
</tr>
<tr>
<td>Data Properties</td>
<td>104</td>
</tr>
<tr>
<td>creationDate</td>
<td>104</td>
</tr>
<tr>
<td>modDate</td>
<td>104</td>
</tr>
<tr>
<td>MIMEType</td>
<td>104</td>
</tr>
<tr>
<td>name</td>
<td>104</td>
</tr>
<tr>
<td>path</td>
<td>105</td>
</tr>
<tr>
<td>size</td>
<td>105</td>
</tr>
<tr>
<td>Dbg Object</td>
<td>105</td>
</tr>
<tr>
<td>Dbg Properties</td>
<td>106</td>
</tr>
<tr>
<td>bps</td>
<td>106</td>
</tr>
<tr>
<td>Dbg Methods</td>
<td>107</td>
</tr>
<tr>
<td>c</td>
<td>107</td>
</tr>
<tr>
<td>cb</td>
<td>107</td>
</tr>
<tr>
<td>q</td>
<td>108</td>
</tr>
<tr>
<td>sb</td>
<td>108</td>
</tr>
<tr>
<td>si</td>
<td>109</td>
</tr>
<tr>
<td>sn</td>
<td>109</td>
</tr>
<tr>
<td>so</td>
<td>110</td>
</tr>
<tr>
<td>sv</td>
<td>110</td>
</tr>
<tr>
<td>Directory Object</td>
<td>110</td>
</tr>
<tr>
<td>Directory Properties</td>
<td>111</td>
</tr>
<tr>
<td>info</td>
<td>111</td>
</tr>
<tr>
<td>Directory Methods</td>
<td>114</td>
</tr>
<tr>
<td>connect</td>
<td>114</td>
</tr>
<tr>
<td>DirConnection Object</td>
<td>114</td>
</tr>
<tr>
<td>DirConnection Properties</td>
<td>115</td>
</tr>
<tr>
<td>canList</td>
<td>115</td>
</tr>
<tr>
<td>canDoCustomSearch</td>
<td>115</td>
</tr>
<tr>
<td>canDoCustomUISearch</td>
<td>115</td>
</tr>
<tr>
<td>canDoStandardSearch</td>
<td>116</td>
</tr>
<tr>
<td>groups</td>
<td>116</td>
</tr>
<tr>
<td>name</td>
<td>116</td>
</tr>
<tr>
<td>uiName</td>
<td>116</td>
</tr>
<tr>
<td>DirConnection Methods</td>
<td>117</td>
</tr>
<tr>
<td>search</td>
<td>117</td>
</tr>
<tr>
<td>setOutputFields</td>
<td>119</td>
</tr>
<tr>
<td>Doc Access from JavaScript</td>
<td>120</td>
</tr>
<tr>
<td>Doc Properties</td>
<td>120</td>
</tr>
<tr>
<td>alternatePresentations</td>
<td>120</td>
</tr>
<tr>
<td>author</td>
<td>121</td>
</tr>
<tr>
<td>baseURL</td>
<td>122</td>
</tr>
<tr>
<td>bookmarkRoot</td>
<td>122</td>
</tr>
<tr>
<td>calculate</td>
<td>122</td>
</tr>
<tr>
<td>creationDate</td>
<td>122</td>
</tr>
<tr>
<td>creator</td>
<td>123</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>dataObjects</td>
<td>123</td>
</tr>
<tr>
<td>delay</td>
<td>123</td>
</tr>
<tr>
<td>dirty</td>
<td>123</td>
</tr>
<tr>
<td>disclosed</td>
<td>124</td>
</tr>
<tr>
<td>documentFileName</td>
<td>124</td>
</tr>
<tr>
<td>external</td>
<td>125</td>
</tr>
<tr>
<td>filesize</td>
<td>125</td>
</tr>
<tr>
<td>icons</td>
<td>125</td>
</tr>
<tr>
<td>info</td>
<td>126</td>
</tr>
<tr>
<td>keywords</td>
<td>127</td>
</tr>
<tr>
<td>layout</td>
<td>128</td>
</tr>
<tr>
<td>metadata</td>
<td>128</td>
</tr>
<tr>
<td>modDate</td>
<td>129</td>
</tr>
<tr>
<td>numFields</td>
<td>129</td>
</tr>
<tr>
<td>numPages</td>
<td>129</td>
</tr>
<tr>
<td>numTemplates</td>
<td>129</td>
</tr>
<tr>
<td>path</td>
<td>129</td>
</tr>
<tr>
<td>pageNum</td>
<td>130</td>
</tr>
<tr>
<td>permStatusReady</td>
<td>130</td>
</tr>
<tr>
<td>producer</td>
<td>130</td>
</tr>
<tr>
<td>securityHandler</td>
<td>130</td>
</tr>
<tr>
<td>selectedAnnots</td>
<td>131</td>
</tr>
<tr>
<td>sounds</td>
<td>131</td>
</tr>
<tr>
<td>spellDictionaryOrder</td>
<td>132</td>
</tr>
<tr>
<td>spellLanguageOrder</td>
<td>132</td>
</tr>
<tr>
<td>subject</td>
<td>132</td>
</tr>
<tr>
<td>templates</td>
<td>132</td>
</tr>
<tr>
<td>title</td>
<td>133</td>
</tr>
<tr>
<td>URL</td>
<td>133</td>
</tr>
<tr>
<td>zoom</td>
<td>133</td>
</tr>
<tr>
<td>zoomType</td>
<td>134</td>
</tr>
<tr>
<td>Doc Methods</td>
<td>134</td>
</tr>
<tr>
<td>addAnnot</td>
<td>134</td>
</tr>
<tr>
<td>addField</td>
<td>135</td>
</tr>
<tr>
<td>addIcon</td>
<td>137</td>
</tr>
<tr>
<td>addLink</td>
<td>137</td>
</tr>
<tr>
<td>addRecipientListCryptFilter</td>
<td>139</td>
</tr>
<tr>
<td>addScript</td>
<td>140</td>
</tr>
<tr>
<td>addThumbnails</td>
<td>140</td>
</tr>
<tr>
<td>addWeblinks</td>
<td>141</td>
</tr>
<tr>
<td>bringToFront</td>
<td>141</td>
</tr>
<tr>
<td>calculateNow</td>
<td>142</td>
</tr>
<tr>
<td>closeDoc</td>
<td>142</td>
</tr>
<tr>
<td>createDataObject</td>
<td>143</td>
</tr>
<tr>
<td>createTemplate</td>
<td>143</td>
</tr>
<tr>
<td>deletePages</td>
<td>144</td>
</tr>
<tr>
<td>deleteSound</td>
<td>145</td>
</tr>
<tr>
<td>encryptForRecipients</td>
<td>145</td>
</tr>
<tr>
<td>exportAsText</td>
<td>147</td>
</tr>
<tr>
<td>exportAsFDF</td>
<td>148</td>
</tr>
<tr>
<td>exportAsXFDF</td>
<td>150</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>exportDataObject</td>
<td>151</td>
</tr>
<tr>
<td>exportXFAData</td>
<td>152</td>
</tr>
<tr>
<td>extractPages</td>
<td>153</td>
</tr>
<tr>
<td>flattenPages</td>
<td>155</td>
</tr>
<tr>
<td>getAnnot</td>
<td>155</td>
</tr>
<tr>
<td>getAnnots</td>
<td>156</td>
</tr>
<tr>
<td>getDataType</td>
<td>157</td>
</tr>
<tr>
<td>getField</td>
<td>158</td>
</tr>
<tr>
<td>getIcon</td>
<td>159</td>
</tr>
<tr>
<td>getLegalWarnings</td>
<td>159</td>
</tr>
<tr>
<td>getLinks</td>
<td>160</td>
</tr>
<tr>
<td>getNthFieldName</td>
<td>161</td>
</tr>
<tr>
<td>getNthTemplate</td>
<td>161</td>
</tr>
<tr>
<td>getOCGs</td>
<td>162</td>
</tr>
<tr>
<td>getPageBox</td>
<td>162</td>
</tr>
<tr>
<td>getPageLabel</td>
<td>163</td>
</tr>
<tr>
<td>getPageNthWord</td>
<td>164</td>
</tr>
<tr>
<td>getPageNthWordQuads</td>
<td>164</td>
</tr>
<tr>
<td>getPageNumWords</td>
<td>165</td>
</tr>
<tr>
<td>getPageRotation</td>
<td>165</td>
</tr>
<tr>
<td>getPageTransition</td>
<td>166</td>
</tr>
<tr>
<td>getPrintParams</td>
<td>166</td>
</tr>
<tr>
<td>getSound</td>
<td>167</td>
</tr>
<tr>
<td>getTemplate</td>
<td>167</td>
</tr>
<tr>
<td>getURL</td>
<td>168</td>
</tr>
<tr>
<td>gotoNamedDest</td>
<td>168</td>
</tr>
<tr>
<td>importAnFDF</td>
<td>169</td>
</tr>
<tr>
<td>importAnXFDF</td>
<td>169</td>
</tr>
<tr>
<td>importDataObject</td>
<td>170</td>
</tr>
<tr>
<td>importIcon</td>
<td>171</td>
</tr>
<tr>
<td>importSound</td>
<td>172</td>
</tr>
<tr>
<td>importTextData</td>
<td>173</td>
</tr>
<tr>
<td>importXFAData</td>
<td>174</td>
</tr>
<tr>
<td>insertPages</td>
<td>174</td>
</tr>
<tr>
<td>mailDoc</td>
<td>175</td>
</tr>
<tr>
<td>mailForm</td>
<td>176</td>
</tr>
<tr>
<td>movePage</td>
<td>177</td>
</tr>
<tr>
<td>newPage</td>
<td>178</td>
</tr>
<tr>
<td>print</td>
<td>178</td>
</tr>
<tr>
<td>removeDataObject</td>
<td>180</td>
</tr>
<tr>
<td>removeField</td>
<td>180</td>
</tr>
<tr>
<td>removeIcon</td>
<td>181</td>
</tr>
<tr>
<td>removeLinks</td>
<td>181</td>
</tr>
<tr>
<td>removeTemplate</td>
<td>182</td>
</tr>
<tr>
<td>remove Thurmbnlds</td>
<td>182</td>
</tr>
<tr>
<td>removeWeblinks</td>
<td>183</td>
</tr>
<tr>
<td>replacePages</td>
<td>184</td>
</tr>
<tr>
<td>resetForm</td>
<td>184</td>
</tr>
<tr>
<td>saveAs</td>
<td>185</td>
</tr>
<tr>
<td>scroll</td>
<td>187</td>
</tr>
<tr>
<td>selectPageNthWord</td>
<td>188</td>
</tr>
<tr>
<td>Topic</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>setAction</td>
<td>188</td>
</tr>
<tr>
<td>setPageAction</td>
<td>189</td>
</tr>
<tr>
<td>setPageBoxes</td>
<td>190</td>
</tr>
<tr>
<td>setPageLabels</td>
<td>190</td>
</tr>
<tr>
<td>setPageRotations</td>
<td>192</td>
</tr>
<tr>
<td>setPageTabOrder</td>
<td>192</td>
</tr>
<tr>
<td>setPageTransitions</td>
<td>193</td>
</tr>
<tr>
<td>spawnPageFromTemplate</td>
<td>194</td>
</tr>
<tr>
<td>submitForm</td>
<td>195</td>
</tr>
<tr>
<td>syncAnnotScan</td>
<td>199</td>
</tr>
<tr>
<td>submitForm</td>
<td>195</td>
</tr>
<tr>
<td>spawnPageFromTemplate</td>
<td>194</td>
</tr>
<tr>
<td>syncAnnotScan</td>
<td>199</td>
</tr>
<tr>
<td>Event Object</td>
<td>202</td>
</tr>
<tr>
<td>Error Properties</td>
<td>200</td>
</tr>
<tr>
<td>name</td>
<td>202</td>
</tr>
<tr>
<td>fileName</td>
<td>201</td>
</tr>
<tr>
<td>lineNumber</td>
<td>202</td>
</tr>
<tr>
<td>message</td>
<td>202</td>
</tr>
<tr>
<td>Error Methods</td>
<td>202</td>
</tr>
<tr>
<td>toString</td>
<td>202</td>
</tr>
<tr>
<td>Event Type/Name Combinations</td>
<td>203</td>
</tr>
<tr>
<td>Document Event Processing</td>
<td>209</td>
</tr>
<tr>
<td>Form Event Processing</td>
<td>210</td>
</tr>
<tr>
<td>Event Properties</td>
<td>210</td>
</tr>
<tr>
<td>change</td>
<td>210</td>
</tr>
<tr>
<td>changeEx</td>
<td>211</td>
</tr>
<tr>
<td>commitKey</td>
<td>213</td>
</tr>
<tr>
<td>fieldFull</td>
<td>213</td>
</tr>
<tr>
<td>keyDown</td>
<td>214</td>
</tr>
<tr>
<td>modifier</td>
<td>214</td>
</tr>
<tr>
<td>name</td>
<td>214</td>
</tr>
<tr>
<td>rc</td>
<td>214</td>
</tr>
<tr>
<td>richChange</td>
<td>215</td>
</tr>
<tr>
<td>richChangeEx</td>
<td>215</td>
</tr>
<tr>
<td>richValue</td>
<td>216</td>
</tr>
<tr>
<td>selEnd</td>
<td>217</td>
</tr>
<tr>
<td>selStart</td>
<td>217</td>
</tr>
<tr>
<td>shift</td>
<td>217</td>
</tr>
<tr>
<td>source</td>
<td>218</td>
</tr>
<tr>
<td>target</td>
<td>218</td>
</tr>
<tr>
<td>targetType</td>
<td>218</td>
</tr>
<tr>
<td>type</td>
<td>218</td>
</tr>
<tr>
<td>value</td>
<td>219</td>
</tr>
<tr>
<td>willCommit</td>
<td>220</td>
</tr>
<tr>
<td>FDF Object</td>
<td>221</td>
</tr>
<tr>
<td>FDF Properties</td>
<td>221</td>
</tr>
<tr>
<td>deleteOption</td>
<td>221</td>
</tr>
<tr>
<td>isSigned</td>
<td>221</td>
</tr>
<tr>
<td>numEmbeddedFiles</td>
<td>222</td>
</tr>
</tbody>
</table>
Contents

FDF Methods ................................................................. 222
   addContact ............................................................ 222
   addEmbeddedFile .................................................... 223
   addRequest ............................................................ 224
   close ................................................................. 224
   mail ................................................................. 225
   save ................................................................. 225
   signatureClear ...................................................... 226
   signatureSign ....................................................... 227
   signatureValidate .................................................. 228
Field Object ............................................................. 229
Field Access from JavaScript ............................................. 229
Field Properties .......................................................... 231
   alignment ........................................................... 231
   borderStyle ........................................................ 232
   buttonAlignX ....................................................... 232
   buttonAlignY ........................................................ 233
   buttonFitBounds ................................................... 233
   buttonPosition ..................................................... 233
   buttonScaleHow .................................................... 234
   buttonScaleWhen .................................................. 234
   calcOrderIndex ..................................................... 235
   charLimit .......................................................... 235
   comb ............................................................... 235
   commitOnSelChange ............................................... 236
   currentValueIndices .............................................. 236
   defaultStyle ........................................................ 237
   defaultValue ....................................................... 239
   doNotScroll ......................................................... 239
   doNotSpellCheck .................................................. 239
   delay .............................................................. 240
   display ............................................................ 240
   doc ................................................................. 240
   editable ............................................................ 241
   exportValues ........................................................ 241
   fileSelect .......................................................... 241
   fillColor ............................................................ 242
   hidden ............................................................. 242
   highlight ........................................................... 243
   lineWidth ........................................................... 243
   multiline ........................................................... 244
   multipleSelection .................................................. 244
   name ............................................................... 244
   numItems ............................................................ 245
   page ............................................................... 245
   password ............................................................ 245
   print ............................................................... 246
   radiosInUnison ..................................................... 246
   readonly ............................................................ 246
   rect ............................................................... 246
<table>
<thead>
<tr>
<th>Field Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>required</td>
<td>247</td>
</tr>
<tr>
<td>richText</td>
<td>248</td>
</tr>
<tr>
<td>richValue</td>
<td>248</td>
</tr>
<tr>
<td>rotation</td>
<td>250</td>
</tr>
<tr>
<td>strokeColor</td>
<td>250</td>
</tr>
<tr>
<td>style</td>
<td>250</td>
</tr>
<tr>
<td>submitName</td>
<td>251</td>
</tr>
<tr>
<td>textColor</td>
<td>251</td>
</tr>
<tr>
<td>textFont</td>
<td>251</td>
</tr>
<tr>
<td>textSize</td>
<td>253</td>
</tr>
<tr>
<td>type</td>
<td>253</td>
</tr>
<tr>
<td>userName</td>
<td>253</td>
</tr>
<tr>
<td>value</td>
<td>253</td>
</tr>
<tr>
<td>valueAsString</td>
<td>254</td>
</tr>
<tr>
<td>browseForFileToSubmit</td>
<td>254</td>
</tr>
<tr>
<td>buttonGetCaption</td>
<td>255</td>
</tr>
<tr>
<td>buttonGetIcon</td>
<td>256</td>
</tr>
<tr>
<td>buttonImportIcon</td>
<td>256</td>
</tr>
<tr>
<td>buttonSetCaption</td>
<td>257</td>
</tr>
<tr>
<td>buttonSetIcon</td>
<td>258</td>
</tr>
<tr>
<td>checkThisBox</td>
<td>259</td>
</tr>
<tr>
<td>clearItems</td>
<td>260</td>
</tr>
<tr>
<td>defaultIsChecked</td>
<td>260</td>
</tr>
<tr>
<td>deleteItemAt</td>
<td>261</td>
</tr>
<tr>
<td>getArray</td>
<td>262</td>
</tr>
<tr>
<td>getItemAt</td>
<td>262</td>
</tr>
<tr>
<td>getLock</td>
<td>263</td>
</tr>
<tr>
<td>insertItemAt</td>
<td>263</td>
</tr>
<tr>
<td>isBoxChecked</td>
<td>264</td>
</tr>
<tr>
<td>isDefaultChecked</td>
<td>265</td>
</tr>
<tr>
<td>setAction</td>
<td>265</td>
</tr>
<tr>
<td>setFocus</td>
<td>266</td>
</tr>
<tr>
<td>setItems</td>
<td>267</td>
</tr>
<tr>
<td>setLock</td>
<td>268</td>
</tr>
<tr>
<td>signatureGetSeedValue</td>
<td>268</td>
</tr>
<tr>
<td>signatureInfo</td>
<td>269</td>
</tr>
<tr>
<td>signatureSetSeedValue</td>
<td>270</td>
</tr>
<tr>
<td>signatureSign</td>
<td>273</td>
</tr>
<tr>
<td>signatureValidate</td>
<td>275</td>
</tr>
<tr>
<td>FullScreen Object</td>
<td>276</td>
</tr>
<tr>
<td>FullScreen Properties</td>
<td>276</td>
</tr>
<tr>
<td>backgroundColor</td>
<td>276</td>
</tr>
<tr>
<td>clickAdvances</td>
<td>276</td>
</tr>
<tr>
<td>cursor</td>
<td>277</td>
</tr>
<tr>
<td>defaultTransition</td>
<td>277</td>
</tr>
<tr>
<td>escapeExits</td>
<td>277</td>
</tr>
<tr>
<td>isFullScreen</td>
<td>277</td>
</tr>
<tr>
<td>loop</td>
<td>278</td>
</tr>
<tr>
<td>timeDelay</td>
<td>278</td>
</tr>
</tbody>
</table>
transitions ..................................................... 278
usePageTiming ............................................. 278
useTimer ..................................................... 279
Global Object .................................................. 279
Creating Global Properties ................................ 279
Deleting Global Properties ................................ 279
Global Methods .............................................. 280
setPersistent ............................................... 280
subscribe ..................................................... 281
Icon Generic Object ......................................... 282
Icon Stream Generic Object .............................. 282
Identity Object .............................................. 282
Identity Properties .......................................... 283
corporation .................................................... 283
email ............................................................ 283
loginName ..................................................... 283
name ............................................................ 283
Index Object .................................................. 283
Index Properties ............................................. 284
available ..................................................... 284
name ............................................................ 284
path ............................................................ 284
selected ....................................................... 284
Index Methods ............................................... 285
build ........................................................... 285
Link Object ................................................... 285
Link Properties .............................................. 286
borderColor .................................................. 286
borderWidth ................................................ 286
highlightMode .............................................. 286
rect ............................................................. 286
Link Methods ............................................... 287
setAction ..................................................... 287
OCG Object .................................................. 287
OCG Properties .............................................. 287
name ............................................................ 287
state ............................................................ 288
OCG Methods ............................................... 288
setAction ..................................................... 288
PlugIn Object ................................................. 289
PlugIn Properties .......................................... 289
certified ....................................................... 289
loaded ........................................................ 289
name ............................................................ 289
path ............................................................ 290
### Contents

**Report Object** ................................................................. 305

**RDN Generic Object** ......................................................... 304

**Report Properties** .......................................................... 305

**PrintParams Properties** .........................................................

- `absIndent` .......................................................... 305
- `color` ............................................................. 305
- `size` ............................................................... 306
- `style` ............................................................... 306

**Report Methods** ..............................................................

- `breakPage` .............................................................. 307
- `divide` ............................................................... 307
- `indent` ............................................................... 307
- `outdent` .............................................................. 308
- `open` ................................................................. 308
- `save` ................................................................. 309
- `mail` ................................................................. 309
- `Report` ............................................................... 310
- `writeText` ............................................................ 310

**printParams Object** ............................................................ 290

**PrintParams Properties** ..................................................

- `binaryOK` ............................................................. 291
- `bitmapDPI` ............................................................. 291
- `colorOverride` ......................................................... 291
- `colorProfile` .......................................................... 292
- `constants` ............................................................ 292
- `downloadFarEastFonts` .................................................. 293
- `fileName` ............................................................ 293
- `firstPage` ............................................................. 294
- `flags` ................................................................. 294
- `fontPolicy` ............................................................. 296
- `gradientDPI` ............................................................ 296
- `interactive` ............................................................ 297
- `lastPage` .............................................................. 297
- `pageHandling` ........................................................... 298
- `pageSubset` ............................................................. 298
- `printAsImage` ............................................................ 299
- `printContent` ............................................................ 299
- `printerName` ............................................................ 300
- `psLevel` ............................................................... 300
- `rasterFlags` ............................................................. 300
- `reversePages` ............................................................ 302
- `tileLabel` ............................................................. 302
- `tileMark` .............................................................. 302
- `tileOverlap` ............................................................. 303
- `tileScale` .............................................................. 303
- `transparencyLevel` ....................................................... 303
- `usePrinterCRD` ........................................................... 303
- `useT1Conversion` .......................................................... 304

**RDN Generic Object** .......................................................... 304

**Report Object** ............................................................... 305

**Report Properties** ..........................................................

- `absIndent` ............................................................. 305
- `color` ............................................................... 305
- `size` ............................................................... 306
- `style` ............................................................... 306

**Report Methods** ..............................................................

- `breakPage` .............................................................. 307
- `divide` ............................................................... 307
- `indent` ............................................................... 307
- `outdent` .............................................................. 308
- `open` ................................................................. 308
- `save` ................................................................. 309
- `mail` ................................................................. 309
- `Report` ............................................................... 310
- `writeText` ............................................................ 310

**version** ........................................................................ 290

**printParams Object** .......................................................... 290

**PrintParams Properties** ..................................................

- `binaryOK` ............................................................. 291
- `bitmapDPI` ............................................................. 291
- `colorOverride` ......................................................... 291
- `colorProfile` .......................................................... 292
- `constants` ............................................................. 292
- `downloadFarEastFonts` .................................................. 293
- `fileName` ............................................................ 293
- `firstPage` ............................................................. 294
- `flags` ................................................................. 294
- `fontPolicy` ............................................................. 296
- `gradientDPI` ............................................................ 296
- `interactive` ............................................................ 297
- `lastPage` .............................................................. 297
- `pageHandling` ........................................................... 298
- `pageSubset` ............................................................. 298
- `printAsImage` ............................................................ 299
- `printContent` ............................................................ 299
- `printerName` ............................................................ 300
- `psLevel` ............................................................... 300
- `rasterFlags` ............................................................. 300
- `reversePages` ............................................................ 302
- `tileLabel` ............................................................. 302
- `tileMark` .............................................................. 302
- `tileOverlap` ............................................................. 303
- `tileScale` .............................................................. 303
- `transparencyLevel` ....................................................... 303
- `usePrinterCRD` ........................................................... 303
- `useT1Conversion` .......................................................... 304

**RDN Generic Object** .......................................................... 304

**Report Object** ............................................................... 305

**Report Properties** ..........................................................

- `absIndent` ............................................................. 305
- `color` ............................................................... 305
- `size` ............................................................... 306
- `style` ............................................................... 306

**Report Methods** ..............................................................

- `breakPage` .............................................................. 307
- `divide` ............................................................... 307
- `indent` ............................................................... 307
- `outdent` .............................................................. 308
- `open` ................................................................. 308
- `save` ................................................................. 309
- `mail` ................................................................. 309
- `Report` ............................................................... 310
- `writeText` ............................................................ 310
Row Generic Object ............................................................... 311
Search Object ........................................................................... 312
  Search Properties ................................................................... 312
    available ........................................................................ 312
    docInfo ......................................................................... 312
    docText ......................................................................... 313
    docXMP ......................................................................... 313
    bookmarks .................................................................... 313
    ignoreAsianCharacterWidth ........................................... 313
    indexes .......................................................................... 313
    jpegExif ........................................................................ 314
    legacySearch ................................................................. 314
    markup ......................................................................... 314
    matchCase .................................................................... 314
    matchWholeWord .......................................................... 314
    maxDocs ....................................................................... 315
    proximity ....................................................................... 315
    refine ........................................................................... 315
    soundex ....................................................................... 315
    stem ............................................................................ 315
    thesaurus ...................................................................... 316
    wordMatching .................................................................. 316
  Search Methods ..................................................................... 316
    addIndex ....................................................................... 316
    getIndexForPath ............................................................ 317
    index ............................................................................ 317
    removeIndex .................................................................. 318
Security Object ......................................................................... 318
  Security Properties ............................................................ 319
    handlers ....................................................................... 319
    validateSignaturesOnOpen .............................................. 319
  Security Methods ............................................................... 319
    chooseRecipientsDialog ................................................ 319
    getHandler .................................................................... 322
    exportToFile .................................................................. 323
    importFromFile ............................................................. 323
SecurityHandler Object .......................................................... 324
  SecurityHandler Properties ................................................. 325
    appearances ................................................................... 325
    digitalIDs ..................................................................... 325
    directories ..................................................................... 326
    directoryHandlers ........................................................ 327
    isLoggedIn ...................................................................... 327
    loginName ..................................................................... 327
    loginPath ...................................................................... 327
    name ............................................................................ 328
    signAuthor .................................................................... 328
    signFDF ....................................................................... 328
    signInvisible .................................................................. 329
### Contents

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Properties/Methods</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SignatureInfo Object</td>
<td>signValidate, signVisible, uiName, setPasswordTimeout</td>
<td>329</td>
</tr>
<tr>
<td>SecurityHandler Methods</td>
<td>login, logout, newDirectory, newUser, connect, request, response</td>
<td>329</td>
</tr>
<tr>
<td>SOAP Object</td>
<td>wireDump, connect, request, response, streamDecode, streamEncode, streamFromString, stringFromString</td>
<td>343</td>
</tr>
<tr>
<td>Sound Object</td>
<td>name, play, pause, stop</td>
<td>353</td>
</tr>
<tr>
<td>Span Object</td>
<td>alignment, fontFamily, fontStretch, fontSize, fontWeight, fontStyle, text, textColor, textSize, strikethrough, subscript, superscript, underline, underline</td>
<td>354</td>
</tr>
<tr>
<td>Spell Object</td>
<td>available, dictionaryNames</td>
<td>358</td>
</tr>
</tbody>
</table>

- **Span Properties**: alignment, fontFamily, fontStretch, fontSize, fontWeight, fontStyle, text, textColor, textSize, strikethrough, subscript, superscript, underline
- **Spell Properties**: available, dictionaryNames
<table>
<thead>
<tr>
<th>Class</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionaryOrder</td>
<td></td>
</tr>
<tr>
<td>domainNames</td>
<td></td>
</tr>
<tr>
<td>languages</td>
<td></td>
</tr>
<tr>
<td>languageOrder</td>
<td></td>
</tr>
<tr>
<td>Spell Methods</td>
<td></td>
</tr>
<tr>
<td>addDictionary</td>
<td></td>
</tr>
<tr>
<td>addWord</td>
<td></td>
</tr>
<tr>
<td>check</td>
<td></td>
</tr>
<tr>
<td>checkText</td>
<td></td>
</tr>
<tr>
<td>checkWord</td>
<td></td>
</tr>
<tr>
<td>customDictionaryClose</td>
<td></td>
</tr>
<tr>
<td>customDictionaryCreate</td>
<td></td>
</tr>
<tr>
<td>customDictionaryDelete</td>
<td></td>
</tr>
<tr>
<td>customDictionaryExport</td>
<td></td>
</tr>
<tr>
<td>customDictionaryOpen</td>
<td></td>
</tr>
<tr>
<td>ignoreAll</td>
<td></td>
</tr>
<tr>
<td>removeDictionary</td>
<td></td>
</tr>
<tr>
<td>removeWord</td>
<td></td>
</tr>
<tr>
<td>userWords</td>
<td></td>
</tr>
<tr>
<td>Statement Object</td>
<td></td>
</tr>
<tr>
<td>Statement Properties</td>
<td></td>
</tr>
<tr>
<td>columnCount</td>
<td></td>
</tr>
<tr>
<td>rowCount</td>
<td></td>
</tr>
<tr>
<td>Statement Methods</td>
<td></td>
</tr>
<tr>
<td>execute</td>
<td></td>
</tr>
<tr>
<td>getColumn</td>
<td></td>
</tr>
<tr>
<td>getColumnArray</td>
<td></td>
</tr>
<tr>
<td>getRow</td>
<td></td>
</tr>
<tr>
<td>nextRow</td>
<td></td>
</tr>
<tr>
<td>TableInfo Generic Object</td>
<td></td>
</tr>
<tr>
<td>Template Object</td>
<td></td>
</tr>
<tr>
<td>Template Properties</td>
<td></td>
</tr>
<tr>
<td>hidden</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td></td>
</tr>
<tr>
<td>Template Methods</td>
<td></td>
</tr>
<tr>
<td>spawn</td>
<td></td>
</tr>
<tr>
<td>Thermometer Object</td>
<td></td>
</tr>
<tr>
<td>Thermometer Properties</td>
<td></td>
</tr>
<tr>
<td>cancelled</td>
<td></td>
</tr>
<tr>
<td>duration</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td></td>
</tr>
<tr>
<td>text</td>
<td></td>
</tr>
<tr>
<td>Thermometer Methods</td>
<td></td>
</tr>
<tr>
<td>begin</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>TTS Object</td>
<td></td>
</tr>
<tr>
<td>TTS Properties</td>
<td></td>
</tr>
</tbody>
</table>
Contents

Available ................................................................. 380
numSpeakers ............................................................ 380
pitch ................................................................. 380
soundCues .............................................................. 380
speaker ............................................................... 381
speechCues ............................................................ 381
speechRate ............................................................. 381
volume ................................................................. 381
TTS Methods ............................................................ 381
getNthSpeakerName .................................................. 381
pause ................................................................. 382
qSilence ............................................................... 382
qSound ............................................................... 382
qText ................................................................. 383
reset ................................................................. 383
resume ............................................................... 383
stop ................................................................. 384
talk ................................................................. 384
this Object .............................................................. 384
Variable and Function Name Conflicts .............................. 385
Util Object ............................................................. 385
Util Methods ........................................................... 386
printf ............................................................... 386
printf ............................................................... 387
printx ............................................................... 389
scand ............................................................... 390
spansToXML ........................................................ 391
xmlToSpans .......................................................... 391
XFA Object ............................................................ 392

New Features and Changes ........................................... 395

Acrobat 6.0 Changes .................................................. 395
  Introduced in Acrobat 6.0 .......................................... 395
  Modified in Acrobat 6.0 .......................................... 403
  Deprecated in Acrobat 6.0 ........................................ 405
  Introduced in Acrobat 6.0.2 ...................................... 405

Acrobat 5.0 Changes .................................................. 406
  Introduced in Acrobat 5.0 .......................................... 406
  Modified in Acrobat 5.0 .......................................... 413
  Deprecated in Acrobat 5.0 ........................................ 414
  Modified in Acrobat 5.0.0 ........................................ 414
  Modified in Adobe 5.1 Reader .................................... 415

Available ................................................................. 380
numSpeakers ............................................................ 380
pitch ................................................................. 380
soundCues .............................................................. 380
speaker ............................................................... 381
speechCues ............................................................ 381
speechRate ............................................................. 381
volume ................................................................. 381
TTS Methods ............................................................ 381
getNthSpeakerName .................................................. 381
pause ................................................................. 382
qSilence ............................................................... 382
qSound ............................................................... 382
qText ................................................................. 383
reset ................................................................. 383
resume ............................................................... 383
stop ................................................................. 384
talk ................................................................. 384
this Object .............................................................. 384
Variable and Function Name Conflicts .............................. 385
Util Object ............................................................. 385
Util Methods ........................................................... 386
printf ............................................................... 386
printf ............................................................... 387
printx ............................................................... 389
scand ............................................................... 390
spansToXML ........................................................ 391
xmlToSpans .......................................................... 391
XFA Object ............................................................ 392

New Features and Changes ........................................... 395

Acrobat 6.0 Changes .................................................. 395
  Introduced in Acrobat 6.0 .......................................... 395
  Modified in Acrobat 6.0 .......................................... 403
  Deprecated in Acrobat 6.0 ........................................ 405
  Introduced in Acrobat 6.0.2 ...................................... 405

Acrobat 5.0 Changes .................................................. 406
  Introduced in Acrobat 5.0 .......................................... 406
  Modified in Acrobat 5.0 .......................................... 413
  Deprecated in Acrobat 5.0 ........................................ 414
  Modified in Acrobat 5.0.0 ........................................ 414
  Modified in Adobe 5.1 Reader .................................... 415

Available ................................................................. 380
numSpeakers ............................................................ 380
pitch ................................................................. 380
soundCues .............................................................. 380
speaker ............................................................... 381
speechCues ............................................................ 381
speechRate ............................................................. 381
volume ................................................................. 381
TTS Methods ............................................................ 381
getNthSpeakerName .................................................. 381
pause ................................................................. 382
qSilence ............................................................... 382
qSound ............................................................... 382
qText ................................................................. 383
reset ................................................................. 383
resume ............................................................... 383
stop ................................................................. 384
talk ................................................................. 384
this Object .............................................................. 384
Variable and Function Name Conflicts .............................. 385
Util Object ............................................................. 385
Util Methods ........................................................... 386
printf ............................................................... 386
printf ............................................................... 387
printx ............................................................... 389
scand ............................................................... 390
spansToXML ........................................................ 391
xmlToSpans .......................................................... 391
XFA Object ............................................................ 392

New Features and Changes ........................................... 395

Acrobat 6.0 Changes .................................................. 395
  Introduced in Acrobat 6.0 .......................................... 395
  Modified in Acrobat 6.0 .......................................... 403
  Deprecated in Acrobat 6.0 ........................................ 405
  Introduced in Acrobat 6.0.2 ...................................... 405

Acrobat 5.0 Changes .................................................. 406
  Introduced in Acrobat 5.0 .......................................... 406
  Modified in Acrobat 5.0 .......................................... 413
  Deprecated in Acrobat 5.0 ........................................ 414
  Modified in Acrobat 5.0.0 ........................................ 414
  Modified in Adobe 5.1 Reader .................................... 415
Preface

Introduction

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

What’s In This Document

– Acrobat JavaScript Scripting Reference: Describes in detail all objects, properties and methods within the Acrobat extension to JavaScript, and gives code examples
– New Features and Changes: Summarizes the new features and changes introduced in Adobe Acrobat 6.0 and in Adobe Acrobat 5.0.

Document Conventions

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

Font Conventions Used in This Book

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

<table>
<thead>
<tr>
<th>Font</th>
<th>Used for</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>monospaced</td>
<td>Paths and filenames</td>
<td>C:\templates\mytmpl.fm</td>
</tr>
<tr>
<td></td>
<td>Code examples set off from plain text</td>
<td>These are variable declarations: AVMenu, commandMenu, helpMenu;</td>
</tr>
</tbody>
</table>
### Document Conventions

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

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

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

<table>
<thead>
<tr>
<th>Version or Deprecated</th>
<th>Save and Preferences</th>
<th>Security</th>
<th>Reader</th>
<th>Approval</th>
<th>Acrobat Pro</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0</td>
<td>D S C X P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

### Column 1: Version or Deprecated

- **A number indicates the version of the software in which a property or method became available. If the number is specified, then the property or method is available only in versions of the Acrobat software greater than or equal to that number.**

  For Adobe Acrobat 6.0, there are some compatibility issues with older versions. Before accessing this property or method, the script should check that the forms version is greater than or equal to that number to ensure backward compatibility. For example:

  ```javascript
  if (typeof app.formsVersion != "undefined" && app.formsVersion >= 6.0)
  {
    // Perform version specific operations.
  }
  ```

  If the first column is blank, no compatibility checking is necessary.

- **As the Acrobat JavaScript extensions have evolved, some properties and methods have been superseded by other more flexible or appropriate properties and methods. The use of these older methods are discouraged and marked by X in the version column.**

### Column 2: Save and Preferences

- **Using this property or method dirties (modifies) the PDF document. If the document is subsequently saved, the effects of this method are saved as well.**
### Column 2: Save and Preferences

| ⌐ | The preferences symbol indicates that even though this property does not change the document, it can permanently change a user’s application preferences. |

### Column 3: Security

| ⌛ | This property or method may only be available during certain events for security reasons (for example, batch processing, application start, or execution within the console). See the Event Object for details of the various Acrobat events. |

### Column 4: Availability in Adobe Reader

- If the column is blank, the property or method is allowed in any version of the Adobe Reader.
- ☒ The property or method is not allowed in any version of the Adobe Reader.
- ☑ The property or method is allowed only in version 5.1, or later, of the Adobe Reader, not in versions 5.05 or below.
- ☑ The property or method can be accessed only in the Adobe 5.1 Reader depending on document rights (see Modified in Adobe 5.1 Reader).

- **F** requires Advanced Forms Features rights
- **C** requires the right to manipulate Comments.
- **S** requires document Save rights.

### Column 5: Availability in Adobe Acrobat Approval

- If the column is blank, the property or method is allowed in Acrobat Approval.
- ☒ The property or method is not allowed in Acrobat Approval.

### Column 6: Availability in Adobe Acrobat

- If the column is blank, the property or method is allowed in Acrobat Std and Acrobat Pro.
- ⌐ The property or method is available only in Acrobat Pro.
Other Sources of Information

Online Help

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

- [http://partners.adobe.com/asn/acrobat/](http://partners.adobe.com/asn/acrobat/) — A listing of Acrobat resources for developers. This listing includes the following:
  - [http://www.adobe.com/support/database.html](http://www.adobe.com/support/database.html) — In addition to the forums, Adobe maintains a searchable support database with answers to commonly asked questions.

References

**Core JavaScript 1.5 Documentation**

Complete documentation for JavaScript 1.5, the version used by Acrobat 6.0, is available on the web at [http://devedge.netscape.com/central/javascript/](http://devedge.netscape.com/central/javascript/).


**Adobe Web Documentation**

**PDF Reference, Fourth Edition, Version 1.5.** The PDF Reference provides a description of the PDF file format and is intended primarily for application developers wishing to develop PDF producer applications that create PDF files directly. [http://partners.adobe.com/asn/tech/pdf/specifications.jsp](http://partners.adobe.com/asn/tech/pdf/specifications.jsp)

**Acrobat Core API Overview, Technical Note #5190.** Gives an overview of the objects and methods provided by the plug-in API of the Acrobat viewer. This document is available with

*Acrobat Core API Reference, Technical Note #5191.* Describes in detail the objects and methods provided by the Acrobat viewer’s plug-in API. This document is available with the Adobe Acrobat Plug-ins SDK CD-ROM or on the Adobe Web site http://partners.adobe.com/asn/acrobat/docs.jsp.


ADBC Object

The Acrobat Database Connectivity (ADBC) plug-in allows JavaScripts inside of PDF documents to access databases through a consistent object model. The object model is based on general principles used in the object models for the ODBC and JDBC APIs. Like ODBC and JDBC, ADBC is a means of communicating with a database through SQL or Structured Query Language.

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

NOTE: (Security ⑤): It is important to note that ADBC provides no security for any of the databases it is programmed to access. It is the responsibility of the database administrator to keep all data secure.

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

<table>
<thead>
<tr>
<th>Object</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADBC Object</td>
<td>An object through which a list of accessible databases can be obtained and a connection can be made to one of them.</td>
</tr>
<tr>
<td>Connection Object</td>
<td>An object through which a list of tables in the connected database can be obtained.</td>
</tr>
<tr>
<td>Statement Object</td>
<td>An object through which SQL statements can be executed and rows retrieved based on the query.</td>
</tr>
</tbody>
</table>
The `ADBC` object has the following constant properties representing various SQL Types:

<table>
<thead>
<tr>
<th>Constant property name</th>
<th>value</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLT_BIGINT</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>SQLT_BINARY</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SQLT_BIT</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SQLT_CHAR</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>SQLT_DATE</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>SQLT_DECIMAL</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>SQLT_DOUBLE</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>SQLT_FLOAT</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>SQLT_INTEGER</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>SQLT_LONGVARBINARY</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>SQLT_LONGVARCHAR</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>SQLT_NUMERIC</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>SQLT_REAL</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>SQLT_SMALLINT</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>SQLT_TIME</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>SQLT_TIMESTAMP</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>SQLT_TINYINT</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>SQLT_VARBINARY</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>SQLT_VARCHAR</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>SQLT_NCHAR</td>
<td>19</td>
<td>6.0</td>
</tr>
<tr>
<td>SQLT_NVARCHAR</td>
<td>20</td>
<td>6.0</td>
</tr>
</tbody>
</table>
The type properties of the Column Generic Object and ColumnInfo Generic Object use these properties.

### JavaScript Types

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

<table>
<thead>
<tr>
<th>Constant Property Name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>0</td>
</tr>
<tr>
<td>String</td>
<td>1</td>
</tr>
<tr>
<td>Binary</td>
<td>2</td>
</tr>
<tr>
<td>Boolean</td>
<td>3</td>
</tr>
<tr>
<td>Time</td>
<td>4</td>
</tr>
<tr>
<td>Date</td>
<td>5</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>6</td>
</tr>
</tbody>
</table>

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

### ADBC Methods

#### getDataSourceList

Obtains information about the databases accessible from a given system.

**Parameters**

None
Returns

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

Example

See newConnection for an example.

newConnection

Creates a connection object associated with the specified database. Optionally, you can supply a user ID and a password.

NOTE: (Security): It is possible to connect to a database using a connection string with no DSN, but this is only permitted, beginning with Acrobat 6.0, during a console, batch or menu event

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cDSN</td>
<td>The data source name (DSN) of the database.</td>
</tr>
<tr>
<td>cUID</td>
<td>(optional) User ID.</td>
</tr>
<tr>
<td>cPWD</td>
<td>(optional) Password.</td>
</tr>
</tbody>
</table>

Returns

A Connection Object, or null on failure.

Example

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

try {
    /* now display them, while searching for the one named "q32000data". */
    var DB = "", msg = "";
    if (aList != null) {
        for (var i=0; i < aList.length; i++) {
            console.println("Name: "+aList[i].name);
            console.println("Description: "+aList[i].description);
            // and choose one of interest
            if (aList[i].name=="q32000data")
                DB = aList[i].name;
        }
    }
}
// did we find the database?
if (DB != "") {
    // yes, establish a connection.
    console.println("The requested database has been found!");
    var Connection = ADBC.newConnection(DB);
    if (Connection == null) throw "Not Connected!";
} else
    // no, display message to console.
    throw "Could not find the requested database."
} catch (e) {
    console.println(e);
}

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

---

**AlternatePresentation Object**

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

**AlternatePresentation Properties**

**active**

<table>
<thead>
<tr>
<th>Version</th>
<th>Required</th>
<th>Default</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0</td>
<td></td>
<td>true</td>
<td>Boolean</td>
<td>R.</td>
<td>true if presentation is currently active and false otherwise. When a presentation is active it controls how the document that owns it is displayed on the screen.</td>
</tr>
</tbody>
</table>

**type**

<table>
<thead>
<tr>
<th>Version</th>
<th>Required</th>
<th>Default</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0</td>
<td></td>
<td></td>
<td>String</td>
<td>R.</td>
<td>The type of the alternate presentation. Currently, the only supported type is &quot;SlideShow&quot;.</td>
</tr>
</tbody>
</table>
AlternatePresentation Methods

**start**

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

**Parameters**

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

**Returns**

Nothing

**Example**

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

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

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

**stop**

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

**Parameters**

None

**Returns**

Nothing
Example

Assume `oMySlideShow` is an `AlternatePresentations` object. See `start` for a related example.

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

Annot Object

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

The user interface in Acrobat refers to annotations as comments.

Annotation Types

Annotations are of different types, as reflected in the `type` property. The types of annotations available are:

- Circle
- FileAttachment
- FreeText
- Highlight
- Ink
- Line
- Oval
- Rectangle
- Polygon
- Sound
- Square
- Squiggly
- Stamp
- StrikeOut
- Text

Some properties are used only with particular types of annotations, as shown in the following table.

<table>
<thead>
<tr>
<th>Annotation Types</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>All types</td>
<td><code>type</code></td>
</tr>
<tr>
<td></td>
<td><code>name</code></td>
</tr>
<tr>
<td></td>
<td><code>rect</code></td>
</tr>
<tr>
<td></td>
<td><code>contents</code></td>
</tr>
<tr>
<td></td>
<td><code>page</code></td>
</tr>
<tr>
<td></td>
<td><code>modDate</code></td>
</tr>
<tr>
<td></td>
<td><code>author</code></td>
</tr>
<tr>
<td>Annotation Types</td>
<td>Properties</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Circle</td>
<td><code>point</code> <code>popupRect</code> <code>fillColor</code> <code>strokeColor</code> <code>width</code></td>
</tr>
<tr>
<td>FileAttachment</td>
<td><code>print</code> <code>attachIcon</code></td>
</tr>
<tr>
<td>FreeText</td>
<td><code>alignment</code> <code>fillColor</code> <code>rotate</code> <code>strokeColor</code> <code>textFont</code> <code>textSize</code> <code>width</code></td>
</tr>
<tr>
<td>Highlight</td>
<td><code>quads</code> <code>strokeColor</code> <code>point</code> <code>popupRect</code></td>
</tr>
<tr>
<td>Ink</td>
<td><code>gestures</code> <code>strokeColor</code> <code>point</code> <code>popupRect</code> <code>width</code></td>
</tr>
<tr>
<td>Line</td>
<td><code>points</code> <code>arrowBegin</code> <code>arrowEnd</code> <code>point</code> <code>popupRect</code> <code>fillColor</code> <code>strokeColor</code> <code>width</code></td>
</tr>
<tr>
<td>Sound</td>
<td><code>print</code> <code>soundIcon</code></td>
</tr>
<tr>
<td>Square</td>
<td><code>point</code> <code>popupRect</code> <code>fillColor</code> <code>strokeColor</code> <code>width</code></td>
</tr>
<tr>
<td>Squiggly</td>
<td><code>quads</code> <code>strokeColor</code> <code>point</code> <code>popupRect</code></td>
</tr>
<tr>
<td>Stamp</td>
<td><code>point</code> <code>popupRect</code> <code>AP</code></td>
</tr>
<tr>
<td>StrikeOut</td>
<td><code>quads</code> <code>strokeColor</code> <code>point</code> <code>popupRect</code></td>
</tr>
<tr>
<td>Text</td>
<td><code>print</code> <code>noteIcon</code> <code>point</code> <code>popupRect</code></td>
</tr>
<tr>
<td>Underline</td>
<td><code>quads</code> <code>strokeColor</code> <code>point</code> <code>popupRect</code></td>
</tr>
</tbody>
</table>
Annotation Access from JavaScript

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

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

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

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

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

---

### Annot Properties

#### alignment

Controls the alignment of the text for a FreeText annotation.

<table>
<thead>
<tr>
<th>Alignment</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left aligned</td>
<td>0</td>
</tr>
<tr>
<td>Centered</td>
<td>1</td>
</tr>
<tr>
<td>Right aligned</td>
<td>2</td>
</tr>
</tbody>
</table>

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

#### AP

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

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

Type: String Access: R/W Annots: Stamp

Example

```javascript
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"
});
```

NOTE: The name of a particular stamp can be found by opening the PDF file in the Stamps folder that contains the stamp in question. Choose File > Form > Page Templates to see a listing of all appearances and their names. For a list of stamp names currently in use in the document, see doc.icons.

arrowBegin

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

Circle
ClosedArrow
Diamond
None (default)
OpenArrow
Square

Type: String Access: R/W Annots: Line

Example

See setProps.
**arrowEnd**

<table>
<thead>
<tr>
<th>5.0</th>
</tr>
</thead>
</table>

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

- Circle
- ClosedArrow
- Diamond
- None (default)
- OpenArrow
- Square


**Example**

See `setProps`.

**attachIcon**

<table>
<thead>
<tr>
<th>5.0</th>
</tr>
</thead>
</table>

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

- Paperclip
- PushPin (default)
- Graph
- Tag

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

**author**

<table>
<thead>
<tr>
<th>5.0</th>
</tr>
</thead>
</table>

Gets or sets the author of the annotation.

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

**Example**

See `contents`.
borderEffectIntensity

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

Type: Number  Access: R/W
Annots: Rectangle, Polygon, Oval.

borderEffectStyle

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

Type: String  Access: R/W
Annots: Rectangle, Polygon, Oval.

contents

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

NOTE: Getting and setting of this property in Acrobat 5.1 Reader depends on Comments document rights.

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

Example

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

See also addAnnot.
doc

| 5.0 | D | C | X |

Returns the Doc Object of the document in which the annotation resides.

*Type:* doc object  *Access:* R  *Annots:* all.

**Example**

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

fillColor

| 5.0 | D | C | X |

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

*Type:* Color  *Access:* R/W  *Annots:* Circle, Square, Line, FreeText.

**Example**

```javascript
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]
});
```
gestures

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

Type: Array  Access: R/W  Annots: Ink

hidden

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

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

inReplyTo

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

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

modDate

Returns the last modification date for the annotation.

Type: Date  Access: R  Annots: all.

Example

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

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

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

**Example**

```javascript
// 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"
```

**noteIcon**

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

- Comment
- Help
- Insert
- Key
- Note (default)
- NewParagraph
- Paragraph

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

**Example**

See `contents`.

**noView**

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

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

**Example**

See `toggleNoView`.
**page**

The page on which the annotation resides.

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

**Example**

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

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

**point**

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

*Type: Array*  
*Access: R/W*  
*Annots: Text, Sound, FileAttachment.*

**Example**

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

See also `addAnnot` and `noteIcon`.

**points**

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

*Type: Array*  
*Access: R/W*  
*Annots: Line*
Example

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

See addAnnot, arrowBegin, arrowEnd and setProps.

**popupOpen**

![popupOpen](image)

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

*Type: Boolean Access: R/W Annots: all except FreeText, Sound, FileAttachment.*

**Example**

See the print.

**popupRect**

![popupRect](image)

An array of four numbers [xll, yll, xur, yur] specifying the lower-left x, lower-left y, upper-right x and upper-right y coordinates—in default user space—of the rectangle of the popup annotation associated with a parent annotation and defines the location of the popup annotation on the page.

*Type: Array Access: R/W Annots: all except FreeText, Sound, FileAttachment.*

**Example**

See the print.
**print**

| 5.0 | O | C | X |

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

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

**quads**

| 5.0 | O | C | X |

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


**Example**

See `getPageNthWordQuads` for an example.

**rect**

| 5.0 | O | C | X |

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

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

**readOnly**

| 5.0 | O | C | X |

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

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

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

Type: Array of Span Objects
Access: R/W
Annots: all.

Example

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

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

var spans = new Array();
spans[0] = new Object();
spans[0].text = "Attention:\r"
spans[0].textColor = color.blue;
spans[0].textSize = 18;
spans[1] = new Object();
spans[1].text = "Adobe Acrobat 6.0\r"
spans[1].textColor = color.red;
spans[1].textSize = 20;
spans[1].alignment = "center";
spans[2] = new Object();
spans[2].text = "will soon be here!"
spans[2].textColor = color.green;
spans[2].fontStyle = "italic"
spans[2].underline = true;
spans[2].alignment = "right";

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

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

5.0 D C X

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

Type: Integer Access: R/W Annots: FreeText.

strokeColor

5.0 D C X

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

Type: Color Access: R/W Annots: all.

Example

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

textFont

5.0 D C X

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

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

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

Example

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

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

textSize

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

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

Type: Number Access: R/W Annots: FreeText.

Example
See textFont.

toggleNoView

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

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

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

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

- Circle
- FileAttachment
- FreeText
- Highlight
- Ink
- Line
- Oval
- Rectangle
- Polygon
- Sound
- Square
- Squiggly
- Stamp
- StrikeOut
- Text
- Underline

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

soundIcon

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

*Type: String*  
*Access: R/W*  
*Annots: Sound*

width

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

*Type: Number*  
*Access: R/W*  
*Annots: Square, Circle, Line, Ink, FreeText.*
Annot Methods

**destroy**

```
5.0 [C] D C X
```

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

**Parameters**

None

**Returns**

Nothing

**Example**

```javascript
// 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 [C] D A X
```

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

**Parameters**

None

**Returns**

This method returns an object literal of the properties of the annotation. The object literal is just like the one passed to **addAnnot**.

**Example**

```javascript
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;
}

**getStateInModel**

6.0

Gets the current state of the *annot* in the context of a state model. See also `transitionToState`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cStateModel</code></td>
<td>The state model to determine the state of the <em>annot</em>.</td>
</tr>
</tbody>
</table>

**Returns**

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

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

**Exceptions**

None

**setProps**

5.0

Sets many properties of the annotation simultaneously.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>objectLiteral</code></td>
<td>A generic object, which specifies the properties of the <em>annot</em> object annotation, such as <code>type</code>, <code>rect</code>, and <code>page</code>, to be created. (This is the same as the parameter of <code>doc.addAnnot</code>.)</td>
</tr>
</tbody>
</table>

**Returns**

The *annot* object

**Example**

```javascript
var annot = this.addAnnot({type: "Line"})
annot.setProps({
```
transitionToState

Makes the state of the Annot \texttt{cState} by performing a state transition. The state transition is recorded in the audit trail of the Annot.

See also \texttt{getStateInModel}.

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

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{cStateModel}</td>
<td>The state model in which to perform the state transition. \texttt{cStateModel} must have been previously added by calling \texttt{addStateModel}.</td>
</tr>
<tr>
<td>\texttt{cState}</td>
<td>A valid state in the state model to transition to.</td>
</tr>
</tbody>
</table>

### Returns

Nothing

### Exceptions

None

### Example

```javascript
try {
  // Create a document
  var myDoc = app.newDoc();
  // Create an annot
  var myAnnot = myDoc.addAnnot({
    page: 0,
    type: "Text",
    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"
  })
```

point: [300,400],
   name: "myAnnot",
 });

// Create the state model
var myStates = new Object;
myStates["initial"] = {cUIName: "Haven't reviewed it"};
myStates["approved"] = {cUIName: "I approve"};
myStates["rejected"] = {cUIName: "Forget it"};
myStates["resubmit"] = {cUIName: "Make some changes"};

Collab.addStateModel({cName: "ReviewStates", cUIName: "My Review",
   oStates: myStates, Default: initial
});

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

} catch(e) { console.println(e);}

---

**App Object**

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

---

**App Properties**

**activeDocs**

5.0

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

**Note:** For version 5.0, this property returns an array containing the Doc Object for each active document open in the viewer. In version 5.0.5, this property was changed to return an array of Doc Objects of only those documents open in the viewer that have the doc.disclosed property set to true. The “Acrobat 5.0.5 Accessibility and Forms Patch” changed this behavior—and this is the behavior of activeDocs for Acrobat 6.0 or later—as follows: During a batch, console or menu event, activeDocs ignores the disclosed property and returns an array of Doc Objects of the active documents open in the viewer; during any other event, activeDocs returns an array of Doc Objects of only those active documents open in the viewer that have doc.disclosed set to true.
Type: Array  
Access: R.

**Example**

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

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

**calculate**

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

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

Type: Boolean  
Access: R/W.

**focusRect**

4.05

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

Type: Boolean  
Access: R/W.

**Example**

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

**formsVersion**

4.0

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

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

fromPDFConverters

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

Type: Array  Access: R.

Example
List all currently supported conversion ID strings for `doc.saveAs`.
for ( var i = 0; i < app.fromPDFConverters.length; i++)
    console.println(app.fromPDFConverters[i]);

fs

Returns the `FullScreen Object`, which can be used to access the fullscreen properties.

Type: object  Access: R.

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

See also `FullScreenObject.isFullScreen`.
fullscreen

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

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

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

*Type: Boolean  Access: R/W.*

**Example**

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

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

**language**

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

<table>
<thead>
<tr>
<th>String</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHS</td>
<td>Chinese Simplified</td>
</tr>
<tr>
<td>CHT</td>
<td>Chinese Traditional</td>
</tr>
<tr>
<td>DAN</td>
<td>Danish</td>
</tr>
<tr>
<td>DEU</td>
<td>German</td>
</tr>
<tr>
<td>ENU</td>
<td>English</td>
</tr>
<tr>
<td>ESP</td>
<td>Spanish</td>
</tr>
<tr>
<td>FRA</td>
<td>French</td>
</tr>
<tr>
<td>ITA</td>
<td>Italian</td>
</tr>
<tr>
<td>KOR</td>
<td>Korean</td>
</tr>
<tr>
<td><strong>String</strong></td>
<td><strong>Language</strong></td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>JPN</td>
<td>Japanese</td>
</tr>
<tr>
<td>NLD</td>
<td>Dutch</td>
</tr>
<tr>
<td>NOR</td>
<td>Norwegian</td>
</tr>
<tr>
<td>PTB</td>
<td>Brazilian Portuguese</td>
</tr>
<tr>
<td>SUO</td>
<td>Finnish</td>
</tr>
<tr>
<td>SVE</td>
<td>Swedish</td>
</tr>
</tbody>
</table>

**numPlugIns**

Indicates the number of plug-ins that have been loaded by Acrobat. See **plugIns** which supersedes this property in later versions.

*Type: Number*  
*Access: R.*

**openInPlace**

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

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

**Example**

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

**platform**

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

- WIN
- MAC
- UNIX

*Type: String*  
*Access: R.*
plugIns

Determines which plug-ins are currently installed in the viewer. Returns an array of PlugIn Objects.

Type: Array
Access: R.

Example

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

printColorProfiles

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

Type: Array of Strings
Access: R.

Example

Print out a listing of available printer color spaces.

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

printerNames

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

Type: Array of Strings
Access: R.

Example

Print out a listing of available printer color spaces.
var l = app.printerNames.length
for ( var i = 0; i < l; i++)
    console.println("(" + (i+1) + ") " + app.printerNames[i]);

**runtimeHighlight**

| 6.0 | P |

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

*Type: Boolean Access: R/W.*

**Example**

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

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

**runtimeHighlightColor**

| 6.0 | P |

Sets the color for runtime highlighting of form fields.

The value of `runtimeHighlightColor` is a color array, see the Color Object for details.

*Type: A color array Access: R/W.*

**Example**

```javascript
app.runtimeHighlight = true;
app.runtimeHighlightColor = color.red;
```

**thermometer**

| 6.0 |

Returns a Thermometer Object. The `thermometer` object is a combined status window/progress bar that indicates to the user that a lengthy operation is in progress.

*Type: object Access: R.*
Example

See the Thermometer Object for an example.

toolbar

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

Type: Boolean Access: R/W.

Example

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

toolbarHorizontal

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

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

Type: Boolean Access: R/W.

toolbarVertical

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

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

Type: Boolean Access: R/W.
**viewerType**

Determines if the running Adobe Reader, Acrobat Std or Acrobat Pro. Values are:

- Reader
- Exchange
- Exchange-Pro

*Type: String  Access: R.*

**viewerVariation**

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

- Reader
- Fill-In
- Business Tools
- Full

*Type: String  Access: R.*

**viewerVersion**

Indicates the version number of the current viewer.

*Type: Number  Access: R.*

---

**App Methods**

**addMenuItem**

Adds a menu item to the application.

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

See also the addSubMenu, execMenuItem, hideMenuItem, and listMenuItems.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The language independent name of the menu item. This language independent name is used to access the menu item for other methods (for example, <code>hideMenuItem</code>).</td>
</tr>
<tr>
<td>cUser</td>
<td>(optional) The user string (language dependent name) to display as the menu item name. If <code>cUser</code> is not specified then <code>cName</code> is used</td>
</tr>
<tr>
<td>cParent</td>
<td>The name of the parent menu item. Its submenu will have the new menu item added to it. If <code>cParent</code> has no submenu then an exception is thrown. Menu item names can be discovered with <code>listMenuItems</code>.</td>
</tr>
</tbody>
</table>
| nPos      | (optional) The position within the submenu to locate the new menu item. The default behavior is to append to the end of the submenu. Specifying `nPos` as 0 will add to the top of the submenu. Beginning with Acrobat 6.0, the value of `nPos` can also be the language independent name of a menu item. (Version 6.0) If the value `nPos` is a string, this string is interpreted as a named item in the menu (a language independent name of a menu item). The named item determines the position at which the new menu item is to be inserted. See `bPrepend` for additional details. **NOTE:** The `nPos` parameter is ignored in certain menus that are alphabetized. The alphabetized menus are:
  - The first section of `View > Navigational Tabs`.
  - The first section of `View > Toolbars`.
  - The first section of the `Advanced` submenu. **NOTE:** When `nPos` is a number, `nPos` is not obeyed in the `Tools` menu. A menu item introduced into the `Tools` menu comes in at the top of the menu. `nPos` will be obeyed when `nPos` is a string referencing another user-defined menu item. |
| cExec     | An expression string to evaluate when the menu item is selected by the user. |
| cEnable   | (optional) An expression string that determines whether or not to enable the menu item. The default is that the menu item is always enabled. This expression should set `event.rc` to `false` to disable the menu item. |
| cMarked   | (optional) An expression string that determines whether or not the menu item has a check mark next to it. Default is that the menu item is not marked. This expression should set `event.rc` to `false` to uncheck the menu item and true to check it. |
bPrepend (optional, version 6.0) Determines the position of the new menu item relative to the position specified by nPos. The default value is false. If bPrepend is true, the rules for insertion are as follows: If nPos is a string, the new item is placed before the named item; if nPos is a number, the new item is placed before the numbered item; if the named item can’t be found or nPos is not between zero and the number of items in the list, inclusive, then the new item is inserted as the first item in the menu (rather than at the end of the menu).

bPrepend is useful when the named item is the first item in a group.

Returns
Nothing

Example 1

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

Example 2 (version 6.0)

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

// insert after the "Close" item (the default behavior)
app.addMenuItem({ cName: "myItem1", cUser: "My Item 1", cParent: "File", cExec: "_myProc1()", nPos: "Close"});

// insert before the "Close" item, set bPrepend to true.

addSubMenu

5.0

Adds a menu item with a submenu to the application.
See also the addMenuItem, execMenuItem, hideMenuItem, and listMenuItems.

Note: (Security): This method can only be executed during application initialization or console events. See the Event Object for a discussion of Acrobat JavaScript events.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The language independent name of the menu item. This language independent name is used to access the menu item for hideMenuItem, for example.</td>
</tr>
<tr>
<td>cUser</td>
<td>(optional) The user string (language dependent name) to display as the menu item name. If cUser is not specified then cName is used.</td>
</tr>
<tr>
<td>cParent</td>
<td>The name of the parent menu item to receive the new submenu. Menu item names can be discovered with listMenuItems.</td>
</tr>
<tr>
<td>nPos</td>
<td>(optional) The position within the parent’s submenu to locate the new submenu. Default is to append to the end of the parent’s submenu. Specifying nPos as 0 will add to the top of the parent’s submenu.</td>
</tr>
</tbody>
</table>

**Note:** The nPos parameter is ignored in certain menus that are alphabetized. The alphabetized menus are

- The first section of View > Navigational Tabs.
- The first section of View > Toolbars.
- The first section of the Advanced submenu.

**Note:** When nPos is a number, nPos is not obeyed in the Tools menu. A menu item introduced into the Tools menu comes in at the top of the menu. nPos will be obeyed when nPos is a string referencing another user defined menu item.

Returns

Nothing

Example

See newDoc.

addToolButton

6.0

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

See also removeToolButton.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>A unique language independent identifier for the tool button. The language independent name is used to access the tool button for other methods (for example, removeToolButton). <strong>Note:</strong> The value of cName must be unique. To avoid a name conflict, check listToolbarButtons, which lists all toolbar button names currently installed.</td>
</tr>
<tr>
<td>oIcon</td>
<td>A Icon Stream Generic Object.</td>
</tr>
<tr>
<td>cExec</td>
<td>The expression string to evaluate when the tool button is selected.</td>
</tr>
<tr>
<td>cEnable</td>
<td>(optional) An expression string that determines whether or not to enable the tool button. The default is that the tool button is always enabled. This expression should set event.rc to false to disable the tool button.</td>
</tr>
<tr>
<td>cMarked</td>
<td>(optional) An expression string that determines whether or not the tool button is marked. The default is that the tool button is not marked. This expression should set event.rc to true to mark the tool button.</td>
</tr>
<tr>
<td>cTooltext</td>
<td>(optional) The text to display in the tool button help text when the user mouses over the tool button. The default is to not have a tool tip. <strong>Note:</strong> Avoid the use of extended characters in the cTooltext string as the string may be truncated.</td>
</tr>
<tr>
<td>nPos</td>
<td>(optional) The Toolbutton number to place the added Toolbutton before in the Toolbar. If nPos is -1 (the default) then the Toolbutton is appended to the Toolbar.</td>
</tr>
</tbody>
</table>

Returns

An integer.

Exceptions

None

Example

In this example, a series of three tool buttons are created using the icon of a "Text" annotation.

```javascript
// Create a document
var myDoc = app.newDoc();
// Create an annot
var myAnnot = myDoc.addAnnot{}
   {page: 0,
```
create a toolbutton using the icon of the annot.
app.addToolButton
cName: "myButton",
oIcon: myAnnot.uiIcon,
cExec: "console.println('My Button!');",
cTooltext: "This is my button",
nPos: 0
});
// Remove it
app.removeToolButton("myButton");

**alert**

Displays an alert dialog on the screen.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cMsg</td>
<td>A string containing the message to be displayed.</td>
</tr>
</tbody>
</table>
| nIcon     | (optional) An icon type. Values are associated with icons as follows:
0: Error (default)
1: Warning
2: Question
3: Status

**NOTE:** The Macintosh OS does not distinguish between warnings and questions, so it only has three different types of icons.

| nType     | (optional) A button group type. Values are associated with button groups as follows:
0: OK (default)
1: OK, Cancel
2: Yes, No
3: Yes, No, Cancel |

| cTitle    | (optional, version 6.0) A title of the dialog. If not specified the title “Adobe Acrobat” is used. |
| oDoc      | (optional, version 6.0) The Doc Object that the alert should be associated with. |
returns

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

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

Example 1
A simple alert box notifying the user.

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

Example 2
Close the document with the user’s permission

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

Example 3 (Version 6.0)
One doc creates an alert box in another doc. Suppose there are two documents, DocA and DocB. One document is open in a browser and other in the viewer.

```javascript
// The following is a declaration at the document level in DocA
var myAlertBoxes = new Object;
myAlertBoxes.oMyCheckbox = {
    cMsg: "Care to see this message again?",
    bAfterValue: false
```
The following is a MouseUp action in DocA. The variable theOtherDoc is the Doc object of DocB. The alert box ask the user if the user wants to see this alert box again. If the user clicks on the check box provided, the alert does not appear again.

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

**beep**

Causes the system to play a sound.

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

**Parameters**

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

**Returns**

None

**clearInterval**

Cancels a previously registered interval, `oInterval`, initially set by `setInterval`. See also `setTimeout` and `clearTimeout`.

**Parameters**

| oInterval | The registered interval to cancel. |

**Returns**

Nothing
Example
See `setTimeout`.

**clearTimeOut**

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

See also `setInterval` and `clearInterval`.

**Parameters**

<table>
<thead>
<tr>
<th>oTime</th>
<th>The previously registered time-out interval to cancel.</th>
</tr>
</thead>
</table>

**Returns**
Nothing

**Example**
See `setTimeout`.

**execMenuItem**

Executes the specified menu item.

See also `addMenuItem`, `addSubMenu`, `hideMenuItem`. Use `listMenuItems` to list the names of all menu items to the console.

Beginning with version 5.0, `app.execMenuItem("SaveAs")` can be called, subject to the restrictions described below. This saves the current file to the user’s hard drive; a “SaveAs” dialog opens to ask the user to select a folder and file name. Executing the “SaveAs” menu item saves the current file as a linearized file, provided “Save As creates Fast View Adobe PDF files” is checked in the **Edit > Preferences > General** dialog.

**NOTE:** (Security): `app.execMenuItem("SaveAs")` can only be executed during batch, console or menu events. See the Event Object for a discussion of Acrobat JavaScript events.

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

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

Obtains the name of the \( n \)th plug-in that has been loaded by the viewer. See also `numPlugIns`.

See `plugIns` which supersedes this property in later versions.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nIndex</td>
<td>The ( n )th plug-in loaded by the viewer.</td>
</tr>
</tbody>
</table>

#### Returns

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

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

**Parameters**

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

**Returns**

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

**Example 1**

Find the path to the user's Sequences folder

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

**Example 2**

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

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

**goBack**

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

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

**Parameters**

None

**Returns**

Nothing

**hideMenuItem**

Removes a specified menu item.

See also `addMenuItem`, `addSubMenu`, `execMenuItem`, and `listMenuItems`.

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

**Parameters**

<table>
<thead>
<tr>
<th>cName</th>
<th>The menu item name to remove.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Menu item names can be discovered with <code>listMenuItems</code>.</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**hideToolBarButton**

Removes a specified toolbar button.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of the toolbar button to remove. Toolbar item names can be discovered with <code>listToolbarButtons</code>.</td>
</tr>
</tbody>
</table>

### Returns

Nothing

### Example

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

```javascript
app.hideToolbarButton("Hand");
```

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

### listMenuItems

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

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

See also `addMenuItem`, `addSubMenu`, `execMenuItem`, and `hideMenuItem`.

#### Parameters

None

#### Returns

Array of `TreeItem Generic Objects`.

#### TreeItem Generic Object

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of a menu item or toolbar button.</td>
</tr>
<tr>
<td>oChildren</td>
<td>(optional) An array of <code>treeItem</code> objects containing the submenus or flyout buttons.</td>
</tr>
</tbody>
</table>

#### Example 1

List all menu item names to the console.

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

**Example 2**

List all menu items to console, fancy format.

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

**listToolbarButtons**

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

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

### Parameters

None

### Returns

Array of `TreeItem` Generic Objects.

### Example

List all toolbar names to the console.

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

See also the `hideToolbarButton`.

**mailGetAddrs**

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

See also mailMsg, mailDoc, mailForm and Report.mail.

Parameters

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

Returns

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

Example

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

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

See also `mailGetAddrs`, `mailDoc`, `mailForm` and `Report.mail`.

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

**Parameters**

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

**Returns**

Nothing

**Example**

```javascript
/* 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."});
```

newDoc

Creates a new document in the Acrobat Viewer and returns the `doc` object. The optional parameters specify the media box dimensions of the document in points.
**NOTE:** (Security(S)): This method can only be executed during batch, console or menu events. See the Event Object for a discussion of Acrobat JavaScript events.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nWidth</td>
<td>(optional) The width (in points) for the new document. The default value is 612.</td>
</tr>
<tr>
<td>nHeight</td>
<td>(optional) The height (in points) for the new document. The default value is 792.</td>
</tr>
</tbody>
</table>

### Returns

Returns the Doc Object of the newly created document.

### Example

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

```javascript
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 })" });
```

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

### newFDF

Create a new FDF Object that contains no data.

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

### Parameters

- None
Returns

A new FDF Object.

Example

Create a FDF with an embedded PDF file.

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

This example continues following the description of app.openPDF.

openDoc

5.0

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

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

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

Parameters

<table>
<thead>
<tr>
<th>cPath</th>
<th>A device-independent path to the document to be opened. The path can relative to oDoc, if passed. The target document must be accessible in the default file system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>oDoc (optional)</td>
<td>A Doc Object to use as a base to resolve a relative cPath. Must be accessible in the default file system.</td>
</tr>
</tbody>
</table>

Returns

The Doc Object, or null

**NOTE:** For version 5.0, this method returns a Doc Object. In version 5.0.5, the method returns the Doc Object, or null if the target document does not have the doc.disclosed property set to true. The “Acrobat 5.0.5 Accessibility and Forms Patch” changed this behavior—this is the behavior of openDoc in Acrobat 6.0 or later—as follows: During a batch, console or menu event, openDoc ignores the disclosed property and returns the Doc Object of the file specified by cPath; during any other event, openDoc returns the Doc Object, if disclosed is true, and null, otherwise.
Example 1
This example opens another document, inserts a prompting message into a text field, sets the focus in the field, then closes the current document.

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

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

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

Example 2
Open an html document on hard drive and convert to PDF.

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

openFDF

| 6.0 | ☑  | ☑ |

Creates a new FDF Object by opening the specified file. The FDF object has methods and properties that can be used on the data that this file contains.

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

**Parameters**

| cDIPath | The device-independent path to the file to be opened. |

**Returns**

The FDF Object for the FDF file that is opened.

**Example**
Create a FDF with an embedded PDF file.

```javascript
var fdf = app.newFDF();
fdf.addEmbeddedFile("/c/myPDFs/myFile.pdf", 1);
fdf.save("/c/myFDFs/myFile.fdf"); // save and close this FDF

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

See fdf.signatureSign for another example of usage.

**popUpMenu**

| 5.0 |

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

See also popUpMenuEx (preferred).

**Parameters**

<table>
<thead>
<tr>
<th>cItem  (optional) If the argument is a string, then it is listed in the menu as a menu item. The menu item name &quot;.&quot; is reserved to draw a separator line in the menu.</th>
</tr>
</thead>
</table>

| Array (optional) If the argument is an array then it appears as a submenu where the first element in the array is the parent menu item. This array can contain further submenus if desired. |

**Returns**

The name of the menu item that was selected.

**Example**

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

**popUpMenuEx**

| 6.0 |

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

Each of the one or more parameters, denoted as oMenuItem is an object literal that describes a menu item to be included in the pop up menu. The parameters are passed in as an array of objects specifying the properties for each menu item.

The use of popUpMenuEx is preferred over the use of popUpMenu.
Parameters

**oMenuItem**  
A MenuItem Generic Object.

Returns

The **cReturn** value of the menu item that was selected, or its **cName**, if **cReturn** was not specified for that item.

MenuItem Generic Object

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

- **cName**  
The name of the menu item. This is the string to appear on the menu item to be created. The value of ",-" is reserved to draw a separator line in the menu.

- **bMarked**  
(optional) Whether the item is to be marked with a check. The default is **false** (not marked).

- **bEnabled**  
(optional) Whether the item is to appear enabled or grayed out. The default is **true** (enabled).

- **cReturn**  
(optional) A string to be returned when the menu item is selected. The default is the value of **cName**.

- **oSubMenu**  
(optional) A MenuItem Generic Object representing a submenu item, or an array of submenu items, each represented by a MenuItem Generic Object.

Example

The following example illustrates all the features of the `popUpMenuEx` method.

```javascript
var cChoice = app.popUpMenuEx(
    
    {cName: "Item1", bMarked:true, bEnabled:false},

    {cName: "-",}

    {cName: "Item2", oSubMenu:
        [ {cName: "Item 2 Sub2 Subsub1", cReturn: "0"} ]

    },{cName: "Item3"},

    {cName: "Item4", bMarked:true, cReturn: "1"}

    )
```


removeToolButton

6.0

Removes a previously added button from the toolbar.

**Parameters**

- **cName**
  
  The language independent identifier provided when `addToolButton` was called.

**Returns**

Nothing

**Exceptions**

None

**Example**

See the example following `addToolButton`.

response

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

**Parameters**

- **cQuestion**
  
  The question to be posed to the user.

- **cTitle**
  

- **cDefault**
  
  (optional) A default value for the answer to the question. If not specified, no default value is presented.

- **bPassword**
  
  (optional) If `true`, indicates that the user's response should show as asterisks (*) or bullets (•) to mask the response, which might be sensitive information. The default is `false`.

- **cLabel**
  
  (optional, version 6.0) A short string to appear in front of and on the same line as the edit text field.

**Returns**

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

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

**setInterval**

5.0

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

See also `clearInterval`, `setTimeOut` and `clearTimeOut`.

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

**Parameters**

<table>
<thead>
<tr>
<th><strong>cExpr</strong></th>
<th>The JavaScript expression to evaluate.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nMilliseconds</strong></td>
<td>The evaluation time period in milliseconds.</td>
</tr>
</tbody>
</table>

**Returns**

An `interval` object

**Example**

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

```javascript
function DoIt() {
     var f = this.getField("Color");
     var nColor = (timeout.count++ % 10 / 10); // Various shades of red.
     var aColor = new Array("RGB", nColor, 0, 0);
```
f.fillColor = aColor;
}
// save return value as a variable
timeout = app.setInterval("DoIt()", 1000);
// Add a property to our timeout object so that DoIt() can keep
// a count going.
timeout.count = 0;

See `setTimeout` for an additional example.

### `setTimeout`  

| 5.0 | 5.0 |

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

See also `clearTimeout`, `setInterval` and `clearInterval`.

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

**Parameters**

<table>
<thead>
<tr>
<th>cExpr</th>
<th>The JavaScript expression to evaluate.</th>
</tr>
</thead>
<tbody>
<tr>
<td>nMilliseconds</td>
<td>The evaluation time period in milleseconds.</td>
</tr>
</tbody>
</table>

**Returns**

A `timeout` object

**Example**

This example creates a simple running marquee. Assume there is a text field named "marquee". The default value of this field is "Adobe Acrobat version 5.0 will soon be here!".

```javascript
// Document level JavaScript function
function runMarquee() {
    var f = this.getField("marquee");
    var cStr = f.value;
    // get field value
    var aStr = cStr.split(" "); // convert to an array
    aStr.push(aStr.shift()); // move first char to last
```
cStr = aStr.join("");          // back to string again
f.value = cStr;                // put new value in field

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

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

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

---

**Bookmark Object**

A bookmark object represents a node in the bookmark tree that appears in the bookmarks navigational panel. Bookmarks are typically used as a “table of contents” allowing the user to navigate quickly to topics of interest.

---

**Bookmark Properties**

**children**

Returns an array of bookmark objects that are the children of this bookmark in the bookmark tree. See also parent and bookmarkRoot.

*Type: Array  Access: R.*

**Example**

Dump all bookmarks in the document.

```javascript
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**

Specifies the color for a bookmark. Values are defined by using gray, RGB or CMYK color. See Color Arrays for information on defining color arrays and how values are used with this property. See also **style**.

**NOTE:** This property is read-only in Adobe Reader.

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

**Example**

The following fun script will color the top level bookmark red, green and blue.

```javascript
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);
```

**doc**

The **Doc Object** that the bookmark resides in.

**Type:** object **Access:** R.

**name**

The text string for the bookmark that the user sees in the navigational panel.

**NOTE:** This property is read-only in Adobe Reader.

**Type:** String **Access:** R/W.
**open**

Determines whether the bookmark shows its children in the navigation panel (open) or whether the children sub-tree is collapsed (closed).

**NOTE:** This property is read-only in Adobe Reader.

_Type: Boolean  
Access: R/W._

**parent**

Returns the parent bookmark of the bookmark or null if the bookmark is the root bookmark. See also `children` and `bookmarkRoot`.

_Type: object | null  
Access: R._

**style**

Specifies the style for the bookmark's font: 0 indicates normal, 1 is italic, 2 is bold, and 3 is bold-italic. See also `color`.

**NOTE:** This property is read-only in Adobe Reader.

_Type: Integer  
Access: R/W._

---

**Bookmark Methods**

**createChild**

Creates a new child bookmark at the specified location. See also `children`, `insertChild` and `remove`. 
Bookmark Methods

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of the bookmark that the user will see in the navigation panel.</td>
</tr>
<tr>
<td>cExpr</td>
<td>(optional) An expression to be evaluated whenever the user clicks on the bookmark. Default is no expression. This is equivalent to creating a bookmark with a JavaScript action; see the PDF Reference, “JavaScript Action” for details.</td>
</tr>
<tr>
<td>nIndex</td>
<td>(optional) The 0-based index into the children array of the bookmark at which to create the new child. Default is 0.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

Create a bookmark at the top of the bookmark panel that takes you to the next page in the document.

```javascript
bookmarkRoot.createChild("Next Page", "this.pageNum++");
```

**execute**

Executes the action associated with this bookmark. This can have a variety of behaviors. See the PDF Reference, Section 7.5.3, “Actions Types” for a list of common action types. See also `createChild`.

Parameters

None

Returns

Nothing

**insertChild**

Inserts the specified bookmark as a child of this bookmark. If the bookmark already exists in the bookmark tree it is unlinked before inserting it back into the tree. In addition, the insertion is checked for circularities and disallowed if one exists. This prevents users from inserting a bookmark as a child or grandchild of itself. See also `children`, `createChild`, and `remove`. 
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oBookmark</td>
<td>A bookmark object to add as the child of this bookmark.</td>
</tr>
<tr>
<td>nIndex</td>
<td>(optional) The 0-based index into the children array of the bookmark at which to insert the new child. The default is 0.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

Take the first child bookmark and move it to the end of the bookmarks.

```javascript
var bm = bookmarkRoot.children[0];
bookmarkRoot.insertChild(bm, bookmarkRoot.children.length);
```

remove

Removes the bookmark and all its children from the bookmark tree. See also `children`, `createChild`, and `insertChild`.

Parameters

None

Returns

Nothing

Example

Remove all bookmarks from the document.

```javascript
bookmarkRoot.remove();
```

setAction

Sets a JavaScript action for a bookmark. See also `addScript`, `setPageAction`, and `setAction`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cScript</td>
<td>Defines the JavaScript expression that is to be executed whenever the user clicks on the bookmark.</td>
</tr>
</tbody>
</table>
Returns
Nothing

Example
Attach an action to the topmost bookmark.

```javascript
var bm = bookmarkRoot.children[0]
bm.setAction("app.beep(0);")
```

---

**Catalog Object**

A static object that accesses the functionality provided by the Acrobat Catalog plug-in. This plug-in must be installed in order to interface with the catalog object.

**NOTE:** Catalog plug-in (and the catalog object) is available only in the Acrobat Pro.

See also the Index Object, used to invoke various indexing operations provided by Catalog plug-in, and the CatalogJob Generic Object.

---

**Catalog Properties**

**isIdle**

| 6.0 | ⚪ | ☒ |

Returns true when Catalog is idle and not busy with an indexing job.

*Type: Boolean    Access: R.*

**jobs**

| 6.0 | ⚪ | ☒ |

Gets information about the Catalog jobs. Catalog maintains a list of its pending, in progress and completed jobs for each Acrobat session. Returns an array of CatalogJob Generic Objects.

*Type: Array    Access: R.*
Catalog Methods

**getIndex**

6.0 X P

Uses a specified path of a Catalog index to get an `index` object. The returned `index` object can be used to perform various indexing operations such as building or deleting an index.

**Parameters**

- `cDIPath` The device-independent path of a Catalog index.

**Returns**

The `Index Object`.

**remove**

6.0 X P

Removes the specified `CatalogJob` object from Catalog's job list. Catalog maintains a list of pending, in progress and completed jobs for each Acrobat session.

**Parameters**

- `oJob` The `CatalogJob Generic Object` to remove, as returned by the `jobs` property and various methods of the `Index Object`.

**Returns**

Nothing

**Example**

Delete all jobs that are pending and need complete rebuild.

```javascript
if (typeof catalog != undefined) {
    for (var i=0; i<catalog.jobs.length; i++){
        var job = catalog.jobs[i];
        console.println("Index: ", job.path);

        if (job.status == "Pending" && job.type == "Rebuild")
            catalog.remove(job);
    }
}
```
CatalogJob Generic Object

This generic JS object provides information about a job submitted to Catalog. It is returned by `index.build`, and the `catalog.jobs` property, and passed to `catalog.remove`.

It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>String</td>
<td>R</td>
<td>Device independent path of the index associated with the job</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>R</td>
<td>Type of indexing operation associated with the job. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Build</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Rebuild</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Delete</td>
</tr>
<tr>
<td>status</td>
<td>String</td>
<td>R</td>
<td>The status of the indexing operation. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pending</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Processing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CompletedWithErrors</td>
</tr>
</tbody>
</table>

Certificate Object

The Certificate Object provides read-only access to the properties of an X.509 public key certificate.

Related objects and methods are:

- Security Object: `importFromFile` and `exportToFile`
- DirConnection Object: `search`
- Field Object: `signatureInfo`
- FDF Object: `signatureValidate`
- RDN Generic Object
- Usage Generic Object

**Note:** There are no security restrictions on this object.
Certificate Properties

**binary**

The raw bytes of the certificate, as a hex encoded string.

*Type: String  Access: R.*

**issuerDN**

The distinguished name of the issuer of the certificate, returned as an RDN Generic Object.

*Type: RDN object  Access: R.*

**keyUsage**

An array of strings indicating the value of the certificate key usage extension. Possible values are:

- kDigitalSignature
- kNonRepudiation
- kKeyEncipherment
- kDataEncipherment
- kKeyAgreement
- kKeyCertSign
- kCRLSign
- kEncipherOnly
- kDecipherOnly

*Type: Array of Strings  Access: R.*

**MD5Hash**

The MD5 digest of the certificate, represented as a hex-encoded string. This provides a unique fingerprint for this certificate.
Certificate Properties

**SHA1Hash**

Type: String  Access: R.

The SHA1 digest of the certificate, represented as a hex-encoded string. This provides a unique fingerprint for this certificate.

Type: String  Access: R.

**serialNumber**

Type: String  Access: R.

A unique identifier for this certificate, used in conjunction with **issuerDN**.

**subjectCN**

Type: String  Access: R.

The common name of the signer.

**subjectDN**

Type: RDN object  Access: R.

The distinguished name of the signer, returned as an **RDN Generic Object**.

**usage**

Type: Usage Generic Object  Access: R.

The purposes for which this certificate may be used within the Acrobat environment returned as a Usage Generic Object.
Type: usage object  
Access: R.

Usage Generic Object

This generic JS object represents a certificate usage value in the `certificate.usage` property. It has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>endUserSigning</td>
<td>Boolean</td>
<td>R</td>
<td>true if the certificate is usable for end-user signing.</td>
</tr>
<tr>
<td>endUserEncryption</td>
<td>Boolean</td>
<td>R</td>
<td>true if the certificate is usable for end-user encryption.</td>
</tr>
</tbody>
</table>

Example

The following example shows how the `usage` property can be used. The result of this script execution will be that the currently open document is encrypted for everyone in the addressbook. Addressbook entries that contain sign-only certificates, CA certificates, no certificates at all, or are otherwise unsuitable for encryption, will not be included in the final recipient list.

```javascript
var eng = security.getHandler( "Adobe.AAB" );
var dc = eng.directories[0].connect();
var recipients = dc.search();

var filteredRecipients = new Array();
for( i = 0; i < recipients.length; ++i ) {
    if( recipients[i].defaultEncryptCert &&
        recipients[i].defaultEncryptCert.usage.endUserEncryption ) {
        filteredRecipients[filteredRecipients.length] = recipients[i];
        continue;
    }
    if(recipients[i].certificates) {
        for( j = 0; j < recipients[i].certificates.length; ++j )
        if( recipients[i].certificates[j].usage.endUserEncryption ) {
            filteredRecipients[filteredRecipients.length] = recipients[i];
            continue;
        } 
    }
}
this.encryptForRecipients({ [userEntities: filteredRecipients] });
```
Collab Object

This object represents the Collaboration functionality.

Collab Methods

addStateModel

Add a new state model to Acrobat. A state model describes the valid states that an annot using the model can have (see the Annot Object for details about getting and setting the state of an annot). State models can be used to describe the workflow that a document review goes through and can be used for review management.

See also removeStateModel, getStateInModel and transitionToState.

Parameters

cName
A unique, language-independent identifier for the State Model.

cUIName
The display name of the state model used in the User Interface and should be localized.

oStates
The states in the state model, described by a States Object Literal.

cDefault
(optional) One of the states in the model to be used as a default state if no other state is set. The default is for there to be no default state.

bHidden
(optional) Whether the state model should be hidden in the state model user interface. The default is false (the State Model is shown).

bHistory
(optional) Whether an audit history is maintained for the state model. Keeping an audit history requires more space in the file. The default is true.

Returns

Nothing
States Object Literal

This object literal represents a set of states in a state model, and is passed as the oStates parameter. The elements in the object literal are the unique state identifiers and the values are objects having the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cUIName</td>
<td>The UI (display name) for the state.</td>
</tr>
<tr>
<td>oIcon</td>
<td>(optional) An Icon Stream Generic Object that will be displayed in the UI for the state.</td>
</tr>
</tbody>
</table>

Example

Add a new state model with a unique name of "ReviewStates":

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

removeStateModel

Removes a state model that was previously added by calling addStateModel. Removing a state model does not remove the state information associated with individual annots—if the model is removed and added again, all of the state information for the annots will still be available.

See also addStateModel, getStateInModel and transitionToState.

Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>A unique, language-independent identifier for the State Model that was used in addStateModel.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

Continuing the example in addStateModel, we remove the state model "ReviewStates":

```javascript
try {
  // Remove the state model
```
Collab.removeStateModel("ReviewStates");
}
catch(e) { console.println(e); }

**Color Object**

The color object is a convenience static object that defines the basic colors. These colors are accessed in JavaScripts via the color object. Use this object whenever you want to set a property or call a method that require a color array. The color object is defined in AForm.js.

**Color Arrays**

A color is represented in JavaScript as an array containing 1, 2, 4, or 5 elements corresponding to a Transparent, Gray, RGB, or CMYK color space, respectively. The first element in the array is a string denoting the color space type. The subsequent elements are numbers that range between zero and one inclusive. For example, the color red can be represented as 

\[
["RGB", 1, 0, 0]
\]

Invalid strings or insufficient elements in a color array cause the color to be interpreted as the color black.

<table>
<thead>
<tr>
<th>Color Space</th>
<th>String</th>
<th>Number of Additional Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent</td>
<td>&quot;T&quot;</td>
<td>0</td>
<td>A transparent color space indicates a complete absence of color and will allow those portions of the document underlying the current field to show through.</td>
</tr>
<tr>
<td>Gray</td>
<td>&quot;G&quot;</td>
<td>1</td>
<td>Colors in the gray color space are represented by a single value—the intensity of achromatic light. In this color space, 0 is black, 1 is white, and intermediate values represent shades of gray. For example, .5 represents medium gray.</td>
</tr>
<tr>
<td>RGB</td>
<td>&quot;RGB&quot;</td>
<td>3</td>
<td>Colors in the RGB color space are represented by three values: the intensity of the red, green, and blue components in the output. RGB is commonly used for video displays because they are generally based on red, green, and blue phosphors.</td>
</tr>
</tbody>
</table>
The color object defines the following colors:

<table>
<thead>
<tr>
<th>Color Object</th>
<th>Keyword</th>
<th>Equivalent JS</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent</td>
<td>color.transparent</td>
<td>[ &quot;T&quot; ]</td>
<td></td>
</tr>
<tr>
<td>Black</td>
<td>color.black</td>
<td>[ &quot;G&quot;, 0 ]</td>
<td></td>
</tr>
<tr>
<td>White</td>
<td>color.white</td>
<td>[ &quot;G&quot;, 1 ]</td>
<td></td>
</tr>
<tr>
<td>Red</td>
<td>color.red</td>
<td>[ &quot;RGB&quot;, 1, 0, 0 ]</td>
<td></td>
</tr>
<tr>
<td>Green</td>
<td>color.green</td>
<td>[ &quot;RGB&quot;, 0, 1, 0 ]</td>
<td></td>
</tr>
<tr>
<td>Blue</td>
<td>color.blue</td>
<td>[ &quot;RGB&quot;, 0, 0, 1 ]</td>
<td></td>
</tr>
<tr>
<td>Cyan</td>
<td>color.cyan</td>
<td>[ &quot;CMYK&quot;, 1, 0, 0, 0 ]</td>
<td></td>
</tr>
<tr>
<td>Magenta</td>
<td>color.magenta</td>
<td>[ &quot;CMYK&quot;, 0, 1, 0, 0 ]</td>
<td></td>
</tr>
<tr>
<td>Yellow</td>
<td>color.yellow</td>
<td>[ &quot;CMYK&quot;, 0, 0, 1, 0 ]</td>
<td></td>
</tr>
<tr>
<td>Dark Gray</td>
<td>color.dkGray</td>
<td>[ &quot;G&quot;, 0.25 ]</td>
<td>4.0</td>
</tr>
<tr>
<td>Gray</td>
<td>color.gray</td>
<td>[ &quot;G&quot;, 0.5 ]</td>
<td>4.0</td>
</tr>
<tr>
<td>Light Gray</td>
<td>color.ltGray</td>
<td>[ &quot;G&quot;, 0.75 ]</td>
<td>4.0</td>
</tr>
</tbody>
</table>
Example

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.

```javascript
var f = event.target; /* field that the event occurs at */
if (event.value < 0) f.target.textColor = color.red; else f.target.textColor = color.black;
```

Color Methods

convert

Converts the colorspace and color values specified by the `color` object to the specified colorspace. Note that conversion to the gray colorspace is lossy in the same fashion that displaying a color TV signal on a black and white TV is lossy. The conversion of RGB to CMYK does not take into account any black generation or under color removal parameters.

Parameters

- **colorArray**: Array of color values. See Color Arrays.
- **cColorspace**: The colorspace to which to convert.

Returns

A color array.

Example

The return value of the code line below is the array `"CMYK", 0, 1, 1, 0]`.

```javascript
color.convert(["RGB", 1,0,0], "CMYK")
```
equal

Compares two Color Arrays to see if they are the same. The routine performs conversions, if necessary, to determine if the two colors are indeed equal (for example, ["RGB" 1 1 0] is equal to ["CMYK" 0 0 1 0]).

Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>colorArray1</td>
<td></td>
<td></td>
<td>The first color array for comparison.</td>
</tr>
<tr>
<td>colorArray2</td>
<td></td>
<td></td>
<td>The second color array for comparison.</td>
</tr>
</tbody>
</table>

Returns

true if the arrays represent the same color, false otherwise.

Example

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

Column Generic Object

This generic JS object contains the data from every row in a column. A column object is returned by `statement.getColumn` and `statement.getColumnArray`. See also the ColumnInfo Generic Object.

It has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnNum</td>
<td>number</td>
<td>R</td>
<td>The number identifying the column.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>R</td>
<td>The name of the column.</td>
</tr>
<tr>
<td>type</td>
<td>number</td>
<td>R</td>
<td>One of the SQL Types for the data in the column.</td>
</tr>
<tr>
<td>typeName</td>
<td>string</td>
<td>R</td>
<td>The name of the type of data the column contains.</td>
</tr>
<tr>
<td>value</td>
<td>various</td>
<td>R/W</td>
<td>The value of the data in the column, in the format in which the data was originally retrieved.</td>
</tr>
</tbody>
</table>
**ColumnInfo Generic Object**

This generic JS object contains basic information about a column of data, and is returned by `connection.getColumnList`. See also Column Generic Object.

It has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>R</td>
<td>A string that represents the identifying name of a column. This string could be used in a <code>statement.getColumn</code> call to identify the associated column.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>R</td>
<td>A string that contains database-dependent information about the column.</td>
</tr>
<tr>
<td>type</td>
<td>number</td>
<td>R</td>
<td>A numeric value identifying one of the ADBC SQL Types that applies to the data contained in the column associated with the ColumnInfo object.</td>
</tr>
<tr>
<td>typeName</td>
<td>string</td>
<td>R</td>
<td>A string identifying the type of the data contained in the associated column. This is not the SQL Types (see type above), but a database-dependent string representing the data type. This property may give useful information about user-defined data types.</td>
</tr>
</tbody>
</table>

**Connection Object**

The `Connection` object encapsulates a session with a database. `Connection` objects are returned by `ADBC.newConnection`. See also the ADBC Object, Statement Object, Column Generic Object, ColumnInfo Generic Object, Row Generic Object, and TableInfo Generic Object.
Connection Methods

**close**

| 6.0 |  |

Closes an active connection and invalidates all the objects created from the connection.

**Parameters**

None

**Returns**

Nothing

**newStatement**

| 5.0 |  |

Creates a [Statement Object](#) through which database operations may be performed.

**Parameters**

None

**Returns**

A [Statement](#) object on success or null on failure.

**Example**

```javascript
// 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 |  |

Gets information about the various tables in a database.

**Parameters**

None
Returns

It returns an array of TableInfo Generic Objects. This method never fails but may return a zero-length array.

Example

Assuming we have a Connection object (con) already in hand (see newStatement and newConnection), get the list of tables

```javascript
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

Gets information about the various columns in the table

Parameters

<table>
<thead>
<tr>
<th>cName</th>
<th>The name of the table to get column information about.</th>
</tr>
</thead>
</table>

Returns

Returns an array of ColumnInfo Generic Objects. This method never fails but may return a zero-length array.

Example

Assuming we have a Connection object (con) already in hand (see newStatement and newConnection), get list of all column names.

```javascript
var con = ADBC.newConnection("q32000data");
var columnInfo = con.getColumnList("sales");
console.println("Column Information");
for (var i = 0; i < columnInfo.length; i++) {
    console.println(columnInfo[i].name);
    console.println("Description: "+ columnInfo[i].description);
}
```
Console Object

The **Console** object is a static object to access the JavaScript console for displaying debug messages and executing JavaScript. It does not function in the Adobe Reader or Acrobat Approval.

See also the **Dbg Object**.

Console Methods

**show**

Shows the console window.

**Parameters**

None

**Returns**

Nothing

**hide**

Closes the console window.

**Parameters**

None

**Returns**

Nothing

**printIn**

Prints a string value to the console window with an accompanying carriage return.

**Parameters**

<table>
<thead>
<tr>
<th>cMessage</th>
<th>A string message to print.</th>
</tr>
</thead>
</table>

**Returns**

Nothing
Example 1

This example prints the value of a field to the console window. The script could executed during a mouse up event.

```javascript
var f = this.getField("myText");
console.clear();
console.show();
console.println("Field value = " + f.value);
```

Example 2

The console can be used as a debugging tool; you can write values of variables to the console, for example. The script below is taken from the document level.

```javascript
var debugIsOn = true;
function myFunction ( n, m )
{
    if (debugIsOn)
    {
        console.println("Entering function: myFunction");
        console.println(" Parameter 1: n = " + n);
        console.println(" Parameter 2: m = " + m);
    }
    ....
    ....
    if (debugIsOn) console.println(" Return value: rtn = " + rtn);
    return rtn;
}
```

**clear**

Clears the console windows buffer of any output.

**Parameters**

None

**Returns**

Nothing

---

**Data Object**

5.0

The *data* object is the representation of an embedded file or data stream that is stored in the document. *Data* objects are stored in the name tree in the document. See the section on the Names Tree and Embedded File Streams in the *PDF Reference* for details.
Data objects can be inserted from the external file system, queried, and extracted. This is a good way to associate and embed source files, metadata, and other associated data with a document.

See the following Doc Object properties and methods:
- `createDataObject`, `dataObjects`, `exportDataObject`, `getDataObject`, `importDataObject`, `removeDataObject`.

**Note:** While the methods for data objects were implemented in Acrobat 5.0, the ability to use these in an Adobe Reader-extended context only became available in Acrobat 6.0.

---

### Data Properties

**creationDate**

The creation date of the file that was embedded.

*Type: Date  Access:* R.

**modDate**

The modification date of the file that was embedded.

*Type: Date  Access:* R.

**MIMEType**

The MIME type associated with this data object.

*Type: String  Access:* R.

**name**

The name associated with this data object.

*Type: String  Access:* R.

**Example**

```javascript
console.println("Dumping all data objects in the document.");
var d = this.dataObjects;
for (var i = 0; i < d.length; i++)
    console.println("DataObject[" + i + "]=" + d[i].name);
```
path

The device-independent path to the file that was embedded.

_Type: String  Access: R._

size

The size, in bytes, of the uncompressed data object.

_Type: Number  Access: R._

---

**DataSourceInfo Generic Object**

This generic JS object contains basic information about a particular database. The `ADBC.getDataSourceList` method returns an array of these objects. The object has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>R</td>
<td>A string that represents the identifying name of a database. This string could be passed to <code>newConnection</code> to establish a connection to the database that the DataSourceInfo object is associated with.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>R</td>
<td>A string that contains database dependent information about the database.</td>
</tr>
</tbody>
</table>

---

**Dbg Object**

The `dbg` object is used to optionally control the JavaScript Debugger from a command-line console standpoint. The same functionality provided by the buttons in the JavaScript Debugger dialog toolbar available from the `dbg` methods. In addition, breakpoints can be created, deleted and inspected using the `dbg` object.

The `dgb` object and the JavaScript Debugger are only available in Acrobat Pro.
**NOTES:** Should the viewer lock up during a debugging session, pressing the Esc-key may resolve the problem.

Debugging is not possible with a model dialog open, this occurs, for example, when debugging a batch sequence.

Debugging script with an running event initiated by either `app.setInterval` or `app.setTimeOut` may cause a recurring alert boxes to appear. Use the Esc-key after the model dialog is dismissed to resolve the problem.

---

### Dbg Properties

**bps**

Returns an array of Breakpoint Generic Objects, each element corresponding to a breakpoint set in the debugger.

*Type: Array*  
*Access: R.*

#### Breakpoint Generic Object

This generic JS object contains basic information about a breakpoint, and is returned by the `Dbg.bps` property. It contains the following properties and methods:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fileName</code></td>
<td>string</td>
<td>R</td>
<td>A string that identifies the script in the debugger.</td>
</tr>
<tr>
<td><code>condition</code></td>
<td>string</td>
<td>R</td>
<td>A JavaScript expression evaluated whenever the debugger has to decide to stop or not at a breakpoint. Used to create conditional breakpoints. The default value for this property is the string &quot;true&quot;.</td>
</tr>
<tr>
<td><code>lineNum</code></td>
<td>number</td>
<td>R</td>
<td>The line number in the script for which the breakpoint is set.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toString</code></td>
<td>none</td>
<td>String</td>
<td>A string describing the breakpoint.</td>
</tr>
</tbody>
</table>

**Example**

List all currently active breakpoints.

```javascript
var db = dbg.bps
```
```javascript
for ( var i = 0; i < db.length; i++ )
{
    for ( var o in db[i] )
        console.println(o + " : " + db[i][o]);
    console.println("------------------------------------");
}

See sb for another example of usage.
```

### Dbg Methods

#### c

<table>
<thead>
<tr>
<th>6.0</th>
<th>D</th>
<th>P</th>
</tr>
</thead>
</table>

The `c` (continue) method resumes execution of a program stopped in the debugger. The JavaScript program may either stop again, depending on where the breakpoints are set, or reach execution end.

**Parameters**

None

**Returns**

Nothing

#### cb

<table>
<thead>
<tr>
<th>6.0</th>
<th>D</th>
<th>P</th>
</tr>
</thead>
</table>

The `cb` (clear breakpoint) method clears a breakpoint in the debugger.

**Parameters**

- **fileName**
  
  The name of the script from where the breakpoint is going to be deleted.

- **lineNum**
  
  The line number for the breakpoint that is going to be cleared in the script.

**Returns**

Nothing
The `q` (quit) method quits debugging and executing the current JavaScript. It additionally dismisses the debugger dialog.

**Parameters**
None

**Returns**
Nothing

The `sb` (set breakpoint) method sets a new breakpoint in the debugger.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fileName</code></td>
<td>The name of the script where the breakpoint is to be set.</td>
</tr>
<tr>
<td><code>lineNum</code></td>
<td>The line number where the breakpoint is going to be created in the script.</td>
</tr>
<tr>
<td><code>condition</code></td>
<td>(optional) a JavaScript expression evaluated every time the debugger reaches a breakpoint. The decision to stop or not at a breakpoint is based on the result of evaluating such expression. If the expression evaluates to <code>true</code>, the debugger will stop at the breakpoint. If the expression evaluates to <code>false</code>, the debugger continues executing the script and will not stop at the breakpoint. The default value for this parameter is the string &quot;true&quot;.</td>
</tr>
</tbody>
</table>

**Returns**
Nothing

**Example 1**

Some script is run and an exception is thrown due to some error. A breakpoint is programmatically set using the information given in the error message.

```
SyntaxError: missing ; before statement 213:Document-Level: myDLJS
// now set a breakpoint using the console
dbg.sb({
  fileName: "Document-Level: myDLJS",
  lineNum: 213,
  condition: "true"
})
```
Example 2

This example simulates the functionality of the “Store breakpoints in PDF file’ checkbox in the Preferences > JavaScript dialog.

```javascript
// save breakpoints in PDF file
this.addScript("myBreakpoints", "var myBPS = " + dbg.bps.toSource());

// now reset the breakpoints
for ( var i = 0; i < myBPS.length; i++ ) dbg.sb( myBPS[i] );
```

Example 3

Set a conditional break. Consider the following code, which is a mouse up action.

```javascript
for (var i=0; i<100; i++)
    myFunction(i);  // defined at document level

// In the console, set a conditional break. Here, we break when the
// index of the loop is greater than 30.
dbg.sb({
    fileName:"AcroForm:Button1:Annot1:MouseUp:Action1",
    lineNumber:2,
    condition:"i > 30"
})
```

**si**

The **si** (step in) method advances the program pointer to the next instruction in the JavaScript program, entering each function call that is encountered, and for which there is a script defined. Native JavaScript calls cannot be stepped into.

**Parameters**

None

**Returns**

Nothing

**sn**

The **sn** (step instruction) method advances the program pointer to the next byte-code in the JavaScript program. Each JavaScript instruction is made up of several byte-codes as defined by the JavaScript interpreter.
**Parameters**

None

**Returns**

Nothing

**SO**

The **SO** (step out) method executes the program until it comes out of the current function. It stops executing in the instruction immediately following the call to the function. If the scope currently under debug is the top level scope, the program may continue executing until it ends, or stop again when it reaches a breakpoint.

**Parameters**

None

**Returns**

Nothing

**SV**

The **SV** (step over) method advances the program pointer to the next instruction in the JavaScript program. If a function call is encountered, the debugger will not step into the instructions defined inside that function.

**Parameters**

None

**Returns**

Nothing

### Directory Object

Directories are a repository of user information, including public-key certificates. Directory Objects provide directory access and are obtained using the `directories` property or the `newDirectory` method of the `SecurityHandler` Object.
Acrobat 6.0 provides several directories. The Adobe.AAB Security Handler has a single directory named Adobe.AAB.AAB. This directory provides access to the local Acrobat Address Book, also called the Trusted Identity Store. On Windows, the Adobe.PPKMS Security Handler provides access, via Microsoft Active Directory Script Interface (ADSI) to as many directories as have been created by the user. These directories are created sequentially with names Adobe.PPKMS.ADSI.dir0, Adobe.PPKMS.ADSI.dir1, and so on.

**NOTE:** Security This object can only be obtained from a SecurityHandler Object and is thus governed by the security restrictions of the SecurityHandler Object. The Directory Object is therefore available only for batch, console, application initialization and menu execution, including in Acrobat Reader.

**Directory Properties**

**info**

The value of this property is a DirectoryInformation Generic Object, a generic object used to set and get the properties for this Directory Object.

*Type: Object Access: R/W.*

**Example**

```javascript
// Create and activate a new directory
var oDirInfo = {
  dirStdEntryID: "dir0",
  dirStdEntryName: "Employee LDAP Directory",
  dirStdEntryPrefDirHandlerID: "Adobe.PPKMS.ADSI",
  dirStdEntryDirType: "LDAP",
  server: "ldap0.acme.com",
  port: 389
};
var sh = security.getHandler( "Adobe.PPKMS" );
var newDir = sh.newDirectory();
newDir.info = oDirInfo;
```
**DirectoryInformation Generic Object**

A directory information object is a generic object representing the properties for a directory and has the following standard properties:

**Standard Directory Information Object properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dirStdEntryID</td>
<td>String</td>
<td>R/W</td>
<td>Yes</td>
<td>A unique, language independent name for the directory. Must be alphanumeric and can include underscores, periods and hyphens. For new directory objects it is suggested that the ID not be provided, in which case a new unique name will be automatically generated.</td>
</tr>
<tr>
<td>dirStdEntryName</td>
<td>String</td>
<td>R/W</td>
<td>Yes</td>
<td>A user friendly name for the directory.</td>
</tr>
<tr>
<td>dirStdEntryPrefDirHandlerID</td>
<td>String</td>
<td>R/W</td>
<td>No</td>
<td>The name of the directory handler that is to be used by this directory. Security handlers can support multiple directory handlers for multiple directory types (eg. local directories, LDAP directories).</td>
</tr>
<tr>
<td>dirStdEntryDirType</td>
<td>String</td>
<td>R/W</td>
<td>No</td>
<td>The type of directory. An example of this would be LDAP, ADSI, WINNT.</td>
</tr>
<tr>
<td>dirStdEntryVersion</td>
<td>String</td>
<td>R</td>
<td>No</td>
<td>The version of the data. The default value is 0 if this is not set by the directory. The value for Acrobat 6.0 directories for the Adobe.AAB and Adobe.PPKMS.ADSI directory handlers is 0x00010000.</td>
</tr>
</tbody>
</table>
Directory information objects can include additional properties that are specific to a particular directory handler. The Adobe.PPKMS.ADSI directory handler includes the following additional properties:

**Example 1**

Create and activate a new directory.

```javascript
var oDirInfo = { dirStdEntryID: "dir0",
    dirStdEntryName: "Employee LDAP Directory",
    dirStdEntryPrefDirHandlerID: "Adobe.PPKMS.ADSI",
    dirStdEntryDirType: "LDAP",
    server: "ldap0.acme.com",
    port: 389
};
var sh = security.getHandler( "Adobe.PPKMS" );
var newDir = sh.newDirectory();
newDir.info = oDirInfo;
```

**Example 2**

Get information for existing directory.

```javascript
var sh = security.getHandler( "Adobe.PPKMS" );
var dir0 = sh.directories[0];
// Get directory info object just once for efficiency
var dir0Info = dir0.info;
console.println( "Directory " + dir0Info.dirStdEntryName );
console.println( "address " + dir0Info.server + ":" + dir0Info.port );
```

---

**Adobe.PPKMS.ADSI additional directory information object properties**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>server</td>
<td>String</td>
<td>R/W</td>
<td>The server that hosts the data. For example, addresses.employees.xyz.com.</td>
</tr>
<tr>
<td>port</td>
<td>Number</td>
<td>R/W</td>
<td>The port number for the server. The standard LDAP port number is 389.</td>
</tr>
<tr>
<td>searchBase</td>
<td>String</td>
<td>R/W</td>
<td>Narrows down the search to a particular section of the directory. An example of this would be o=XYZ Systems,c=US.</td>
</tr>
<tr>
<td>maxNumEntries</td>
<td>Number</td>
<td>R/W</td>
<td>The maximum number of entries that would be retrieved in a single search.</td>
</tr>
<tr>
<td>timeout</td>
<td>Number</td>
<td>R/W</td>
<td>The maximum time allowed for a search.</td>
</tr>
</tbody>
</table>
Directory Methods

connect

6.0 $

Returns a DirConnection Object that is a connection to the directory with the specified name. There can be more than one active connection for a directory.

See also DirConnection Object and the SecurityHandler Object's directories property.

Parameters

- `oParams` (optional) A generic object that can contain parameters that are necessary in order to create the connection. Properties of this object are dependent on the particular directory handler and can include `userid` and `password`.

- `bUI` (optional) A boolean value that defaults to `false`. It conveys to the directory handler if it could bring its UI in case that is required for establishing the connection.

Returns

A DirConnection Object, or null, if there is no directory with the specified name.

Example:

Enumerate available directories and connect.

```javascript
var sh = security.getHandler( "Adobe.PPKMS" );
var dirList = sh.directories;
var dirConnection = sh.dirList[0].connect();
```

DirConnection Object

6.0 $

The DirConnection object represents an open connection to a directory: a repository of user information, including public-key certificates. Directory connections are opened using the Directory Object's connect method. A directory with a particular name can have more than one connection open at a time. All DirConnection objects must support all properties and methods listed here, unless otherwise specified.
**NOTE:** (Security ⛔️): This object can only be obtained from a **Directory Object** and is thus governed by the security restrictions of the **Directory Object**. The **DirConnection Object** is therefore available only for batch, console, application init and menu exec, including in Acrobat Reader.

---

### DirConnection Properties

#### canList

**Type:** Boolean  
**Access:** R.

Indicates whether the directory connection is capable of listing all of its entries. Some directories may contain too many entries for this operation to be practical.

**Example**

The AAB directory allows listing of the local trusted identity list

```javascript
var sh = security.getHandler( "Adobe.AAB" );
var dc = sh.directories[0].connect();
console.println( "CanList = " + dc.canList );
```

#### canDoCustomSearch

**Type:** Boolean  
**Access:** R.

Whether the directory connection supports search using directory-specific search parameter attributes. As an example, directory-specific attributes for an LDAP directory include: o (organization), c (country), cn (common name), givenname, sn (surname), uid, st, postalcode, mail, and telephonenumber.

#### canDoCustomUISearch

**Type:** Boolean  
**Access:** R.

Whether the directory connection supports search using its own custom user interface to collect the search parameters.
**canDoStandardSearch**

Whether the directory connection supports search using standard search parameter attributes. The standard attributes are:

- firstName
- lastName
- fullName
- email
- certificates

Some directory database implementations may not support these attributes, but directory handlers are free to translate these attributes to names understood by the directory.

*Type: Boolean Access: R.*

**groups**

Returns an array of language dependent names for groups that are available through this connection.

*Type: Array Access: R.*

**name**

Returns the language independent name of the directory that this object is connected to. An example of this would be `Adobe.PPKMS.ADSI.dir0`. All DirConnection objects must support this property.

*Type: String Access: R.*

**uiName**

Returns the language dependent string of the directory this object is connected to. This string is suitable for user interfaces. An example of this would be `XYZ's Employees`. All DirConnection objects must support this property.

*Type: String Access: R.*
DirConnection Methods

search

Searches the directory and returns an array of UserEntity Generic Objects that match the search parameters. A UserEntity Generic Object is a generic object that contains properties for all attributes that were requested via the `setOutputFields` method. If the `setOutputFields` method is not called prior to a search it would return a UserEntity Generic Object containing no entries.

Parameters

<table>
<thead>
<tr>
<th>oParams</th>
<th>(optional) A generic object containing an array of key-value pairs consisting of search attribute names and their corresponding strings. If <code>oParams</code> is not provided and <code>canList</code> is true for this directory then all entries in the directory will be returned. If <code>oParams</code> is not provided and <code>canList</code> is false, an exception occurs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cGroupName</td>
<td>(optional) The name of a group (not to be confused with Group Objects). If specified then the search will be restricted to this group.</td>
</tr>
<tr>
<td>bCustom</td>
<td>(optional) If false (the default), <code>oParams</code> contains standard search attributes. The <code>canDoStandardSearch</code> property must be true, or an exception occurs. If true, then <code>oParams</code> contains directory-specific search parameters. The <code>canDoCustomSearch</code> property must be true, or an exception occurs.</td>
</tr>
<tr>
<td>bUI</td>
<td>(optional) If true, the handler shows user interface to allow collection of search parameters. The results of the search are returned by this method. <code>canDoCustomUISearch</code> must also be true if <code>bUI</code> is true, or an exception will occur. If <code>bUI</code> is specified then <code>bCustom</code> must also be specified, though its value is ignored.</td>
</tr>
</tbody>
</table>

Returns

An array of UserEntity Generic Objects.

Example 1

Directory search

```javascript
var sh = security.getHandler( "Adobe.PPKMS" );
var dc= sh.directories[0].connect();
dc.setOutputFields( {oFields: ["certificates","email"]} )
var retVal = dc.search({oParams:{lastName:"Smith"}});
if( retVal.length )
  console.println( retVal[0].email );
```
Example 2

List all entries in local Acrobat Address Book

```javascript
var sh = security.getHandler( "Adobe.AAB" );
var dc = shdirectories[0].connect();
if( dc.canList ) {
    var x = dc.search();
    for( j=0; j<x.length; ++j ) {
        console.println("Entry[" + j + "] = " + x[j].fullName + ":");
        for( i in x[j]) console.println("  " + i + " = " + x[j][i]);
    }
}
```

Searches the directory and returns an array of users, along with their certificate information.

**UserEntity Generic Object**

A generic JS object that describes a user in a directory and the user's associated certificates. It contains standard properties that have a specific meaning for all directory handlers. Directory handlers translate these entries to the ones that are specific to them when required. An array of these objects is returned by `dirConnection.search`.

It has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstName</td>
<td>String</td>
<td>R/W</td>
<td>The first name for the user.</td>
</tr>
<tr>
<td>lastName</td>
<td>String</td>
<td>R/W</td>
<td>The last name of the user.</td>
</tr>
<tr>
<td>fullName</td>
<td>String</td>
<td>R/W</td>
<td>The full name of the user.</td>
</tr>
<tr>
<td>certificates</td>
<td>Array of Certificate Objects</td>
<td>R/W</td>
<td>An array of certificates that belong to this user. To find a certificate that is to be used for a particular use, the caller should inspect the certificate's <code>keyUsage</code> property.</td>
</tr>
<tr>
<td>defaultEncryptCert</td>
<td>Array of Certificate Objects</td>
<td>R/W</td>
<td>The preferred certificate to use when encrypting documents for this user entity. Routines that process <code>User Entity Objects</code> will look first to this property when choosing an encryption certificate: if this property is not set then the first valid match in the certificates property will be used.</td>
</tr>
</tbody>
</table>
setOutputFields

6.0  S  X

Defines the list of attributes that should be returned when executing the search method.

**NOTE:** This method is not supported by the Adobe.AAB directory handler. Custom options are not supported by the Adobe.PPKMS.ADSI directory handler.

**Parameters**

<table>
<thead>
<tr>
<th>oFields</th>
<th>An array of strings containing the names of attributes that should be returned from the directory when calling the search method. The names in this array must either be names of standard attributes that can be used for all directory handlers, or custom attributes that are defined for a particular directory. The standard attributes are the property names defined for the UserEntity Generic Object. Directory handlers can, when desired, translate standard attribute names to names that it understands.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bCustom</td>
<td>(optional) A boolean indicating that the names in oFields are standard output attribute names. If true then the names represent directory-specific attributes that are defined for a particular directory handler. The default is false.</td>
</tr>
</tbody>
</table>

**Returns**

An array of strings, containing the names of attributes from oFields that are not supported by this directory. An empty array is returned if the oFields array is empty.

**Example**

In this example, `dc.setOutputFields()` returns the array of strings `"x", "y"]`.

```javascript
var sh = security.getHandler("Adobe.PPKMS");
var dc = sh.directories[0].connect();
var w = dc.setOutputFields( [ "certificates", "email", "x", "y"] );
console.println( w );
```

See also the examples that follow the `DirConnection.search` method.

**Doc Object**

The JavaScript `doc` object provides the interfaces between a PDF document open in the viewer and the JavaScript interpreter. It provides methods and properties of the PDF document.
Doc Access from JavaScript

You can access the doc object from JavaScript in a variety of ways.

- The most common way is through the this Object, which usually points to the doc object of the underlying document.
- Some properties and methods return doc objects; for example, activeDocs, openDoc, or extractPages all return doc objects.
- JavaScript is executed as a result of some event. For each event, an Event Object is created. A doc object can often be accessed through event.target:
  - For mouse, focus, blur, calculate, validate, and format events, event.target returns the Field Object that initiated the event. You can then access the doc object through field.doc.
  - For all other events, event.target points to the doc object.

Example 1: Access through this object

Use this to get the number of pages in this document:

```javascript
var nPages = this.numPages;
// get the crop box for "this" document:
var aCrop = this.getPageBox();
```

Example 2: Access through return values

Return values from one document to open, modify, save and close another.

```javascript
// path relative to "this" doc:
var myDoc = app.openDoc("myNovel.pdf", this);
myDoc.info.Title = "My Great Novel";
myDoc.saveAs(myDoc.path);
myDoc.closeDoc(true);
```

Example 3: Access through the event object.

For mouse, calculate, validate, format, focus, and blur events:

```javascript
var myDoc = event.target.doc;
```

For all other events (for example, batch or console events):

```javascript
var myDoc = event.target;
```

Doc Properties

alternatePresentations

References the document’s AlternatePresentation Object. If the functionality needed to display alternate presentations is not available, this property is undefined.
The `alternatePresentation` object provides access to the document's alternate presentations. The PDF language extension specifies that each document can potentially have many named alternate presentations. Each alternate presentation with a known `type` will have a corresponding `doc.alternatePresentations` property in the document. This property should have the same name as its alternate presentation and should reference its alternate presentation's `AlternatePresentation Object`. If there are no recognized alternate presentations in the document, this object is empty (does not have any properties).

**NOTE:** For compatibility with current implementation alternate presentation name must be an ASCII string. The only alternate presentation type currently implemented is "SlideShow".

See the `AlternatePresentation Object` for properties and methods that can be used to control an alternate presentation.

*Type: Object | undefined  Access: R.*

**Example 1**

Test whether the `alternatePresentations` object is present:

```javascript
if( typeof this.alternatePresentations != "undefined" )
{
    // assume AlternatePresentations are present
    // list the names of all alternate presentations in the doc
    for ( var ap in this.alternatePresentations ) console.println(ap);
}
```

**Example 2**

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

```javascript
// oMySlideShow is an AlternatePresentation object
oMySlideShow = this.alternatePresentations["MySlideShow"]; oMySlideShow.start();
```

**author**

The author of the document. See `info`, which supersedes this property in later versions.

**NOTE:** This property is read-only in Adobe Reader.

*Type: String  Access: R/W.*
**baseURL**

The base URL for the document, used to resolve relative web links within the document. See also URL.

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

**Example**

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

**bookmarkRoot**

The root bookmark for the bookmark tree. This bookmark is not displayed to the user; it is a programmatic construct used to access the tree and the child bookmarks. See the [Bookmark Object](#) an example of usage.

*Type: object*  
*Access: R.*

**calculate**

If true, allows calculations to be performed for this document. If false, prevents all calculations from happening for this document. Its default value is true. This property supersedes the app.calculate, whose use is now discouraged.

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

**creationDate**

The document’s creation date. See info, which supersedes this property in later versions.

*Type: Date*  
*Access: R.*
creator

The creator of the document (for example, "Adobe FrameMaker", "Adobe PageMaker", and so on). See info, which supersedes this property in later versions.

Type: String Access: R.

dataObjects

An array containing all the named data objects in the document. See also the Data Object, dataObjects, createDataObject, exportDataObject, getDataObject, importDataObject, and removeDataObject.

Type: Array Access: R.

Example

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

delay

This boolean property can delay the redrawing of any appearance changes to every field in the document. It is generally used to buffer a series of changes to fields before requesting that the fields regenerate their appearance. When true, forces all changes to be queued until delay is reset to false. Once set to false, all the fields on the page are redrawn.

See also the field.delay property.

Type: Boolean Access: R/W.

dirty

This boolean property can be used to determine whether the document has been dirtied as the result of a changes to the document, and therefore needs to be saved. It is useful to reset the dirty flag in a document when performing changes that do not warrant saving, for example, updating a status field in the document.
disclosed

A boolean property that determines whether the document should be accessible to JavaScripts in other documents.

The two methods `app.openDoc` and `app.activeDocs` check the `disclosed` property of the document before returning its `Doc Object`.

**Note:** (Security $\S$): The `disclosed` property can only be set during batch, console, Page/Open and Doc/Open events. See the Event Object for a discussion of Acrobat JavaScript events.

**Example**

A document can be disclosed to others by placing the code at the document level (or as a page open action) at the top level:

```javascript
this.disclosed = true;
```

documentFileName

The base filename with extension of the document referenced by the `doc` object. The device-independent path is not returned. See also `path` and `filesize`.

**Example**

Executing the script

```javascript
console.println("The filename of this document is "+this.documentFileName);
```

on this document, the Acrobat JavaScript Scripting Reference, yields

"The filename of this document is AcroJS.pdf".
external

Whether the current document is being viewed in the Acrobat application or in an external window (such as a web browser).

Type: Boolean  Access: R.

Example

```javascript
if ( this.external )
{
    // viewing from a browser
}
else
{
    // viewing in the Acrobat application.
}
```

filesize

The file size of the document in bytes.

Type: Integer  Access: R.

Example (Version 5.0)

Get a readout of difference is file sizes before and after saving a document.

```javascript
// add the following code to the "Document Will Save" section
var filesizeBeforeSave = this.filesize
console.println("File size before saving is " + filesizeBeforeSave);

// add the following code to the "Document Did Save" section
var filesizeAfterSave = this.filesize
var difference = filesizeAfterSave - filesizeBeforeSave;
console.println("The difference is " + difference );
if ( difference < 0 )
    console.println("Reduced filesize!");
else
    console.println("Increased filesize!");
```

icons

An array of named Icon Generic Objects that are present in the document level named icons tree.
See also `addIcon`, `getIcon`, `importIcon`, `removeIcon`, the `Field Object` properties `buttonGetIcon`, `buttonImportIcon`, `buttonSetIcon`, and the `Icon Generic Object`.

Type: Array  
Access: R.

Example 1
```
if (this.icons == null)
    console.println("No named icons in this doc");
else
    console.println("There are " + this.icons.length + " named icons in this doc");
```

Example 2
```
// list all named icons
for (var i = 0; i < this.icons.length; i++) {
    console.println("icon[" + i + "]=" + this.icons[i].name);
}
```

info

In Adobe Reader

For the Adobe Reader, returns an object with properties from the document information dictionary in the PDF file. Standard entries are:

- Title
- Author
- Subject
- Keywords
- Creator
- Producer
- CreationDate
- ModDate
- Trapped

See Table 8.2, “Entries in a document information dictionary,” in the PDF Reference, for more details.

Writing to any property in this object in the Adobe Reader throws an exception.

Type: object  
Access: R.

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

For Acrobat, properties of the `info` object are writeable, and setting a property in this object will dirty the document. Additional document information fields can be added by setting non-standard properties.

**NOTE:** Standard entries are case insensitive, that is, `doc.info.Keywords` is the same as `doc.info.keywords`.

*Type: object*  
*Access: R/W.*

**Example**

The following script

```javascript
this.info.Title = "JavaScript, The Definitive Guide";
this.info.ISBN = "1-56592-234-4";
this.info.PublishDate = new Date();
for (var i in this.info)
    console.println(i + " : " + this.info[i]);
```

could produce the following output:

- **CreationDate:** Mon Jun 12 14:54:09 GMT-0500 (Central Daylight Time) 2000
- **Producer:** Acrobat Distiller 4.05 for Windows
- **Title:** JavaScript, The Definitive Guide
- **Creator:** FrameMaker 5.5.6p145
- **ModDate:** Wed Jun 21 17:07:22 GMT-0500 (Central Daylight Time) 2000
- **SavedBy:** Adobe Acrobat 4.0 Jun 19 2000
- **PublishDate:** Tue Aug 8 10:49:44 GMT-0500 (Central Daylight Time) 2000
- **ISBN:** 1-56592-234-4

**keywords**

The keywords that describe the document (for example, "forms", "taxes", "government"). See `info`, which supersedes this property in later versions.

**NOTE:** This property is read-only in the Adobe Reader.

*Type: object*  
*Access: R/W.*
### layout

Changes the page layout of the current document. Valid values are:

- SinglePage
- OneColumn
- TwoColumnLeft
- TwoColumnRight

In Acrobat 6.0, there are two additional properties:

- TwoPageLeft
- TwoPageRight

_Type: String  
_Access: R/W._

### metadata

Allows you to access the XMP metadata embedded in a PDF document. Returns a string containing the XML text stored as metadata in a particular PDF document. For information on embedded XMP metadata, see section 9.6 of the PDF Reference. This property throws a _RaiseError_ if the user tries to set the property to a string that is not in the XMP metadata format.

_Type: String  
_Access: R/W._

**Exceptions**

_RaiseError_ is thrown if setting metadata to a string not in XMP format.

**Example 1**

Try to create metadata not in XMP format.

```javascript
this.metadata = "this is my metadata";
RaiseError: The given metadata was not in the XMP format
Global.metadata:1:Console undefined:Exec
===> The given metadata was not in the XMP format
```

**Example 2**

Create a PDF report file with metadata from a document.

```javascript
var r = new Report();
r.writeln(this.metadata);
r.open("myMetadataReportFile");
r.save(); // save or mail the metadata report
r.mail();
```
modDate

The date the document was last modified. See info, which supersedes this property in later versions.

*Type:* Date  
*Access:* R.

numFields

The total number of fields in the document. See also getNthFieldName.

*Type:* Integer  
*Access:* R.

**Example**

```javascript
console.println("There are "+this.numFields+" in this document");
```

numPages

The number of pages in the document.

*Type:* Integer  
*Access:* R.

**Example**

```javascript
console.println("There are "+this.numPages+" in this document");
```

numTemplates

The number of templates in the document. See templates, which supersedes this property in later versions.

*Type:* Integer  
*Access:* R.

path

The device-independent path of the document, for example /c/Program Files/Adobe/Acrobat 5.0/Help/AcroHelp.pdf. See Section 3.10.1, “File Specification Strings”, in the PDF Reference for exact syntax of the path.

*Type:* String  
*Access:* R.
**pageNum**

Gets or sets a page of the document. When setting the `pageNum` to a specific page, remember that the values are 0-based.

*Type: Integer  Access: R/W.*

**Example**

This example goes to the first page of the document.

```javascript
this.pageNum = 0;
```

This example advances the document to the next page.

```javascript
this.pageNum++;
```

**permStatusReady**

Indicates whether the permissions for this document have been resolved. This can return `false` if the document is not available, for example when downloading over a network connection, and permissions are determined based on a signature that covers the entire document. Such documents will be signed with an author signature.

*Type: Boolean  Access: R.*

**producer**

The producer of the document (for example, ”Acrobat Distiller”, ”PDFWriter”, and so on). See `info`, which supersedes this property in later versions.

*Type: String  Access: R.*

**securityHandler**

The name of the security handler used to encrypt the document. Returns `null` if there is no security handler (for instance, the document is not encrypted).

*Type: String | null  Access: R.*

**Example**

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

This could print out the following if the document was encrypted with the standard security handler.

This document is encrypted with Standard security.

**selectedAnnots**

An array of Annot Objects corresponding to every markup annotation the user currently has selected.

See also `getAnnot` and `getAnnots`.

*Type: Array  Access: R.*

**Example**

Show all the comments of selected annots in console.

```javascript
var aAnnots = this.selectedAnnots;
for (var i=0; i < aAnnots.length; i++)
    console.println(aAnnots[i].contents);
```

**sounds**

An array containing all of the named Sound Objects in the document.

See also `getSound`, `importSound`, `deleteSound`, and the Sound Object.

*Type: Array  Access: R.*

**Example**

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

5.0

Gets or sets the dictionary array search order for this document. For example, if a user is filling out a Medical Form the form designer may want to specify a Medical dictionary to be searched first before searching the user’s preferred order.

The Spelling plug-in searches for words first in this array, and then searches the dictionaries the user has selected on the Spelling Preference panel. The user’s preferred order is available from spell.dictionaryOrder. An array of the currently installed dictionaries can be obtained using spell.dictionaryNames.

**NOTE:** When setting this property, an exception is thrown if any of the elements in the array is not a valid dictionary name.

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

spellLanguageOrder

6.0

This property can be used to access or specify the language array search order for this document. The Spelling plug-in will search for words first in this array and then in will search the languages the user has selected on the Spelling Preferences panel. The user’s preferred order is available from the spell.languageOrder. An array of currently installed languages can be obtained using the spell.languages property.

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

subject

The document’s subject. See info, which supersedes this property in later versions.

**NOTE:** This property is read-only in Adobe Reader.

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

templates

An array of all of the Template Objects in the document. See also createTemplate, getTemplate an removeTemplate.
Doc Properties

**Type:** Array  
**Access:** R.

**Example**

List all templates in the document.

```javascript
var t = this.templates
for ( var i=0; i<t.length; i++)
{
    var state = (t[i].hidden) ? "visible" : "hidden"
    console.println("Template: ", t[i].name + ", current state: " + state);
}
```

**title**

The title of the document. See info, which supersedes this property in later versions.

**NOTE:** This property is read-only in Adobe Reader.

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

**URL**

The document’s URL. If the document is local, returns a URL with a file:// scheme. This may be different from the baseURL.

**Type:** String  
**Access:** R.

**zoom**

Gets or sets the current page zoom level. Allowed values are between 8.33% and 6400%, specified as an percentage number, for example, a zoom value of 100 specifies 100%.

**Type:** Number  
**Access:** R/W.

**Example**

This example zooms in to twice the current zoom level.

```javascript
this.zoom *= 2;
```

This sets the zoom to 200%.

```javascript
this.zoom = 200;
```
zoomType

The current zoom type of the document. Valid zoom types are:

- none
- fit page
- fit width
- fit height
- fit visible width
- ReflowWidth (Acrobat 6.0).

A convenience `zoomtype` object defines all the valid zoom types. It provides the following zoom types:

<table>
<thead>
<tr>
<th>Zoom Type</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoVary</td>
<td>zoomtype.none</td>
</tr>
<tr>
<td>FitPage</td>
<td>zoomtype.fitP</td>
</tr>
<tr>
<td>FitWidth</td>
<td>zoomtype.fitW</td>
</tr>
<tr>
<td>FitHeight</td>
<td>zoomtype.fitH</td>
</tr>
<tr>
<td>FitVisibleWidth</td>
<td>zoomtype.fitV</td>
</tr>
<tr>
<td>Preferred</td>
<td>zoomtype.pref</td>
</tr>
<tr>
<td>ReflowWidth</td>
<td>zoomtype.refW</td>
</tr>
</tbody>
</table>

Type: String  
Access: R/W.

Example

This example sets the zoom type of the document to fit the width.

```javascript
this.zoomType = zoomtype.fitW;
```

Doc Methods

**addAnnot**

This creates an `annot` object having the specified properties. Properties not specified are given their default values for the specified `type` of annotation.
Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectLiteral</td>
<td>A generic object which specifies the properties of the annot object, such as type, rect, and page, to be created.</td>
</tr>
</tbody>
</table>

Returns

The new Annot Object.

Example 1

This example creates a "Square" annotation.

```javascript
var sqannot = this.addAnnot({type: "Square", page: 0});
```

This is a minimal example; sqannot will be created as annotation of type "Square" located on the first page (0-based page numbering).

Example 2

```javascript
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

Creates a new form field and returns it as a Field Object.

**NOTE:** (F, version 6.0): Beginning with version 6.0, doc.addField can now be used from within Adobe Reader for documents with “Advanced Form Features”.

Parameters

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of the new field to create. This name can use the dot separator syntax to denote a hierarchy (for example, name.last creates a parent node, name, and a child node, last).</td>
</tr>
<tr>
<td>cFieldType</td>
<td>The type of form field to create. Valid types are: text, button, combobox, listbox, checkbox, radiobutton, signature</td>
</tr>
</tbody>
</table>
**$nPageNum$**  The 0-based index of the page to which to add the field.

**oCoords**  An array of four numbers in *rotated user space* that specifies the size and placement of the form field. These four numbers are the coordinates of the bounding rectangle, in the following order: upper-left x, upper-left y, lower-right x and lower-right y. See also `field.rect`.

**Note:** If you use the **Info** panel to obtain the coordinates of the bounding rectangle, you must transform them from **info space** to *rotated user space*. To do this, subtract the info space y-coordinate from the onscreen page height.

---

**Returns**

The newly created **Field Object**.

**Example**

The following code might be used in a batch sequence to create a navigational icon on every page of a document, for each document in a selected set of documents.

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

  // 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.borderWidth = border.s;
  f.highlight = "push";
  f.fontSize = 0;  // auto sized
  f.textColor = color.blue;
  f.fillColor = color.ltGray;
  f.textFont = font.ZapfD
  f.buttonSetCaption("\341") // a right arrow
  f.delay = false;
}
```

See `field.setAction` for another example.
addIcon

Add a new named Icon Generic Object to the document-level icon tree, storing it under the specified name.

See also icons, getIcon, importIcon, removeIcon, and the field methods buttonGetIcon, buttonImportIcon, and buttonSetIcon.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of the new object</td>
</tr>
<tr>
<td>icon</td>
<td>The Icon Generic Object to add.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

This example takes an icon already attached to a form button field in the document and assigns a name to it. This name can be used to retrieve the icon object with a getIcon for use in another button, for example.

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

addLink

Add a new link to the specified page with the specified coordinates, if the user has permission to add links to the document. See also getLinks, removeLinks and the Link Object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>The page on which to add the new link.</td>
</tr>
<tr>
<td>oCoords</td>
<td>An array of four numbers in rotated user space that specifies the size and placement of the link. These four numbers are the coordinates of the bounding rectangle, listed in the following order: upper-left x, upper-left y, lower-right x and lower-right y.</td>
</tr>
</tbody>
</table>

Returns

The newly created Link Object.
Example 1

Create simple navigational links in the lower left and right corners of each page of the current document. The link in lower left corner goes to the previous page; the one in the lower right corner goes to the next page.

```javascript
var linkWidth = 36, linkHeight = 18;
for ( var i=0; i < this.numPages; i++)
{
    var cropBox = this.getPageBox("Crop", i);
    var linkRect1 = [0,linkHeight,linkWidth,0];
    var offsetLink = cropBox[2] - cropBox[0] - linkWidth;
    var linkRect2 = [offsetLink,linkHeight,linkWidth + offsetLink,0]
    var lhLink = this.addLink(i, linkRect1);
    var rhLink = this.addLink(i, linkRect2);
    var nextPage = (i + 1) % this.numPages;
    var prevPage = (i - 1) % this.numPages;
    var prevPage = (prevPage>=0) ? prevPage : -prevPage;
    lhLink.setAction( "this.pageNum = " + prevPage);
    lhLink.borderColor = color.red;
    lhLink.borderWidth = 1;
    rhLink.setAction( "this.pageNum = " + nextPage);
    rhLink.borderColor = color.red;
    rhLink.borderWidth = 1;
}

See the Link Object for setting the properties and for setting the action of a link.

Example 2

Search through the document for the word “Acrobat” and create a link around that word.

```javascript
for ( var p = 0; p < this.numPages; p++)
{
    var numWords = this.getPageNumWords(p);
    for ( var i=0; i<numWords; i++)
    {
        var ckWord = this.getPageNthWord(p, i, true);
        if ( ckWord == "Acrobat")
        {
            var q = this.getPageNthWordQuads(p, i);
            // convert quads in default user space to rotated
            // user space used by Links.
            m = (new Matrix2D).fromRotated(this,p);
            mInv = m.invert()
            r = mInv.transform(q)
            r=r.toString()
            r = r.split(",");
            l = addLink(p, [r[4], r[5], r[2], r[3]]);
            l.borderColor = color.red
            l.borderWidth = 1
            l.setAction("this.getURL('http://www.adobe.com/');");
        }
    }
}
The **Matrix2D** object and its methods are defined in the *Annots.js* file.

### addRecipientListCryptFilter

**6.0**

This method adds a crypt filter to this document. The crypt filter is used for encrypting **Data Objects**.

See also the **cCryptFilter** parameter of the **importDataObject** and **createDataObject** methods.

**Note:** (Security ⑤): Can only be executed during batch, application initialization, menu or console events. Not available in the Adobe Reader.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cCryptFilter</strong></td>
<td>The language independent name of the crypt filter. This same name should be used as the value of the <strong>cCryptFilter</strong> parameter of the <strong>Doc Object importDataObject</strong> and <strong>createDataObject</strong> methods.</td>
</tr>
<tr>
<td><strong>oGroup</strong></td>
<td>An array of <strong>Group Objects</strong> that lists the recipients for whom the data is to be encrypted.</td>
</tr>
</tbody>
</table>

#### Returns

Nothing

#### Example

This script takes the current document open in the viewer, and encrypts and embeds the document into a "ePaper" envelope PDF document. This script was executed in the console, but is perhaps best executed a folder JavaScript as part of larger script for sending PDF docs in a secure way.

```javascript
var Note = "Select the list of people that you want to send this" + " document to. Each person must have both an email address" + " and a certificate that you can use when creating the" + " envelope.";
var oOptions = { bAllowPermGroups: false, cNote: Note, bRequireEmail: true };
var oGroups = security.chooseRecipientsDialog( oOptions );
var env = app.openDoc( "/c/temp/ePaperMailEnvelope.pdf" );
env.addRecipientListCryptFilter( "MyFilter", oGroups );
env.importDataObject( "secureMail0", this.path, "MyFilter" );
var envPath = "/c/temp/outMail.pdf";
env.saveAs( envPath );
```
**addScript**


### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of the script that will be added. If a script with this name already exists, the new script replaces the old one.</td>
</tr>
<tr>
<td>cScript</td>
<td>The JavaScript expression that is to be executed when the document is opened.</td>
</tr>
</tbody>
</table>

### Returns

Nothing

### Example

Create a beeping sound every time the document is opened.

```javascript
this.addScript("My Code", "app.beep(0);" );
```

**addThumbnails**

Creates thumbnails for the specified pages in the document. See also `removeThumbnails`.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nStart</td>
<td>(optional) A 0-based index that defines the start of an inclusive range of pages. If <code>nStart</code> and <code>nEnd</code> are not specified then the range of pages is for all pages in the document. If only <code>nStart</code> is specified then the range of pages is the single page specified by <code>nStart</code>.</td>
</tr>
<tr>
<td>nEnd</td>
<td>(optional) A 0-based index that defines the end of an inclusive range of pages. If <code>nStart</code> and <code>nEnd</code> are not specified then the range of pages is for all pages in the document. If only <code>nEnd</code> is specified then the range of a pages is 0 to <code>nEnd</code>.</td>
</tr>
</tbody>
</table>

### Returns

Nothing
addWeblinks

Scans the specified pages looking for instances of text with an `http:` scheme and converts them into links with URL actions. See also `removeWeblinks`.

**Parameters**

- **nStart** (optional) A 0-based index that defines the start of an inclusive range of pages. If `nStart` and `nEnd` are not specified then the range of pages is for all pages in the document. If only `nStart` is specified then the range of pages is the single page specified by `nStart`.
- **nEnd** (optional) A 0-based index that defines the end of an inclusive range of pages. If `nStart` and `nEnd` are not specified then the range of pages is for all pages in the document. If only `nEnd` is specified then the range of a pages is 0 to `nEnd`.

**Returns**

The number of web links added to the document.

**Example**

```javascript
var numWeblinks = this.addWeblinks();
console.println("There were " + numWeblinks + " instances of text that looked like a web address," +" and converted as such.");
```

bringToFront

Brings the document open in the Viewer to the front, if it is not already there.

**Parameters**

None

**Returns**

Nothing

**Example**

This example searches among the documents open in the Viewer for the document with a title of "Annual Report" and brings it to the front.

```javascript
var d = app.activeDocs; // lists only disclosed documents
for (var i = 0; i < d.length; i++)
    if (d[i].info.Title == "Annual Report") d[i].bringToFront();
```
calculateNow

Forces computation of all calculation fields in the current document.

Parameters

None

Returns

Nothing

closeDoc

Closes the document.

NOTE: (Document Save Rights): For Adobe 5.1 Reader or later, the method is always allowed. However, if the document was changed and no Document Save Rights are available, the document is closed without any warnings and changes are lost. If Document Save Rights are available, the user gets the option of saving the changed file. It is important to use this method carefully as it is an abrupt change in the document state that can affect any JS executing after the close. Triggering this method from a Page event or Document event could cause the application to behave strangely.

Parameters

bNoSave (optional) Whether to close the document without saving. If false (the default), the user is prompted to save the document if it has been modified. If true, the document is closed without prompting the user and without saving, even if the document has been modified. Because this can cause data loss without user approval, use this feature judiciously.

Returns

false

Example

Create a series of three test files and save them to a directory. This code needs to be executed in the console, because saveAs has a security restriction.

```javascript
var myDoc = app.newDoc();
for (var i=0; i<3; i++) {
    myDoc.info.Title = "Test File " + i;
    myDoc.saveAs("/c/temp/test"+i+.pdf);
}
myDoc.closeDoc(true);
```
See `saveAs` for another example of `closeDoc`.

### createDataObject

Creates a Data Object.

Data objects can be constructed *ad hoc*. This is useful if the data is being created in JavaScript from sources other than an external file (for example, ADBC database calls). See also `dataObjects`, `exportDataObject`, `getDataObject`, `importDataObject`, `removeDataObject`, and the Data Object.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cName</code></td>
<td>The name to associate with the data object.</td>
</tr>
<tr>
<td><code>cValue</code></td>
<td>A string containing the data to be embedded.</td>
</tr>
<tr>
<td><code>cMIMEType</code></td>
<td>(optional) The MIME type of the data. Default is &quot;text/plain&quot;.</td>
</tr>
<tr>
<td><code>cCryptFilter</code></td>
<td>(optional, version 6.0) The language independent name of a crypt filter to use when encrypting this data object. This crypt filter must have previously been added to the document’s list of crypt filters, using the Doc Object <code>addRecipientListCryptFilter</code> method, otherwise an exception will be thrown. The predefined &quot;Identity&quot; crypt filter can be used if it is desired that this data object not be encrypted in a file that is otherwise encrypted by the Doc Object <code>encryptForRecipients</code> method.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**Example**

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

See also the example that follows `addRecipientListCryptFilter`.

### createTemplate

Creates a visible template from the specified page. See also `templates`, the `getTemplate`, `removeTemplate`, and the Template Object.

**Note:** (Security ⚠️): This method can only be executed during batch, console, or menu events. See the Event Object for a discussion of Acrobat JavaScript events.
### createTemplate

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name to be associated with this page.</td>
</tr>
<tr>
<td>nPage</td>
<td>(optional) The 0-based index of the page to operate on. Default is 0, the first page in the document.</td>
</tr>
</tbody>
</table>

**Returns**

The newly created Template Object.

**Example**

Convert all pages beginning with page 2 (base 0) to hidden templates. We have to be a little careful, as the templates are hidden, `this.numPages` is updated to reflect that change in the number of (visible) pages. Notice that in the loop below, only page 2 is made a template then hidden; the next page will become the new page 2.

```javascript
numNewTemplates = this.numPages - 2;
for ( var i = 0; i < numNewTemplates; i++)
{
    var t = this.createTemplate({cName:"myTemplate"+i, nPage:2 });
    t.hidden = true;
}
```

### deletePages

**Parameters**

- nStart: (optional) The 0-based index of the first page in the range of pages to be deleted. Default is 0, the first page in the document.
- nEnd: (optional) The last page in the range of pages to be deleted. If nEnd is not specified then only the page specified by nStart is deleted.

**Returns**

Nothing

Delete pages from the document. If neither page of the range is specified, the first page (page 0) is deleted. See also `insertPages`, `extractPages` and `replacePages`.

**NOTE:** You cannot delete all pages in a document: there must be at least one page remaining.

**NOTE:** (F, version 6.0): Beginning with version 6.0, `doc.deletePages` can now delete `spawned` pages from within Adobe Reader for documents with “Advanced Form Features”.
Delete pages 1 through 3 (base 0), inclusive

```javascript
    this.deletePages({nStart: 1, nEnd: 3});
```

### deleteSound

5.0

Deletes the sound object with the specified name from the document. See also `sounds`, `getSound`, `importSound`, and the `Sound Object`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of the sound object to delete.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**Example**

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

### encryptForRecipients

6.0

Encrypts the document for the specified lists of recipients, using the public-key certificates of each recipient. Encryption does not take place until the document is saved. Recipients can be placed into groups, and each group can have its own unique permission settings. This method throws an exception if it is unsuccessful.

**NOTE:** (Security soared): This method is available from batch, console, app initialization and menu events. It is also available in the Adobe Reader.

See also:

- The `security.chooseRecipientsDialog` method.
- The `Data Object`.
- `createDataObject`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oGroups</td>
<td>(optional) An array of generic <code>Group Objects</code> that list the recipients for which the document is to be encrypted.</td>
</tr>
</tbody>
</table>
**Group Object**

A generic JS object that allows a set of permissions to be attached to a list of recipients for which a document or data is to be encrypted. This object is passed to `doc.encryptForRecipients`, and returned by `security.chooseRecipientsDialog`. It contains the following properties:

### Permissions Object

A generic JS object that contains a set of permissions, used in a Group Object. It contains the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowAll</td>
<td>Boolean</td>
<td>R/W</td>
<td>Whether full, unrestricted access is permitted. If true, overrides all other properties.</td>
</tr>
<tr>
<td>allowAccessibility</td>
<td>Boolean</td>
<td>R/W</td>
<td>Whether content access for the visually impaired is permitted. When true, allows content to be extracted for use by applications that, for example, read text aloud.</td>
</tr>
<tr>
<td>allowContentExtraction</td>
<td>Boolean</td>
<td>R/W</td>
<td>Whether content copying and extraction is permitted.</td>
</tr>
</tbody>
</table>
Example

Encrypt all strings and streams in the document. This will produce a file that can be opened with Acrobat 5.0

```javascript
var sh = security.getHandler("Adobe.PPKMS");
var dir = shdirectories[0];
var dc = dir.connect();

dc.setOutputFields({oFields: ["certificates"]});
var importantUsers = dc.search({oParams: {lastName: "Smith"}});
var otherUsers = dc.search({oParams: {lastName: "jones"}});

this.encryptForRecipients({
  oGroups: [
    {oCerts: importantUsers,
      oPermissions: {allowAll: true}},
    {oCerts: otherUsers,
      oPermissions: {allowPrinting: "highQuality"}}]
}, bMetaData); // Score 6.0
```

**exportAsText**

Exports form fields as a tab-delimited text file to a local hard disk. The text file that is created follows the conventions specified by Microsoft Excel. In particular, `exportAsText` correctly handles quotes and multiline text fields.

**NOTE:** (Security): If the `cPath` parameter is specified, this method can only be executed during batch, console or menu events. See the Event Object for a discussion of Acrobat JavaScript events.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bNoPassword</td>
<td>(optional) If true (the default), do not include text fields that have the &quot;password&quot; flag set in the exported XFDF.</td>
</tr>
<tr>
<td>aFields</td>
<td>(optional) The array of field names to submit or a string containing a single field name.</td>
</tr>
<tr>
<td></td>
<td>- If specified, only these fields are exported, except those excluded by bNoPassword.</td>
</tr>
<tr>
<td></td>
<td>- If aFields is an empty array, no fields are exported.</td>
</tr>
<tr>
<td></td>
<td>- If this parameter is omitted or is null, all fields in the form are exported, except those excluded by bNoPassword.</td>
</tr>
<tr>
<td>cPath</td>
<td>(optional) A string specifying the device-independent pathname for the file. (See Section 3.10.1 of the PDF Reference for a description of the device-independent pathname format.) The pathname may be relative to the location of the current document. If the parameter is omitted a dialog is shown to let the user select the file.</td>
</tr>
<tr>
<td></td>
<td>- Note: (Security[S]): The parameter cPath is required to have a Safe Path and have a .txt extension. This method will throw a NotAllowedError (see the Error Objects) exception if these security conditions are not met, and the method will fail.</td>
</tr>
</tbody>
</table>

Returns

Nothing

exportAsFDF

Exports form fields as an FDF file to the local hard drive.

- Note: (Security[S]): If the cPath parameter is specified, then this method can only be executed during batch, console, or menu events. See the Event Object for a discussion of Acrobat JavaScript events.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bAllFields</td>
<td>(optional) If true, all fields are exported, including those that have no value. If false (the default), excludes those fields that currently have no value.</td>
</tr>
<tr>
<td>bNoPassword</td>
<td>(optional) If true (the default), do not include text fields that have the &quot;password&quot; flag set in the exported FDF.</td>
</tr>
</tbody>
</table>
### `exportAsFDF` Method

Exports the current form as a FDF file, including fields and annotations if specified.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| `aFields` | (optional) The array of field names to submit or a string containing a single field name.  
  - If specified, only these fields are exported, except those excluded by `bNoPassword`.  
  - If `aFields` is an empty array, no fields are exported. The FDF might still contain data, depending on the `bAnnotations` parameter.  
  - If this parameter is omitted or is `null`, all fields in the form are exported, except those excluded by `bNoPassword`.  
  Specify non-terminal field names to export an entire subtree of fields; see the example below. |
| `bFlags` | (optional) If `true`, field flags are included in the exported FDF. The default is `false` |
| `cPath` | (optional) A string specifying the device-independent pathname for the file. (See Section 3.10.1 of the PDF Reference for a description of the device-independent pathname format.) The pathname may be relative to the location of the current document. If the parameter is omitted a dialog is shown to let the user select the file.  
  **Note:** (Security): The parameter `cPath` is required to have a Safe Path and have a `.fdf` extension. This method will throw a `NotAllowedError` (see the Error Objects) exception if these security conditions are not met, and the method will fail. |
| `bAnnotations` | (optional, version 6.0) If `true`, annotations are included in the exported FDF. The default is `false` |

**Returns**

Nothing

**Example 1**

Export the entire form (including empty fields) with flags.

```javascript
this.exportAsFDF(true, true, null, true);
```

**Example 2**

Export the `name` subtree with no flags.

```javascript
this.exportAsFDF(false, true, "name");
```

The example above illustrates a shortcut to exporting a whole subtree. Passing "name" as part of the `aFields` parameter, exports "name.title", "name.first", "name.middle" and "name.last", and so on.
exportAsXFDF

Exports form fields an XFDF file to the local hard drive. XFDF is an XML representation of Acrobat form data. See the Acrobat CD Documentation “Forms System Implementation Notes” for details.

**Note:** (Security): If the `cPath` parameter is specified, then this method can only be executed during batch, console or menu events. See the Event Object for a discussion of Acrobat JavaScript events.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bAllFields</td>
<td>(optional) If <code>true</code>, all fields are exported, including those that have no value. If <code>false</code> (the default), excludes those fields that currently have no value.</td>
</tr>
<tr>
<td>bNoPassword</td>
<td>(optional) If <code>true</code> (the default), do not include text fields that have the &quot;password&quot; flag set in the exported XFDF.</td>
</tr>
</tbody>
</table>
| aFields         | (optional) The array of field names to submit or a string containing a single field name.  
  - If specified, only these fields are exported, except those excluded by `bNoPassword`.  
  - If `aFields` is an empty array, no fields are exported. The XFDF might still contain data, depending on the `bAnnotations` parameter.  
  - If this parameter is omitted or is `null`, all fields in the form are exported, except those excluded by `bNoPassword`.  
  Specify non-terminal field names to export an entire subtree of fields; see the example below. |
| cPath           | (optional) A string specifying the device-independent pathname for the file. (See Section 3.10.1 of the PDF Reference for a description of the device-independent pathname format.) The pathname may be relative to the location of the current document. If the parameter is omitted a dialog is shown to let the user select the file.  
  **Note:** (Security): The parameter `cPath` is required to have a Safe Path and have a `.xfdf` extension. This method will throw a `NotAllowedError` (see the Error Objects) exception if these security conditions are not met, and the method will fail. |
| bAnnotations    | (optional, version 6.0) If `true`, annotations are included in the exported XFDF. The default is `false`. |
Returns

Nothing

exportDataObject

This method extracts the specified data object to an external file. See also dataObjects, createDataObject, getDataObject, importDataObject, removeDataObject, and the Data Object.

NOTES: (Security): Beginning with Acrobat 6.0, if the parameter cDIPath is non-NULL a NotAllowedError (see the Error Objects) exception will be thrown and the method will fail.

If cDIPath is not passed to this method, a file selection dialog will open to allow the user to select a save path for the embedded data object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of the data object to extract.</td>
</tr>
<tr>
<td>cDIPath</td>
<td>(optional) A device-independent path to which to extract the data object.</td>
</tr>
<tr>
<td></td>
<td>This path may be absolute or relative to the current document.</td>
</tr>
<tr>
<td></td>
<td>If not specified, the user is prompted to specify a save location. See</td>
</tr>
<tr>
<td></td>
<td>“File Specification Strings” in the PDF Reference Manual for the exact</td>
</tr>
<tr>
<td></td>
<td>syntax of the path.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE:</strong> (version 6.0) The use of this parameter is no longer supported</td>
</tr>
<tr>
<td></td>
<td>and should not be used. See the security notes above.</td>
</tr>
<tr>
<td>bAllowAuth</td>
<td>(optional, version 6.0) If true, a dialog is used to obtain user</td>
</tr>
<tr>
<td></td>
<td>authorization. Authorization may be required if the data object was</td>
</tr>
<tr>
<td></td>
<td>encrypted using Doc. encryptForRecipients. Authorization dialogs are</td>
</tr>
<tr>
<td></td>
<td>allowed if bAllowAuth is true. The default value is false.</td>
</tr>
</tbody>
</table>
nLaunch (optional, version 6.0) nLaunch controls whether the file is launched, or opened, after it is saved. Launching may involve opening an external application if the file is not a PDF file. The values of nLaunch are
- If the value is 0, the file will not be launched after it is saved.
- If the value is 1, the file will be saved and then launched.
  Launching will prompt the user with a security alert warning if the file is not a PDF file. The user will be prompted for a save path.
- If the value is 2, the file will be saved and then launched.
  Launching will prompt the user with a security alert warning if the file is not a PDF file. A temporary path is used, and the user will not be prompted for a save path. The temporary file that is created will be deleted by Acrobat upon application shutdown.

The default value is 0.

Returns
Nothing

Example 1
Prompt the user for a file and location to extract to.

```javascript
this.exportDataObject("MyData");
```

Example 2 (Version 6.0)
Extract PDF document and launch it in the viewer.

```javascript
this.exportDataObject({ cName: "MyPDF", nLaunch: 2 });
```

exportXFADData

Exports an XFA data file to the local hard drive.

Form Rights (F): When exporting XFA data from the Adobe Reader, the document must have export form rights.

NOTE: (Security S): If the cPath parameter is specified, then this method can only be executed during batch, console or menu events. See the Event Object for a discussion of Acrobat JavaScript events.
**extractPages**

Creates a new document consisting of pages extracted from the current document. If a page range is not specified, extracts all pages in the document.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cPath</strong></td>
<td>(optional) A device-independent pathname for the file. The pathname may be relative to the document. See “File Specification Strings” in the PDF Reference Manual for the exact syntax of the path. If this parameter is omitted, a dialog is shown to let the user select the file. <strong>Note:</strong> (Security): The parameter <code>cPath</code> is required to have a Safe Path. Additionally, the file name must have a .xdp extension, if <code>bXDP</code> is true, or a .xml extension, if <code>bXDP</code> is false. This method will throw a <code>NotAllowedError</code> (see the Error Objects) exception if these security conditions are not met, and the method will fail.</td>
</tr>
<tr>
<td><strong>bXDP</strong></td>
<td>(optional) If true (the default), the method exports in the XDP format. Otherwise, it exports in the plain XML data format.</td>
</tr>
<tr>
<td><strong>aPackets</strong></td>
<td>(optional) An array of strings specifying which packets to include in the XDP export. This parameter is only applicable if <code>bXDP</code> is true. Possible strings are: <code>template datasets stylesheet xfdf sourceSet pdf config *</code> <code>pdf</code> means that the PDF should be embedded. If <code>pdf</code> is not specified, only a link to the PDFs included in the XDP. <code>xfdf</code> means include annotations in the XDP (since that packet uses XFDF format). <code>*</code> means that all packets should be included in the XDP. The default for this parameter is: <code>&quot;datasets&quot;, &quot;xfdf&quot;]</code>. <strong>Note:</strong> (Save rights required): When exporting a document with from the Adobe Reader with <code>aPackets</code> set to <code>pdf</code> (or <code>*</code>, which implies <code>pdf</code>), the document must have document save rights.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing
See also deletePages, insertPages, and replacePages.

**NOTE:** (Security) If the cPath parameter is specified, then this method can only be executed during batch, console or menu events, or through an external call (for example, OLE). See the Event Object for a discussion of Acrobat JavaScript events.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nStart</td>
<td>(optional) A 0-based index that defines the start of the range of pages to extract from the source document. If only nStart is specified then the range of pages is the single page specified by nStart.</td>
</tr>
<tr>
<td>nEnd</td>
<td>(optional) A 0-based index that defines the end of the range of pages to extract from the source document. If only nEnd is specified then the range of pages is 0 to nEnd.</td>
</tr>
<tr>
<td>cPath</td>
<td>(optional) The device-independent pathname to which to save the new document. See 3.10.1 of the PDF Reference for a description of the device-independent path name format. The path name may be relative to the location of the current document.</td>
</tr>
</tbody>
</table>

**NOTE:** (Security): The parameter cPath is required to have a Safe Path and have a .pdf extension. This method will throw aNotAllowedError (see the Error Objects) exception if these security conditions are not met, and the method will fail.

**Returns**

If cPath is not specified, returns the Doc Object for the new document; otherwise, returns the null object.

**Example**

The following batch sequence would take each of the selected files and extract each page and save the page to a folder with an unique name. This example may be useful in the following setting. Clients one-page bills are produced by an application and placed in a single PDF file. It is desired to separate the pages for distribution and/or separate printing jobs.

```javascript
/* 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({
```
```javascript
nStart: i,
cPath: "/F/temp/"+filename+"_" + i +".pdf"
});
} catch (e) {
    console.println("Aborted: " + e)
}
```

### flattenPages

Converts all annotations in the specified page range to page contents. If a page range is not specified, converts annotation for all the pages in the current document.

**NOTE:** Great care must be used when using this method. All annotations—including form fields, comments and links—on the specified range of pages are flattened; they may have appearances, but they will no longer be annotations.

**Parameters**

- **nStart** *(optional)* A 0-based index that defines the start of an inclusive range of pages in the current document. If only `nStart` is specified, then the page range is the single page specified by `nStart`.

- **nEnd** *(optional)* A 0-based index that defines the end of an inclusive range of pages in the current document.

- **nNonPrint** *(optional, version 6.0)* This parameter determines how to handle non-printing annotations. Values are
  0 (default): Non-printing annotations are flattened.
  1: Non-printing annotations are left as is.
  2: Non-printing annotations are removed from the document.

**Returns**

Nothing

**Example**

Flatten all pages in the document.

```javascript
this.flattenPages();
```

### getAnnot

Gets the name of an `annot` object contained on a specific document page.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>The page that contains the desired Annot Object.</td>
</tr>
<tr>
<td>cName</td>
<td>The name of the desired Annot Object.</td>
</tr>
</tbody>
</table>

Returns

The Annot Object, or null if there is no such annotation.

Example

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

getAnnots

5.0 A X

Gets an array of Annot Objects satisfying specified criteria. See also getAnnot and syncAnnotScan.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) A 0-based page number. If specified, gets only annotations on the given page. If not specified, gets annotations that meet the search criteria from all pages.</td>
</tr>
</tbody>
</table>
| nSortBy   | A sort method applied to the array. Values are:  
            - ANSB_None: (default) Do not sort; equivalent to not specifying this parameter.  
            - ANSB_Page: Use the page number as the primary sort criteria.  
            - ANSB_Author: Use the author as the primary sort criteria.  
            - ANSB_ModDate: Use the modification date as the primary sort criteria.  
            - ANSB_Type: Use the annot type as the primary sort criteria. |
| bReverse  | (optional) If true, causes the array to be reverse sorted with respect to nSortBy. |
nFilterBy

Gets only annotations satisfying certain criteria. Values are:

- **ANFB_ShouldNone**: (default) Get all annotations. Equivalent of not specifying this parameter.
- **ANFB_ShouldPrint**: Only include annotations that can be printed.
- **ANFB_ShouldView**: Only include annotations that can be viewed.
- **ANFB_ShouldEdit**: Only include annotations that can be edited.
- **ANFB_ShouldAppearInPanel**: Only annotations that appear in the annotations pane.
- **ANFB_ShouldSummarize**: Only include annotations that can be included in a summarization
- **ANFB_ShouldExport**: Only include annotations that can be included in an export

Returns

An array of Annot Objects.

Example

```javascript
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));
```

**getDataObject**

Obtains a specific data object. See also `dataObjects`, `createDataObject`, `exportDataObject`, `getDataObject`, `importDataObject`, `removeDataObject`. 
Documentation for Adobe Acrobat JavaScript Scripting Reference (Doc Methods)

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cName</code></td>
<td>The name of the data object to obtain.</td>
</tr>
</tbody>
</table>

### Returns

The Data Object corresponding to the specified name.

### Example

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

### getField

Maps a Field Object in the PDF document to a JavaScript variable.

Beginning with Acrobat 6.0, this method can return the Field Object of an individual Widget. For more information, see Field Access from JavaScript.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cName</code></td>
<td>The name of the field of interest.</td>
</tr>
</tbody>
</table>

### Returns

A Field Object representing a form field in the PDF document.

### Example 1

Make a text field multiline and triple its height

```javascript
var f = this.getField("myText");
var aRect = f.rect; // get bounding rectangle
f.multiline = true; // make it multiline
var height = aRect[1]-aRect[3]; // calculate height
aRect[3] -= 2*height; // triple the height of the text field
f.rect = aRect; // and make it so
```

### Example 2

Attach a JavaScript action to an individual widget, in this case, a Radio Button.

```javascript
var f = this.getField("myRadio.0");
f.setAction("MouseUp",
    "app.alert('Thanks for selecting the first choice.');");
```
getIcon

Obtains a specific icon object. See also `icons`, `addIcon`, `importIcon`, and `removeIcon`, and `field` methods `buttonGetIcon`, `buttonImportIcon`, and `buttonSetIcon`.

### Parameters

| cName | The name of the icon object to obtain. |

### Returns

An Icon Generic Object associated with the specified name in the document or `null` if no icon of that name exists.

### Example

The following is a custom keystroke script from a combobox. The face names of the items in the combobox are the names of some of the icons that populate the document. As the user chooses different items from the combobox, the corresponding icon appears as the button face of the field "myPictures".

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

See `field.buttonSetIcon` for a more elaborate variation on this example.

getLegalWarnings

This method returns the legal warnings for this document in the form of an object with entries for each warning that has been found in the document. Legal warnings can be embedded in a file at the time that a file is signed by an author signature. Legal warnings can be embedded using the `cLegalAttest` of the `field.signatureSign` method.

The process that analyses a file to determine this list of warnings not available in the Adobe Reader. The value of each entry is the number of occurrences of this warning in the document. Refer to PDF Reference 1.5.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bExecute</td>
<td>If true, will cause the file to be examined and all detected warnings will be returned. If false, the default value, the warnings that have been embedded in the file will be returned.</td>
</tr>
</tbody>
</table>

Returns

A object containing property names and values of legal warnings

Example

Process a document and get legal PDF warnings.

```javascript
var w = this.getLegalWarnings( true );
console.println( "Actual Legal PDF Warnings:" );
for(i in w) console.println( i + " = " + w[i] );

var w1 = this.getLegalWarnings( false );
console.println( "Declared Legal PDF Warnings:" );
for(i in w1) console.println( i + " = " + w1[i] );

// For an author signature, note also if annotations are
// allowed by MDP settings

var f = this.getField( "AuthorSig" );
var s = f.signatureInfo();
if( s.mdp == "defaultAndComments" )
    console.println( "Annotations are allowed" );

// What does author have to say about all this?

console.println( "Legal PDF Attestation:" );
console.println( w1.Attestation );
```

getLinks

<table>
<thead>
<tr>
<th>Version</th>
<th>Compatibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0</td>
<td>X</td>
</tr>
</tbody>
</table>

Gets an array of link objects that reside on a specified page at specified coordinates. See also addLink, removeLinks and the Link Object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>The page that contains the desired link object. The first page is 0.</td>
</tr>
<tr>
<td>oCoords</td>
<td>An array of four numbers in rotated user space, the coordinates of a rectangle listed in the following order: upper-left x, upper-left y, lower-right x and lower-right y.</td>
</tr>
</tbody>
</table>
Returns

An array of Link Objects.

Example

Count the number of links in a document and report to the console.

```javascript
var numLinks=0;
for ( var p = 0; p < this.numPages; p++)
{
    var b = this.getPageBox("Crop", p);
    var l = this.getLinks(p, b);
    console.println("Number of Links on page " + p +" is "+l.length);
    numLinks += l.length;
}
console.println("Number of Links in Document is "+numLinks);
```

**getNthFieldName**

4.0

Gets the \( n \)th field name in the document. See also **numFields**.

**Parameters**

<table>
<thead>
<tr>
<th>nIndex</th>
<th>The field name to obtain.</th>
</tr>
</thead>
</table>

**Returns**

The name of the field in the document.

**Example**

Enumerate through all of the fields in the document.

```javascript
for (var i = 0; i < this.numFields; i++)
    console.println("Field[" + i + "] = " + this.getNthFieldName(i));
```

**getNthTemplate**

Gets the name of the \( n \)th template within the document.

This method is superseded by the **templates** property, the **getTemplate** method, and the **Template Object** in later versions.
**getOCGs**

Gets an array of OCG Objects found on a specified page.

**Parameters**

- **nPage** (optional) The 0-based page number. If not specified, all the OCGs found in the document are returned.

**Returns**

Returns an array of OCG Objects or **null** if no OCGs are present.

**Example**

Turn on all the OCGs on the given document and page.

```javascript
function TurnOnOCGsForPage(doc, nPage)
{
    var ocgArray = doc.getOCGs(nPage);
    for (var i=0; i < ocgArray.length; i++)
        ocgArray[i].state = true;
}
```

**getPageBox**

Gets a rectangle in rotated user space that encompasses the named box for the page. See also `setPageBoxes`.

**Parameters**

- **nIndex** The template to obtain.

**Returns**

The name of the specified template.
Parameters

<table>
<thead>
<tr>
<th>cBox</th>
<th>(optional) The type of box. Values are:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Art</td>
<td>Art</td>
</tr>
<tr>
<td>Bleed</td>
<td>Bleed</td>
</tr>
<tr>
<td>BBox</td>
<td>BBox</td>
</tr>
<tr>
<td>Crop</td>
<td>Crop (default)</td>
</tr>
<tr>
<td>Trim</td>
<td>Trim</td>
</tr>
</tbody>
</table>

For definitions of these boxes see “Page Boundaries” in the PDF Reference.

| nPage      | (optional) The 0-based index of the page. Default is 0, the first page in the document. |

Returns

A rectangle in rotated user space that encompasses the named box for the page.

Example

Get the dimensions of the Media box.

```
var aRect = this.getPageBox("Media");
var width = aRect[2] - aRect[0];
var height = aRect[1] - aRect[3];
console.println("Page 1 has a width of " + width + " and a height of " + height);
```

getPageLabel

5.0

Gets page label information for the specified page.

Parameters

| nPage      | (optional) The 0-based index of the page. Default is 0, the first page in the document. |

Returns

Page label information for the specified page.

Example

See setPageLabels for an example.
getPageNthWord

```
5.0
S
```

Gets the nth word on the page. See also `getPageNumWords` and `selectPageNthWord`.

**NOTE:** (Security ⪯): This method throws an exception if the document security is set to prevent content extraction.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) The 0-based index of the page. Default is 0, the first page in the document.</td>
</tr>
<tr>
<td>nWord</td>
<td>(optional) The 0-based index of the word. Default is 0, the first word on the page.</td>
</tr>
<tr>
<td>bStrip</td>
<td>(optional) Whether punctuation and whitespace should be removed from the word before returning. Default is <code>true</code>.</td>
</tr>
</tbody>
</table>

**Returns**

The nth word on the page.

**Example**

See Example 2 of `spell.checkWord` for an example.

getPageNthWordQuads

```
5.0
S
```

Gets the quads list for the nth word on the page. The quads can be used for constructing text markup annotations, Underline, StrikeOut, Highlight and Squiggly. See also `getPageNthWord`, `getPageNumWords`, and `selectPageNthWord`.

**NOTE:** (Security ⪯): This method throws an exception if the document security is set to prevent content extraction.

**Parameters**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) The 0-based index of the page. Default is 0, the first page in the document.</td>
</tr>
<tr>
<td>nWord</td>
<td>(optional) The 0-based index of the word. Default is 0, the first word on the page.</td>
</tr>
</tbody>
</table>
Returns

The **quads** list for the n\textsuperscript{th} word on the page.

Example

The following example underlines the fifth word on the second page of a document.

```javascript
var annot = this.addAnnot({
    page: 1,
    type: "Underline",
    quads: this.getPageNthWordQuads(1, 4),
    author: "A. C. Acrobat",
    contents: "Fifth word on second page"
});
```

See **spell.checkWord** for an additional example.

**getPageNumWords**

5.0

Gets the number of words on the page. See also **getPageNthWord**, **getPageNthWordQuads**, and **selectPageNthWord**.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) The 0-based index of the page. Default is 0, the first page in the document.</td>
</tr>
</tbody>
</table>

Returns

The number of words on the page.

Example

// count the number of words in a document
var cnt=0;
for (var p = 0; p < this.numPages; p++)
    cnt += getPageNumWords(p);
console.println("There are " + cnt + " words in this doc.");

See Example 2 of **spell.checkWord** for an additional example.

**getPageRotation**

5.0

 Gets the rotation of the specified page. See also **setPageRotations**.
getPageTransition

5.0

Gets the transition of the specified page. See also setPageTransitions.

Parameters

| nPage | (optional) The 0-based index of the page. Default is 0, the first page in the document. |

Returns

An array of three values: [nDuration, cTransition, nTransDuration].

- nDuration is the maximum amount of time the page is displayed before the viewer automatically turns to the next page. A duration of -1 indicates that there is no automatic page turning.
- cTransition is the name of the transition to apply to the page. See the application property transitions for a list of valid transitions.
- cTransDuration is the duration (in seconds) of the transition effect.

getPrintParams

6.0

Gets a printParams object that reflects the default print settings. See print, which now takes the printParams object as its parameter.

Parameters

None

Returns

A printParams Object.

Example

Get the printParams object of the default printer.
```javascript
var pp = this.getPrintParams();
pp.colorOverride = pp.colorOverrides.mono; // set some properties
this.print(pp); // print
```

**getSound**

5.0

Gets the `sound` object corresponding to the specified name. See also `sounds`, `importSound`, `deleteSound`, and the Sound Object.

**Parameters**

<table>
<thead>
<tr>
<th>cName</th>
<th>The name of the object to obtain.</th>
</tr>
</thead>
</table>

**Returns**

The Sound Object corresponding to the specified name.

**Example**

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

**getTemplate**

5.0

Gets the named template from the document. See also templates, `createTemplate`, `removeTemplate`, and the Template Object.

**Parameters**

<table>
<thead>
<tr>
<th>cName</th>
<th>The name of the template to retrieve.</th>
</tr>
</thead>
</table>

**Returns**

The Template Object or null if the named template does not exist in the document.

**Example**

```javascript
var t = this.getTemplate("myTemplate");
if (t != null) console.println( "myTemplate exists and is "+ eval( '(' + t.hidden + ') ? "hidden" : "visible" ') + ".");
else console.println( "myTemplate is not present!");
```
**getURL**

4.0 ćΔ ćS

Gets the specified URL over the internet using a GET. If the current document is being viewed inside the browser, or Acrobat Web Capture is not available, the method uses the Weblink plug-in to retrieve the requested URL. If running inside Acrobat, the method gets the URL of the current document either from the `baseURL`, from the URL of the first page (page 0) if the document was WebCaptured, or from the file system.

**NOTE:** This method roughly corresponds to the “open a web page” action.

**Parameters**

<table>
<thead>
<tr>
<th>cURL</th>
<th>A fully qualified URL or a relative URL. There can be a query string at the end of the URL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>bAppend</td>
<td>(optional) If true (the default), the resulting page or pages should be appended to the current document. This flag is considered to be false if the document is running inside the web browser, the Acrobat Web Capture plug-in is not available, or if the URL is of type &quot;file:///&quot;.</td>
</tr>
</tbody>
</table>

**NOTE:** (Security ćS): Beginning with Acrobat 6.0, if `bAppend` is true, the `getURL` method can only be executed during a console, menu or batch event.

**Returns**

Nothing

**Example**

```javascript
this.getURL("http://www.adobe.com/", false);
```

**gotoNamedDest**

Goes to a named destination within the PDF document. For details on named destinations and how to create them, see page 387 of the PDF Reference.

**Parameters**

| cName | The name of the destination within a document. |

**Returns**

Nothing

**Example**

The following example opens a document then goes to a named destination within that document.
// open new document
var myNovelDoc = app.openDoc("/c/fiction/myNovel.pdf");
// go to destination in this new doc
myNovelDoc.gotoNamedDest("chapter5");
// close old document
this.closeDoc();

**importAnFDF**

| 4.0 | D | F |

Imports the specified FDF file. See also **importAnXFDF** and **importTextData**.

**Parameters**

- **cPath** (optional) The device-independent pathname to the FDF file. See Section 3.10.1 of the PDF Reference for a description of the device-independent pathname format. It should look like the value of the /F key in an FDF exported with the **submitForm** method or with the **Advanced > Forms > Export Form Data** menu item. The pathname may be relative to the location of the current document. If this parameter is omitted, a dialog is shown to let the user select the file.

**Returns**

Nothing

**Example**

The following code, which is an action of a Page Open event, checks whether a certain function, **ProcResponse**, is already defined, if not, it installs a document level JavaScript, which resides in an FDF file.

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

Here, the pathname is a relative one. This technique may be useful for automatically installing document level JavaScripts for PDF files distilled from a PostScript file.

**importAnXFDF**

| 5.0 | D | F |

Imports the specified XFDF file containing XML form data. See also **importAnFDF** and **importTextData**. For a description of XFDF, see “Forms System Implementation Notes” in the Acrobat CD Documentation.
### Parameters

**cPath** (optional) The device-independent pathname to the XFDF file. See Section 3.10.1 of the PDF Reference for a description of the device-independent pathname format. The pathname may be relative to the location of the current document. If the parameter is omitted, a dialog is shown to let the user select the file.

### Returns

Nothing

## importDataObject

Imports an external file into the document and associates the specified name with the data object. Data objects can later be extracted or manipulated. See also Data Object, dataObjects, createDataObject, exportDataObject, getDataObject, importDataObject and removeDataObject.

**Note:** (Security): If the cDIPath parameter is specified, then this method can only be executed during batch, console or menu events, or through an external call (for example, OLE). See the Event Object for a discussion of Acrobat JavaScript events.

### Parameters

**cName** The name to associate with the data object.

**cDIPath** (optional) A device-independent path to a data file on the user’s hard drive. This path may be absolute or relative to the current document. If not specified, the user is prompted to locate a data file. See File Specification Strings in the PDF Reference Manual for the exact syntax of the path.

**cCryptFilter** (optional, version 6.0) The language independent name of a crypt filter to use when encrypting this data object. This crypt filter must have previously been added to the document’s list of crypt filters, using the Document Object addRecipientListCryptFilter method, otherwise an exception will be thrown. The predefined "Identity" crypt filter can be used if it is desired that this data object not be encrypted in a file that is otherwise encrypted by the Document Object encryptForRecipients method.

### Returns

true on success. An exception is thrown on failure.
Example

```javascript
function DumpDataObjectInfo(dataobj) {
    for (var i in dataobj)
        console.println(dataobj.name + "[" + i + "]=" + dataobj[i]);
}
// Prompt the user for a data file to embed.
this.importDataObject("MyData");
DumpDataObjectInfo(this.getDataObject("MyData"));
// Embed Foo.xml (found in parent director for this doc).
this.importDataObject("MyData2", ".\..\Foo.xml");
DumpDataObjectInfo(this.getDataObject("MyData2"));
```

**importIcon**

Imports an icon into the document and associates it with the specified name. See also icons, addIcon, getIcon, removeIcon, field methods buttonGetIcon, buttonImportIcon, buttonSetIcon, and the Icon Generic Object.

Beginning with version 6.0, Acrobat will first attempt to open `cDIPath` as a PDF. On failure, Acrobat will try to convert `cDIPath` to PDF from one of the known graphics formats (BMP, GIF, JPEG, PCX, PNG, TIFF) and then import the converted file as a button icon.

**NOTE:** (Security): If `cDIPath` is specified, this method can only be executed during batch, console or menu events. See the Event Object for a discussion of Acrobat JavaScript events.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cName</code></td>
<td>The name to associate with the icon.</td>
</tr>
<tr>
<td><code>cDIPath</code></td>
<td>(optional) A device-independent path to a PDF file on the user's hard drive. This path may be absolute or relative to the current document. <code>cDIPath</code> may only be specified in a batch environment or from the console. See Section 3.10.1, “File Specification Strings” in the PDF Reference for the exact syntax of the path. If not specified, the <code>nPage</code> parameter is ignored and the user is prompted to locate a PDF file and browse to a particular page.</td>
</tr>
<tr>
<td><code>nPage</code></td>
<td>(optional) The 0-based index of the page in the PDF file to import as an icon. Default is 0.</td>
</tr>
</tbody>
</table>
Returns

An integer code indicating whether it was successful or not:

0: No error
1: The user cancelled the dialog
-1: The selected file could not be opened
-2: The selected page was invalid

Example

This function is useful to populate a document with a series of named icons for later retrieval. For example, if a user of a document selects a particular state in a listbox, the author may want the picture of the state to appear next to the listbox. In prior versions of the application, this could be done using a number of fields that could be hidden and shown. This is difficult to author, however; instead, the appropriate script might be something like this:

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

This uses a single field to perform the same effect.

A simple user interface can be constructed to add named icons to a document. Assume the existence of two fields: a field called `IconName` which will contain the icon name and a field called `IconAdd` which will add the icon to the document. The mouse up script for `IconAdd` would be:

```javascript
var t = this.getField("IconName");
this.importIcon(t.value);
```

The same kind of script can be applied in a batch setting to populate a document with every selected icon file in a folder.

**importSound**

Imports a sound into the document and associates the specified name with the sound.

**NOTE:** (Security): If `cDIPath` is specified, this method can only be executed during batch, console, or menu events. See the **Event Object** for a discussion of Acrobat JavaScript events.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cName</code></td>
<td>The name to associate with the sound object.</td>
</tr>
</tbody>
</table>
cDIPath
(optional) A device-independent path to a sound file on the user's hard drive. This path may be absolute or relative to the current document. If not specified, the user is prompted to locate a sound file. See Section 3.10.1, "File Specification Strings", in the PDF Reference for the exact syntax of the path.

Returns
Nothing

Example
this.importSound("Moo");
this.getSound("Moo").play();
this.importSound("Moof", ".\moof.wav");
this.getSound("Moof").play();

See also sounds, getSound, deleteSound, and the Sound Object.

importTextData

Imports a row of data from a text file. Each row must be tab delimited. The entries in the first row of the text file are the column names of the tab delimited data. These names are also field names for text fields present in the PDF file. The data row numbers are 0-based; that is, the first row of data is row zero (this does not include the column name row). When a row of data is imported, each column datum becomes the field value of the field that corresponds to the column to which the data belongs.

Parameters

Parameters

<table>
<thead>
<tr>
<th>cPath</th>
<th>(optional) A relative device-independent path to the text file. If not specified, the user is prompted to locate the text data file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRow</td>
<td>(optional) The 0-based index of the row of the data to import, not counting the header row. If not specified, the user is prompted to select the row to import.</td>
</tr>
</tbody>
</table>

Returns
Nothing

Example 1

Suppose there are text fields named "First", "Middle" and "Last", and there is also a data file, the first row of which consists of the three strings, First, Middle and Last, separated by tabs. Suppose there are four additional rows of name data, again separated by tabs.

<table>
<thead>
<tr>
<th>First</th>
<th>Middle</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Al</td>
<td>Recount</td>
<td>Gore</td>
</tr>
</tbody>
</table>
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)

Example (continued)

The following code is a mouse up action for a button. Clicking on the button cycles through
the text file and populates the three fields "First", "Middle" and "Last" with the name data.

    if (typeof cnt == "undefined") cnt = 0;
    this.importTextData("/c/data/textdata.txt", cnt++ % 4)

The same functionality can be obtained using the ADBC Object and associated properties
and methods. The data file can be a spreadsheet or a database.

**importXFADData**

Imports the specified XFA file. See also **importAnXFDF** and **importTextData**.

**NOTE:** (SecurityŚ): This method is only allowed in batch, console, and menu events.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cPath</td>
<td>(optional) The device-independent pathname to the XFA file. The pathname may be relative to the location of the current document. If this parameter is omitted a dialog is shown to let the user select the file.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**insertPages**

Inserts pages from the source document into the current document. If a page range is not
specified, gets all pages in the source document. See also **deletePages** and
**replacePages**.

**NOTE:** (SecurityŚ): This method can only be executed during batch, console, or menu
events. See the Event Object for a discussion of Acrobat JavaScript events.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) The 0-based index of the page after which to insert the source document pages. Use -1 to insert pages before the first page of the document.</td>
</tr>
<tr>
<td>cPath</td>
<td>The device-independent pathname to the PDF file that will provide the inserted pages. See Section 3.10.1 of the PDF Reference for a description of the device-independent pathname format. The pathname may be relative to the location of the current document.</td>
</tr>
<tr>
<td>nStart</td>
<td>(optional) A 0-based index that defines the start of an inclusive range of pages in the source document to insert. If only nStart is specified then the range of pages is the single page specified by nStart.</td>
</tr>
<tr>
<td>nEnd</td>
<td>(optional) A 0-based index that defines the end of an inclusive range of pages in the source document to insert. If only nEnd is specified then the range of pages is 0 to nEnd.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

Insert a cover page to the current document.

```javascript
this.insertPages
  ({
    nPage: -1,
    cPath: "/c/temp/myCoverPage.pdf",
    nStart: 0
  });
```

mailDoc

Saves the current PDF document and mails it as an attachment to all recipients, with or without user interaction. See also `mailGetAddrs`, `mailMsg`, `mailForm` and `report.mail`.

**NOTE:** For Adobe 5.1 Reader and beyond, this method is commonly allowed, but document Save rights are required in case the document is changed.

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUI</td>
<td>(optional) If true (the default), the rest of the parameters are used in a compose-new-message window that is displayed to the user. If false, the cTo parameter is required and all others are optional.</td>
</tr>
<tr>
<td>cTo</td>
<td>(optional) The semicolon-delimited list of recipients for the message.</td>
</tr>
<tr>
<td>cCc</td>
<td>(optional) The semicolon-delimited list of CC recipients for the message.</td>
</tr>
<tr>
<td>cBcc</td>
<td>(optional) The semicolon-delimited list of BCC recipients for the message.</td>
</tr>
<tr>
<td>cSubject</td>
<td>(optional) The subject of the message. The length limit is 64k bytes.</td>
</tr>
<tr>
<td>cMsg</td>
<td>(optional) The content of the message. The length limit is 64k bytes.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

This pops up the compose-new-message window.

```javascript
this.mailDoc(true);
```

This sends out mail with the attached PDF file to fun1@fun.com and fun2@fun.com.

```javascript
this.mailDoc(false, "fun1@fun.com", "fun2@fun.com", ",
"This is the subject", "This is the body.");
```

mailForm

Exports the form data and mails the resulting FDF file as an attachment to all recipients, with or without user interaction. The method does not support signed signature fields. See also mailGetAddrs, mailMsg, mailDoc and report.mail.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUI</td>
<td>(optional) If true (the default), the rest of the parameters are used in a compose-new-message window that is displayed to the user. If false, the cTo parameter is required and all others are optional.</td>
</tr>
</tbody>
</table>
movePage

Moves a page within the document.

Parameters

- **nPage** (optional) The 0-based index of the page to move. Default is 0.
- **nAfter** (optional) The 0-based index of the page after which to move the specified page. Use -1 to move the page before the first page of the document. Default is the last page in the document.

Returns

Nothing

Example

Reverse the pages in the document.

```javascript
for (i = this.numPages - 1; i >= 0; i--) this.movePage(i);
```
newPage

6.0  D  S  X

Adds a new page to the active document in the Acrobat Viewer.

**Note:** (Security S): This method can only be executed during batch, console or menu events.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) The page after which to add the new page in a 1-based page numbering system. The default is the last page of the document. Use 0 to add a page before the first page. An invalid page range is truncated to the valid range of pages.</td>
</tr>
<tr>
<td>nWidth</td>
<td>(optional) The width of the page in points. The default value is 612.</td>
</tr>
<tr>
<td>nHeight</td>
<td>(optional) The height of the page in points. The default value is 792.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**Example**

Add a new page to match the page size of the doc.

```javascript
var Rect = this.getPageBox("Crop");
this.newPage(0, Rect[2], Rect[1]);
```

print

Prints all or a specific number of pages of the document.

Beginning with Acrobat 6.0, the method can print the document using the settings contained in a `printParams Object`, rather than through the other parameters. The permanent print settings are not altered.

**Notes:** (Security S, version 6.0) When printing to a file, the path must be a **Safe Path**. The `print` method will not overwrite an existing file.

On a Windows platform, the file name must include an extension of .ps or .prn (case insensitive). Additionally, the `print` method will not create a file directly in the root directory, the windows directory, or the windows system directory.

An **InvalidArgError** (see the **Error Objects**) exception will be thrown and `print` will fail if any of the above security restrictions are not met.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUI</td>
<td>(optional) If true (the default), will cause a UI to be presented to the user to obtain printing information and confirm the action.</td>
</tr>
<tr>
<td>nStart</td>
<td>(optional) A 0-based index that defines the start of an inclusive range of pages. If nStart and nEnd are not specified, prints all pages in the document. If only nStart is specified then the range of pages is the single page specified by nStart. If nStart and nEnd parameters are used, bUI must be false.</td>
</tr>
<tr>
<td>nEnd</td>
<td>(optional) A 0-based index that defines the end of an inclusive range of page. If nStart and nEnd are not specified, prints all pages in the document. If only nEnd is specified then the range of pages is 0 to nEnd. If nStart and nEnd parameters are used, bUI must be false.</td>
</tr>
<tr>
<td>bSilent</td>
<td>(optional) If true, suppresses the cancel dialog box while the document is printing. The default is false.</td>
</tr>
<tr>
<td>bShrinkToFit</td>
<td>(optional, version 5.0) If true, the page is shrunk (if necessary) to fit within the imageable area of the printed page. If false, it is not. The default is false.</td>
</tr>
<tr>
<td>bPrintAsImage</td>
<td>(optional, version 5.0) If true, print pages as an image. The default is false.</td>
</tr>
<tr>
<td>bReverse</td>
<td>(optional, version 5.0) If true, print from nEnd to nStart. The default is false.</td>
</tr>
<tr>
<td>bAnnotations</td>
<td>(optional, version 5.0) If true (the default), annotations are printed.</td>
</tr>
<tr>
<td>printParams</td>
<td>(optional, version 6.0) The printParams Object containing the settings to use for printing. If this parameter is passed, any other parameters are ignored.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example 1

This example prints current page the document is on.

this.print(false, this.pageNum, this.pageNum);
// print a file silently
this.print({bUI: false, bSilent: true, bShrinkToFit: true});

Example 2 (Version 6.0)

var pp = this.getPrintParams();
pp.interactive = pp.constants.interactionLevel.automatic;
pp.printerName = "hp officejet d series";
this.print(pp);

**NOTE:** When `printerName` is an empty string and `fileName` is nonempty the current document is saved to disk as a PostScript file.

**Example 3 (Version 6.0)**
Save the current document as a PostScript file.

```javascript
var pp = this.getPrintParams();
pp.fileName = "/c/temp/myDoc.ps";
pp.printerName = "";
this.print(pp);
```

### removeDataObject

`removeDataObject` deletes the data object corresponding to the specified name from the document. See also `dataObjects`, `createDataObject`, `exportDataObject`, `getDataObject`, `importDataObject`, `removeDataObject` and the Data Object.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cName</code></td>
<td>The name of the data object to remove.</td>
</tr>
</tbody>
</table>

#### Returns

Nothing

#### Example

```javascript
this.removeDataObject("MyData");
```

### removeField

`removeField` removes the specified field from the document. If the field appears on more than one page then all representations are removed.

**NOTE:** (version 6.0): Beginning with version 6.0, `doc.removeField` can now be used from within Adobe Reader for documents with “Advanced Form Features.”

```javascript
```
removelIcon

Removes the specified named icon from the document. See also icons, addIcon, getIcon, and importIcon, the field methods buttonGetIcon, buttonImportIcon, and buttonSetIcon, and the Icon Generic Object.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of the icon to remove.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

Remove all named icons from the document.

```javascript
for ( var i = 0; i < this.icons.length; i++)
    this.removelIcon(this.icons[i].name);
```

removeLinks

Removes all the links on the specified page within the specified coordinates, if the user has permission to remove links from the document. See also addLink, getLinks and the Link Object.
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>The 0-based index of the page from which to remove links.</td>
</tr>
<tr>
<td>oCoords</td>
<td>An array of four numbers in rotated user space, the coordinates of a rectangle listed in the following order: upper-left x, upper-left y, lower-right x and lower-right y.</td>
</tr>
</tbody>
</table>

### Returns

Nothing

### Example

Remove all links from the document.

```javascript
// remove all links from the document
for ( var p = 0; p < this.numPages; p++)
{
    var b = this.getPageBox("Crop", p);
    this.removeLinks(p, b);
}
```

Use `getLinks` to help count the number of links removed.

### removeTemplate

Removes the named template from the document. See also `templates`, `createTemplate`, `getTemplate`, and the Template Object.

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

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The name of the template to remove.</td>
</tr>
</tbody>
</table>

#### Returns

Nothing

### removeThumbnails

Deletes thumbnails for the specified pages in the document. See also `addThumbnails`. 
Parameters

nStart  (optional) A 0-based index that defines the start of an inclusive range of pages. If nStart and nEnd are not specified, operates on all pages in the document. If only nStart is specified, the range of pages is the single page specified by nStart.

nEnd  (optional) A 0-based index that defines the end of an inclusive range of pages. If nStart and nEnd are not specified, operates on all pages in the document. If only nEnd is specified, the range of pages is 0 to nEnd.

Returns

Nothing

removeWeblinks

Scans the specified pages looking for links with actions to go to a particular URL on the web and deletes them. See also addWeblinks.

**Note:** This method only removes weblinks authored in the application using the UI. Web links that are executed via JavaScript (for example, using `getURL`) are not removed.

Parameters

nStart  (optional) A 0-based index that defines the start of an inclusive range of pages. If nStart and nEnd are not specified, operates on all pages in the document. If only nStart is specified, the range of pages is the single page specified by nStart.

nEnd  (optional) A 0-based index that defines the end of an inclusive range of pages. If nStart and nEnd are not specified, operates on all pages in the document. If only nEnd is specified, the range of a pages is 0 to nEnd.

Returns

The number of web links removed from the document.

```javascript
var numWeblinks = this.removeWeblinks();
console.println("There were " + numWeblinks + " web links removed from the document.");
```
replacePages

Replaces pages in the current document with pages from the source document. See also deletePages, extractPages, and insertPages.

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) The 0-based index of the page at which to start replacement. Default is 0.</td>
</tr>
<tr>
<td>cPath</td>
<td>The device-independent pathname to the PDF file that will provide the replacement pages. See Section 3.10.1 of the PDF Reference for a description of the device-independent pathname format. The pathname may be relative to the location of the current document.</td>
</tr>
<tr>
<td>nStart</td>
<td>(optional) A 0-based index that defines the start of an inclusive range of pages in the source document to be used for replacement. If nStart and nEnd are not specified, gets all pages in the source document. If only nStart is specified, the range of pages is the single page specified by nStart.</td>
</tr>
<tr>
<td>nEnd</td>
<td>(optional) A 0-based index that defines the end of an inclusive range of pages in the source document to be used for replacement. If nStart and nEnd are not specified, gets all pages in the source document. If only nEnd is specified, the range of pages is 0 to nEnd.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

resetForm

Resets the field values within a document.

**NOTE:** Resetting a field causes it to take on its default value, which in the case of text fields is usually blank.
Parameters

**aFields** (optional) An array specifying the fields to reset. If not present or **null**, all fields in the form are reset. You can include non-terminal fields in the array.

Returns

Nothing

Example

Use this as a simple shortcut for having a whole subtree reset. For example, if you pass "name" as part of the fields array then **name.first**, **name.last**, and so on, are reset.

```javascript
var fields = new Array(2);
fields[0] = "P1.OrderForm.Description";
this.resetForm(fields);
```

**saveAs**

5.0

S S

Saves the file to the device-independent path specified by the required parameter, **cPath**. The file is not saved in linearized format. Beginning with Acrobat 6.0, the document can be converted to another file type (other than PDF) and saved as specified by the value of the **cConvID** parameter.

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

**NOTE:** (Adobe Reader **S**): This method is available in the Adobe Reader for documents that have "Save rights".
Parameters

**cPath**

The device-independent path in which to save the file.

**Note:** (Security ()): The parameter **cPath** is required to have a Safe Path and have an extension appropriate to the value of **cConvID**. See the Values of **cConvID** and Valid Extensions table below. This method will throw a **NotAllowedError** (see the **Error Objects**) exception if these security conditions are not met, and the method will fail.

**cConvID**

(optional, version 6.0) A conversion ID string that specifies the conversion file type. Currently supported values for **cConvID** are listed by the **app.fromPDFConverters**. If **cConvID** is not specified, then PDF is assumed.

Returns

Nothing

Values of **cConvID** and Valid Extensions

<table>
<thead>
<tr>
<th><strong>cConvID</strong></th>
<th><strong>Valid Extensions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>com.adobe.acrobat.eps</td>
<td>eps</td>
</tr>
<tr>
<td>com.adobe.acrobat.html-3-20</td>
<td>html, htm</td>
</tr>
<tr>
<td>com.adobe.acrobat.html-4-01-css-1-00</td>
<td>html, htm</td>
</tr>
<tr>
<td>com.adobe.acrobat.jpeg</td>
<td>jpeg, jpg, jpe</td>
</tr>
<tr>
<td>com.adobe.acrobat.jp2k</td>
<td>jpf, jpx, jp2, j2k, j2c, jpc</td>
</tr>
<tr>
<td>com.adobe.acrobat.doc</td>
<td>doc</td>
</tr>
<tr>
<td>com.adobe.acrobat.png</td>
<td>png</td>
</tr>
<tr>
<td>com.adobe.acrobat.ps</td>
<td>ps</td>
</tr>
<tr>
<td>com.adobe.acrobat.rtf</td>
<td>rft</td>
</tr>
<tr>
<td>com.adobe.acrobat.accesstext</td>
<td>txt</td>
</tr>
<tr>
<td>com.adobe.acrobat.plain-text</td>
<td>txt</td>
</tr>
<tr>
<td>com.adobe.acrobat.tif</td>
<td>tiff, tif</td>
</tr>
<tr>
<td>com.adobe.acrobat.xml-1-00</td>
<td>xml</td>
</tr>
<tr>
<td>com.adobe.acrobat.xdp</td>
<td>xdp</td>
</tr>
</tbody>
</table>
Example 1

The following code could appear as a batch sequence. Assume there is a PDF file already open containing form files that needs to be populated from a database and saved. Below is an outline of the script:

```javascript
// code lines to read from a database and populate the form with data
// now save file to a folder; use customerID from database record
// as name
var row = statement.getRow();

this.saveAs("/c/customer/invoices/"+row.customerID + ".pdf");
```

Example 2

You can use `newDoc` and `addField` to dynamically layout a form, then populate it from a database and save.

```javascript
var myDoc = app.newDoc();
// layout some dynamic form fields
// connect to database, populate with data, perhaps from a database

// save the doc and/or print it; print it silently this time
// to default printer
myDoc.saveAs("/c/customer/invoices/"+row.customerID + ".pdf");
myDoc.closeDoc(true); // close the doc, no notification
```

Example 3 (Version 6.0)

Save the current document in rich text format:

```javascript
this.saveAs("/c/myDocs/myDoc.rtf", "com.adobe.acrobat.rtf");
```

See `fromPDFConverters` for a listing of supported conversion ID strings.

**scroll**

Scrolls the specified point on the current page into middle of the current view. These coordinates must be defined in rotated user space. See the PDF Reference for details on the user space coordinate system.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nX</td>
<td>The x-coordinate for the point to scroll.</td>
</tr>
<tr>
<td>nY</td>
<td>The y-coordinate for the point to scroll.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing
selectPageNthWord

Changes the current page number and selects the specified word on the page. See also `getPageNthWord`, `getPageNthWordQuads` and `getPageNumWords`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) The 0-based index of the page to operate on. Default is 0, the first page in the document.</td>
</tr>
<tr>
<td>nWord</td>
<td>(optional) The 0-based index of the word to obtain. Default is 0, the first word on the page.</td>
</tr>
<tr>
<td>bScroll</td>
<td>(optional) Whether to scroll the selected word into the view if it is not already viewable. Default is <code>true</code>.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

setAction

Sets the JavaScript action of the document for a given trigger. See also `addScript`, `setPageAction`, `bookmark.setAction`, and `field.setAction`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cTrigger</td>
<td>The name of the trigger point to which to attach the action. Values are: <code>WillClose</code>, <code>WillSave</code>, <code>DidSave</code>, <code>WillPrint</code>, <code>DidPrint</code></td>
</tr>
<tr>
<td>cScript</td>
<td>The JavaScript expression to be executed when the trigger is activated.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**Example**

This example insert `WillSave` and `DidSave` actions. The code gets the filesize before saving and after saving, and compares the two.
// WillSave Script
var myWillSave = 'var filesizeBeforeSave = this.filesize;\r'
    + 'console.println("File size before saving is ");
    + 'filesizeBeforeSave);';

// DidSave Script
var myDidSave =  'var filesizeAfterSave = this.filesize;\r'
    + 'console.println("File size after saving is ");
    + 'filesizeAfterSave);\r'
    + 'var difference = filesizeAfterSave - filesizeBeforeSave;\r'
    + 'console.println("The difference is ");\r'
    + 'if ( difference < 0 )\r'
    + 'console.println("Reduced filesize!");\r'
    + 'else\r'
    + 'console.println("Increased filesize!");';

// Set Document Actions...
this.setAction("WillSave", myWillSave);
this.setAction("DidSave", myDidSave);

**setPageAction**

Sets the action of a page in a document for a given trigger. See also `setAction`, `addScript`, `bookmark.setAction`, and `field.setAction`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>The 0-based index of the page in the document to which an action is added.</td>
</tr>
<tr>
<td>cTrigger</td>
<td>The trigger for the action. Values are: Open Close</td>
</tr>
<tr>
<td>cScript</td>
<td>The JavaScript expression to be executed when the trigger is activated.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**Example**

```
this.setPageAction(0, "Open", "app.beep(0);");
```
### setPageBoxes

Sets a rectangle that encompasses the named box for the specified pages. See also `getPageBox`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cBox</td>
<td>(optional) The box type value, one of: Art, Bleed, Crop, Media, Trim. Note that the BBox box type is read-only and only supported in <code>getPageBox</code>. For definitions of these boxes see Section 8.6.1, “Page Boundaries” in the PDF Reference.</td>
</tr>
<tr>
<td>nStart</td>
<td>(optional) A 0-based index that defines the start of an inclusive range of pages in the document to be operated on. If <code>nStart</code> and <code>nEnd</code> are not specified, operates on all pages in the document. If only <code>nStart</code> is specified, the range of pages is the single page specified by <code>nStart</code>.</td>
</tr>
<tr>
<td>nEnd</td>
<td>(optional) A 0-based index that defines the end of an inclusive range of pages in the document to be operated on. If <code>nStart</code> and <code>nEnd</code> are not specified, operates on all pages in the document.</td>
</tr>
<tr>
<td>rBox</td>
<td>(optional) An array of four numbers in rotated user space to which to set the specified box. If not provided, the specified box is removed.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

### setPageLabels

Establishes the numbering scheme for the specified page and all pages following it until the next page with an attached label is encountered. See also `getPageLabel`.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) The 0-based index for the page to be labelled.</td>
</tr>
<tr>
<td>aLabel</td>
<td>(optional) An array of three required items <code>[cStyle, cPrefix, nStart]</code>.</td>
</tr>
<tr>
<td>● <strong>cStyle</strong> is the style of page numbering. Can be:</td>
<td></td>
</tr>
<tr>
<td>D: decimal numbering</td>
<td></td>
</tr>
<tr>
<td>R or r: roman numbering, upper or lower case</td>
<td></td>
</tr>
<tr>
<td>A or a: alphabetic numbering, upper or lower case</td>
<td></td>
</tr>
<tr>
<td>See the PDF Reference, Section 7.3.1, for the exact definitions of these styles.</td>
<td></td>
</tr>
<tr>
<td>● <strong>cPrefix</strong> is a string to prefix the numeric portion of the page label.</td>
<td></td>
</tr>
<tr>
<td>● <strong>nStart</strong> is the ordinal with which to start numbering the pages.</td>
<td></td>
</tr>
<tr>
<td>If not supplied, any page numbering is removed for the specified page and any others up to the next specified label.</td>
<td></td>
</tr>
<tr>
<td>The value of <strong>aLabel</strong> cannot be <strong>null</strong>.</td>
<td></td>
</tr>
</tbody>
</table>

Returns

Nothing

Example 1

10 pages in the document, label the first 3 with small roman numerals, the next 5 with numbers (starting at 1) and the last 2 with an "Appendix- prefix" and alphabatics.

```javascript
this.setPageLabels(0, ["r", ",", 1]);
this.setPageLabels(3, ["D", ",", 1]);
this.setPageLabels(8, ["A", "Appendix-", 1]);
var s = this.getPageLabel(0);
for (var i = 1; i < this.numPages; i++)
  s += ", " + this.getPageLabel(i);
console.println(s);
```

The example will produce the following output on the console:

```
i, ii, iii, 1, 2, 3, 4, 5, Appendix-A, Appendix-B
```

Example 2

Remove all page labels from a document.

```javascript
for (var i = 0; i < this.numPages; i++) {
  if (i + 1 != this.getPageLabel(i)) {
    // Page label does not match ordinal page number.
    this setPageLabels(i);
  }
}
```
**setPageRotations**

Rotates the specified pages in the current document. See also `getPageRotation`.

**Parameters**

- **nStart** (optional) A 0-based index that defines the start of an inclusive range of pages in the document to be operated on. If `nStart` and `nEnd` are not specified, operates on all pages in the document. If only `nStart` is specified, the range of pages is the single page specified by `nStart`.

- **nEnd** (optional) A 0-based index that defines the end of an inclusive range of pages in the document to be operated on. If `nStart` and `nEnd` are not specified, operates on all pages in the document. If only `nEnd` is specified, the range of pages is 0 to `nEnd`.

- **nRotate** (optional) The amount of rotation that should be applied to the target pages. Can be 0, 90, 180, or 270. Default is 0.

**Returns**

Nothing

**Example**

Rotate pages 0 through 10 of the current document.

```javascript
this.setPageRotations(0, 10, 90);
```

**setPageTabOrder**

Sets the tab order of the form fields on a page. The tab order can be set by row, by column, or by structure.

If a PDF 1.4 document is viewed in Acrobat 6.0, tabbing between fields is in the same order as it is in Acrobat 5.0. Similarly, if a PDF 1.5 document is opened in Acrobat 5.0, the tabbing order for fields is the same as it is in Acrobat 6.0.

**Parameters**

- **nPage** The 0-based index of the page number on which the tabbing order is to be set.
cOrder | The order to be used. Values are:
---|---
rows
columns
structure

**Returns**

Nothing

**Example**

Set the page tab order for all pages to *rows*.

```javascript
for (var i = 0; i < this.numPages; i++)
  this setPageTabOrder(i, "rows");
```

**setPageTransitions**

Sets the page transition for a specific range of pages. See also `getPageTransition`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nStart</td>
<td>(optional) A 0-based index that defines the start of an inclusive range of pages in the document to be operated on. If <code>nStart</code> and <code>nEnd</code> are not specified, operates on all pages in the document. If only <code>nStart</code> is specified, the range of pages is the single page specified by <code>nStart</code>.</td>
</tr>
<tr>
<td>nEnd</td>
<td>(optional) A 0-based index that defines the end of an inclusive range of pages in the document to be operated on. If <code>nStart</code> and <code>nEnd</code> are not specified, operates on all pages in the document. If only <code>nEnd</code> is specified, the range of pages is 0 to <code>nEnd</code>.</td>
</tr>
</tbody>
</table>
| aTrans | (optional) The page transition array consists of three values: `[nDuration, cTransition, nTransDuration]`.  
  - `nDuration` is the maximum amount of time the page is displayed before the viewer automatically turns to the next page. Set to -1 to turn off automatic page turning.  
  - `cTransition` is the name of the transition to apply to the page. See `fullScreen.transitions` for a list of valid transitions.  
  - `nTransDuration` is the duration (in seconds) of the transition effect.  
  If `aTrans` is not present, any page transitions for the pages are removed. |
Returns

Nothing

Example

Put document into fullscreen mode, and apply some transitions.

```javascript
this.setPageTransitions({ aTrans: [-1, "Random", 1] })
app.fullscreen = true;
```

**spawnPageFromTemplate**

Spawns a page in the document using the given template, as returned by `getNthTemplate`.

See `templates`, `createTemplate`, and `template.spawn`, which supersede this method in later versions.

**NOTE:** The template feature does not work in Adobe Reader.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cTemplate</td>
<td>The template name.</td>
</tr>
<tr>
<td>nPage</td>
<td>(optional) The 0-based page number before which or into which the template is spawned, depending on the value of <code>bOverlay</code>. If <code>nPage</code> is omitted, a new page is created at the end of the document.</td>
</tr>
<tr>
<td>bRename</td>
<td>(optional) Whether fields should be renamed. The default is <code>true</code>.</td>
</tr>
<tr>
<td>bOverlay</td>
<td>(optional, version 4.0) If <code>false</code>, the template is inserted before the page specified by <code>nPage</code>. When <code>true</code> (the default) it is overlaid on top of that page.</td>
</tr>
<tr>
<td>oXObject</td>
<td>(optional, version 6.0) The value of this parameter is the return value of an earlier call to <code>spawnPageFromTemplate</code>.</td>
</tr>
</tbody>
</table>

**Returns**

Prior to Acrobat 6.0, this method returned nothing. Now, this method returns an object representing the page contents of the page spawned. This return object can then be used as the value of the optional parameter `oXObject` for subsequent calls to `spawnPageFromTemplate`.

**NOTE:** Repeatedly spawning the same page can cause a large inflation in the file size. To avoid this file size inflation problem, `spawnPageFromTemplate` now returns an object that represents the page contents of the spawned page. This return value can
be used as the value of the `oXObject` parameter in subsequent calls to the `spawnPageFromTemplate` method to spawn the same page.

Example 1

```javascript
var n = this.numTemplates;
var cTempl;
for (i = 0; i < n; i++) {
    cTempl = this.getNthTemplate(i);
    this.spawnPageFromTemplate(cTempl);
}
```

Example 2 (version 6.0)

The following example spawns the same template 31 times using the `oXObject` parameter and return value. Using this technique avoids overly inflating the file size.

```javascript
var t = this.getNthTemplate(0)
var XO = this.spawnPageFromTemplate(t, this.numPages, false, false);
for (var i=0; i < 30; i++)
    this.spawnPageFromTemplate(t,this.numPages, false, false, XO);
```

submitForm

Submits the form to a specified URL. To call this method, you must be running inside a web browser or have the Acrobat Web Capture plug-in installed (unless the URL uses the "mailto" scheme, in which case it will be honored even if not running inside a web browser, as long as the SendMail plug-in is present). Beginning with Adobe Reader 6.0, you need not be inside a web browser to call this method.

**NOTE:** (Version 6.0), Depending on the parameters used, there are restrictions on the use of `submitForm`. See the notes embedded in the description of the parameters.

The `https` protocol is supported for secure connections.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cURL</code></td>
<td>The URL to submit to. This string must end in <code>#FDF</code> if the result from the submission is FDF or XFDF (that is, the value of <code>cSubmitAs</code> is &quot;FDF&quot; or &quot;XFDF&quot;) and the document is being viewed inside a browser window.</td>
</tr>
<tr>
<td><code>bFDF</code></td>
<td>(optional) Whether to submit as FDF or HTML. If <code>true</code>, the default, submits the form data as FDF. If <code>false</code>, submits it as URL-encoded HTML. This option has been deprecated, use <code>cSubmitAs</code> instead.</td>
</tr>
</tbody>
</table>
**bEmpty**

(optional) When `true`, submit all fields, including those that have no value. When `false` (the default), exclude fields that currently have no value.

**NOTE:** If data is submitted as XDP, XML or XFD (see the **cSubmitAs** parameter, below), the **bEmpty** parameter is ignored. All fields are submitted, even fields that are empty. See **aFields** below.

**aFields**

(optional) An array of field names to submit or a string containing a single field name.

- If supplied, only the fields indicated are submitted, except those excluded by **bEmpty**.
- If omitted or `null`, all fields are submitted, except those excluded by **bEmpty**.
- If an empty array, no fields are submitted. A submitted FDF might still contain data if **bAnnotations** is `true`.

You can specify non-terminal field names to export an entire subtree of fields.

**NOTE:** If data is submitted as XDP, XML or XFD (see the **cSubmitAs** parameter, below), the **aFields** parameter is ignored. All fields are submitted, even fields that are empty. See **bEmpty** above.

**bGet**

(optional, version 4.0) When `true`, submit using the HTTP GET method. When `false` (the default), use a POST. GET is only allowed if using Acrobat Web Capture to submit (the browser interface only supports POST) and only if the data is sent as HTML (that is, **cSubmitAs** is `HTML`).

**bAnnotations**

(optional, version 5.0) When `true`, annotations are included in the submitted FDF or XML. The default is `false`. Only applicable if **cSubmitAs** is `FDF` or `XFDF`.

**bXML**

(●) (optional, version 5.0) If `true`, submit as XML. The default is `false`.

This option has been deprecated, use **cSubmitAs** instead.

**bIncrChanges**

(optional, version 5.0) When `true`, include the incremental changes to the PDF in the submitted FDF. The default is `false`. Only applicable if **cSubmitAs** is `FDF`. Not available in the Adobe Reader.

**bPDF**

(●) (optional, version 5.0) If `true`, submit the complete PDF document. The default is `false`. When `true`, all other parameters except **cURL** are ignored. Not available in the Adobe Reader.

This option has been deprecated, use **cSubmitAs** instead.
bCanonical (optional, version 5.0) When true, convert any dates being submitted to standard format (that is, D:YYYYMMDDHHmmSSOH'mm'; see the PDF Reference for details). The default is false.

bExclNonUserAnnots (optional, version 5.0) A boolean that indicates, if true, to exclude any annotations that are not owned by the current user. The default is false.

bExclFKey (optional, version 5.0) When true, exclude the "F" key. The default is false.

cPassword (optional, Version 5.0) The password to use to generate the encryption key, if the FDF needs to be encrypted before getting submitted.
Pass the value true (no quotes), to use the password that the user has previously entered (within this Acrobat session) for submitting or receiving an encrypted FDF. If no password has been entered, prompts the user to enter a password.
Regardless of whether the password is passed in or requested from the user, this new password is remembered within this Acrobat session for future outgoing or incoming encrypted FDFs.
Only applicable if cSubmitAs is FDF.

bEmbedForm (optional, version 6.0) When true, the call embeds the entire form from which the data is being submitted in the FDF.
Only applicable if cSubmitAs is FDF.

oJavaScript (optional, version 6.0) Can be used to include Before, After, and Doc JavaScripts in a submitted FDF. If present, the value is converted directly to an analogous CosObj and used as the /JavaScript attribute in the FDF. For example:
oJavaScript:
{
Before: 'app.alert("before!")',
After: 'app.alert("after")',
Doc: [ "MyDocScript1", "myFunc1()", "MyDocScript2", "myFunc2()" ]
}
Only applicable if cSubmitAs is FDF.
### cSubmitAs

(optional, version 6.0) The format for submission. Values are:
- FDF *(default)*
- XFDF
- HTML
- XDP
- XML
- XFD
- PDF

**PDF** means submit the complete PDF document; in this case, all other parameters except **cURL** are ignored.

**NOTE:** Save rights required (サー): The **PDF** choice is not available in Adobe Reader, unless the document has save rights.

This parameter supercedes the individual format parameters; however, they are considered in the following priority order, from high to low: **cSubmitAs**, **bPDF**, **bXML**, **bFDF**.

### bInclNMKey

(optional, version 6.0) When **true**, include the "NM" key of any annotations. The default is **false**.

### aPackets

(optional, version 6.0) An array of strings, specifying which packets to include in an XDP submission. Possible strings are:
- template
dataSets
stylesheet
xdf
sourceSet
pdf
cfg

**pdf** means that the PDF should be embedded; if **pdf** is not included here, only a link to the PDF is included in the XDP.

**xdf** means to include annotations in the XDP (since that packet uses XFDF format).

***** means that all packets should be included in the XDP.

The default is: ['datasets', 'xdf'].

This parameter is only applicable if **cSubmitAs** is **XDP**.

**NOTE:** Save rights required (サー): When submitting a document as **XDP** from the Adobe Reader with **aPackets** set to **pdf** (or *****, which implies **pdf**), the document must have document save rights.
cCharset (optional, version 6.0) The encoding for the values submitted. String values are:
   utf-8
   utf-16
   Shift-JIS
   BigFive
   GBK
   UHC

If not passed, the current Acrobat behavior applies. For XML-based formats, utf-8 is used. For other formats, Acrobat tries to find the best host encoding for the values being submitted. XFDF submission ignores this value and always uses utf-8.

Returns
   Nothing

Example 1
   Submit the form to the server.

   this.submitForm("http://myserver/cgi-bin/myscript.cgi#FDF");

Example 2
   var aSubmitFields = new Array( "name", "id", "score" );
   this.submitForm({
      cURL: "http://myserver/cgi-bin/myscript.cgi#FDF",
      aFields: aSubmitFields,
      cSubmitAs: "FDF" // the default, not needed here
   });

Example 3
   This example illustrates a shortcut to submitting a whole subtree. Passing "name" as part of the field parameter, submits "name.title","name.first","name.middle" and "name.last".

   this.submitForm("http://myserver/cgi-bin/myscript.cgi#FDF",
   true, false, "name");

Example 4
   this.submitForm({
      cURL: "http://myserver/cgi-bin/myscript.cgi#FDF",
      cSubmitAs: "XFDF"
   });

syncAnnotScan

Guarantees that all annotations will be scanned by the time this method returns.
In order to show or process annotations for the entire document, all annotations must have been detected. Normally, a background task runs that examines every page and looks for annotations during idle time, as this scan is a time consuming task. Much of the annotation behavior works gracefully even when the full list of annotations is not yet acquired by background scanning.

In general, you should call this method if you want the entire list of annotations.

See also `getAnnots`.

**Parameters**

None

**Returns**

Nothing

**Example**

The second line of code will not be executed until `syncAnnotScan` returns and this will not occur until the annot scan of the document is completed.

```javascript
this.syncAnnotScan();
annots = this.getAnnots({nSortBy:ANSB_Author});
// now, do something with the annotations.
```

## Error Objects

Error objects are dynamically created whenever an exception is thrown from methods or properties implemented in Acrobat JavaScript. Several sub-classes of the `Error` object can be thrown by core JavaScript (`EvalError`, `RangeError`, `SyntaxError`, `TypeError`, `ReferenceError`, `URLError`). They all have the `Error` object as prototype. Acrobat JavaScript can throw some of these exceptions, or implement subclasses of the `Error` object at its convenience. If your scripts are using the mechanism of `try/catch` error handling, the object thrown should be one of the types listed in the following table.

<table>
<thead>
<tr>
<th>Error Object</th>
<th>Brief Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>RangeError</code></td>
<td>Argument value is out of valid range</td>
</tr>
<tr>
<td><code>TypeError</code></td>
<td>Wrong type of argument value</td>
</tr>
<tr>
<td><code>ReferenceError</code></td>
<td>Reading a variable that does not exist</td>
</tr>
<tr>
<td><code>MissingArgError</code></td>
<td>Missing required argument</td>
</tr>
<tr>
<td><code>NumberOfArgsError</code></td>
<td>Invalid number of arguments to a method</td>
</tr>
<tr>
<td><code>InvalidSetError</code></td>
<td>A property set is not valid or possible</td>
</tr>
<tr>
<td><code>InvalidGetError</code></td>
<td>A property get is not valid or possible</td>
</tr>
</tbody>
</table>
Error Properties

**Error Object** | **Brief Description**
--- | ---
`OutOfMemoryError` | Out of memory condition occurred
`NoSuchMethodError` | Functionality not supported in this configuration (for example: Reader)
`NoSuchHFTError` | HFT is not available (a plug-in may be missing)
`NotAllowedError` | Method or property is not allowed for security reasons
`GeneralError` | Unspecified error cause
`RaiseError` | Acrobat internal error
`DeadObjectError` | Object is dead
`HelpError` | User requested for help with a method

Error object types implemented by Acrobat JavaScript inherit properties and methods from the core Error object. Some Acrobat Javascript objects may implement their own specific types of exception. A description of the Error subclass (with added methods and properties, if any) should be provided in the documentation for the particular object.

**Example**

Print all properties of the Error object to the console.

```javascript
try {
    app.alert(); // one argument is required for alert
} catch(e) {
    for (var i in e)
        console.println( i + ": " + e[i])
}
```

**Error Properties**

**fileName**

6.0

The name of the script which caused the exception to be thrown.

*Type: String*  
*Access: R.*
**lineNumber**

The offending line number from where an exception was thrown in the JavaScript code.

*Type: Integer Access: R.*

**message**

The error message providing details about the exception.

*Type: String Access: R.*

**name**

The name of the **Error** object subclass, indicating the type of the **Error** object instance.

*Type: String Access: R/W.*

---

**Error Methods**

**toString**

Gets the error message providing details about the exception.

**Parameters**

None

**Returns**

The error message string. (See **message**.)

---

**Event Object**

All JavaScripts are executed as the result of a particular event. Each event has a type and a name. The events detailed here are listed as type/name pairs.
For each of these events, Acrobat JavaScript creates an event object. During the occurrence of each event, you can access this event object to get, and possibly manipulate, information about the current state of the event.

It is important for JavaScript writers to know when these events occur and in what order they are processed. Some methods or properties can only be accessed during certain events; therefore, a knowledge of these events will prove useful.

**Event Type/Name Combinations**

**App/Init**

When the Viewer is started, the Application Initialization Event occurs. Script files, called Folder Level JavaScripts, are read in from the application and user JavaScript folders. They load in the following order: Config.js, glob.js, all other files, then any user files.

This event defines the name and type properties for the event object.

This event does not listen to the rc return code.

**Batch/Exec**

A batch event occurs during the processing of each document of a batch sequence. JavaScripts that authored as part of a batch sequence can access the event object upon execution.

This event defines the name, target, and type properties for the event object. The target in this event is the document object.

This event listens to the rc return code. If the return code is set to false, the batch sequence is stopped.

**Bookmark/Mouse Up**

This event occurs whenever a user clicks on a bookmark that executes a JavaScript.

This event defines the name, target, and type properties for the event object. The target in this event is the bookmark object that was clicked.

This event does not listen to the rc return code.

**Console/Exec**

A console event occurs whenever a user evaluates a JavaScript in the console.

This event defines the name, and type properties for the event object.

This event does not listen to the rc return code.
Doc/DidPrint

This event is triggered after a document has printed.
This event defines the name, target, and type properties for the event object. The target in this event is the document object.
This event does not listen to the rc return code.

Doc/DidSave

This event is triggered after a document has been saved.
This event defines the name, target, and type properties for the event object. The target in this event is the document object.
This event does not listen to the rc return code.

Doc/Open

This event is triggered whenever a document is opened. When a document is opened, the document level script functions are scanned and any exposed scripts are executed.
This event defines the name, target, targetName, and type properties for the event object. The target in this event is the document object.
This event does not listen to the rc return code.

Doc/WillClose

This event is triggered before a document is closed.
This event defines the name, target, and type properties for the event object. The target in this event is the document object.
This event does not listen to the rc return code.

Doc/WillPrint

This event is triggered before a document is printed.
This event defines the name, target, and type properties for the event object. The target in this event is the document object.
This event does not listen to the rc return code.
Doc/WillSave

This event is triggered before a document is saved.

This event defines the name, target, and type properties for the event object. The target in this event is the document object.

This event does not listen to the rc return code.

External/Exec

This event is the result of an external access, for example, through OLE, AppleScript, or loading an FDF.

This event defines the name and type properties for the event object.

This event does not listen to the rc return code.

Field/Blur

The blur event occurs after all other events just as the field loses focus. This event is generated regardless of whether or not a mouse click is used to deactivate the field (for example, tab key).

This event defines the modifier, name, shift, target, targetName, type, and value properties for the event object. The target in this event is the field whose validation script is being executed.

This event does not listen to the rc return code.

Field/Calculate

This event is defined when a change in a form requires that all fields that have a calculation script attached to them be executed. All fields that depend on the value of the changed field will now be re-calculated. These fields may in turn generate additional Field/Validate, Field/Blur, and Field/Focus events.

Calculated fields may have dependencies on other calculated fields whose values must be determined beforehand. The calculation order array contains an ordered list of all the fields in a document that have a calculation script attached. When a full calculation is needed, each of the fields in the array is calculated in turn starting with the zeroth index of the array and continuing in sequence to the end of the array.

To change the calculation order of fields, use the Advanced>Forms>Set Field Calculation Order... menu item in Adobe Acrobat.

This event defines the name, source, target, targetName, type, and value properties for the event object. The target in this event is the field whose calculation script is being executed.
This event does listen to the rc return code. If the return code is set to false, the field’s value is not changed. If true, the field takes on the value found in the value.

**Field/Focus**

The focus event occurs after the mouse down but before the mouse up after the field gains the focus. This routine is called whether or not a mouse click is used to activate the field (for example, tab key) and is the best place to perform processing that must be done before the user can interact with the field.

This event defines the modifier, name, shift, target, targetName, type, and value properties for the event object. The target in this event is the field whose validation script is being executed.

This event does not listen to the rc return code.

**Field/Format**

Once all dependent calculations have been performed the format event is triggered. This event allows the attached JavaScript to change the way that the data value appears to a user (also known as its presentation or appearance). For example, if a data value is a number and the context in which it should be displayed is currency, the formatting script can add a dollar sign ($) to the front of the value and limit it to two decimal places past the decimal point.

This event defines the commitKey, name, target, targetName, type, value, and willCommit properties for the event object. The target in this event is the field whose format script is being executed.

This event does not listen to the rc return code. However, the resulting value is used as the fields formatted appearance.

**Field/Keystroke**

The keystroke event occurs whenever a user types a keystroke into a textbox or combobox (this includes cut and paste operations), or selects an item in a combobox dropdown or listbox field. A keystroke script may want to limit the type of keys allowed. For example, a numeric field might only allow numeric characters.

The user interface for Acrobat allows the author to specify a Selection Change script for listboxes. The script is triggered every time an item is selected. This is implemented as the keystroke event where the keystroke value is equivalent to the user selection. This behavior is also implemented for the combobox—the "keystroke" could be thought to be a paste into the text field of the value selected from the drop down list.

There is a final call to the keystroke script before the validate event is triggered. This call sets the willCommit to true for the event. With keystroke processing, it is sometimes useful to make a final check on the field value before it is committed (pre-commit). This allows the script writer to gracefully handle particularly complex formats that can only be partially checked on a keystroke by keystroke basis.
The **keystroke** event of text fields is called in situations other than when the user is entering text with the keyboard or committing the field value. It is also called to validate the default value of a field when set through the UI or by JavaScript, and to validate entries provided by autofill. In these situations not all properties of the event are defined. Specifically `event.target` will be `undefined` when validating default values and `event.richChange` and `event.richValue` will be `undefined` when validating autofill entries.

This event defines the `commitKey`, `change`, `changeEx`, `keyDown`, `modifier`, `name`, `selEnd`, `selStart`, `shift`, `target` (except when validating default values), `targetName`, `type`, `value`, and `willCommit` properties for the event object. The target in this event is the field whose keystroke script is being executed.

This event does listen to the `rc` return code. If set to `false`, the keystroke is ignored. The resulting `change` is used as the keystroke if the script desires to replace the keystroke code. The resultant `selEnd` and `selStart` properties can change the current text selection in the field.

**Field/Mouse Down**

The **mouse down** event is triggered when a user starts to click on a form field and the mouse button is still down. It is advised that you perform very little processing (that is, play a short sound) during this event. A mouse down event will not occur unless a **mouse enter** event has already occurred.

This event defines the `modifier`, `name`, `shift`, `target`, `targetName`, and `type` properties for the event object. The target in this event is the field whose validation script is being executed.

This event does not listen to the `rc` return code.

**Field/Mouse Enter**

The **mouse enter** event is triggered when a user moves the mouse pointer inside the rectangle of a field. This is the typical place to open a text field to display help text, and so on.

This event defines the `modifier`, `name`, `shift`, `target`, `targetName`, and `type` properties for the event object. The target in this event is the field whose validation script is being executed.

This event does not listen to the `rc` return code.

**Field/Mouse Exit**

The **mouse exit** event is the opposite of the **mouse enter** event and occurs when a user moves the mouse pointer outside of the rectangle of a field. A **mouse exit** event will not occur unless a **mouse enter** event has already occurred.

This event defines the `modifier`, `name`, `shift`, `target`, `targetName`, and `type` properties for the event object. The target in this event is the field whose validation script is being executed.

This event does not listen to the `rc` return code.
Field/Mouse Up

The `mouse up` event is triggered when the user clicks on a form field and releases the mouse button. This is the typical place to attach routines such as the submit action of a form. A `mouse up` event will not occur unless a `mouse down` event has already occurred.

This event defines the `modifier, name, shift, target, targetName, and type` properties for the event object. The target in this event is the field whose validation script is being executed.

This event does not listen to the `rc` return code.

Field/Validate

Regardless of the field type, user interaction with a field may produce a new value for that field. After the user has either clicked outside a field, tabbed to another field, or pressed the enter key, the user is said to have committed the new data value.

The `validate` event is the first event generated for a field after the value has been committed so that a JavaScript can verify that the value entered was correct. If the validate event is successful, the next event triggered is the `calculate` event.

This event defines the `change, changeEx, keyDown, modifier, name, shift, target, targetName, type, and value` properties for the event object. The target in this event is the field whose validation script is being executed.

This event does listen to the `rc` return code. If the return code is set to `false`, the field value is considered to be invalid and the value of the field is unchanged.

Link/Mouse Up

This event is triggered when a link containing a JavaScript action is activated by the user.

This event defines the `name, target, and type` properties for the event object. The target in this event is the document object.

This event does not listen to the `rc` return code.

Menu/Exec

A menu event occurs whenever JavaScript that has been attached to a menu item is executed. In Acrobat 5.0, the user can add a menu item and associate JavaScript actions with it. For example,

```javascript
app.addMenuItem({ cName: "Hello", cParent: "File", cExec: "app.alert('Hello',3);", nPos: 0});
```

The script `app.alert('Hello',3);` will execute during a `menu` event. There are two ways for this to occur:

1. Through the user interface, the user can click on that menu item and the script will execute; and
2. Programmatically, when `app.execMenuItem("Hello")` is executed (perhaps, during a mouse up event of a button field), the script will execute.

This event defines the `name`, `target`, `targetName`, and `type` properties for the event object. The target in this event is the currently active document, if one is open.

This event listens to the `rc` return code in the case of the enable and marked proc for menu items. A return code of `false` will disable or unmark a menu item. A return code of `true` will Event Processing.

**Page/Open**

This event happens whenever a new page is viewed by the user and after page drawing for the page has occurred.

This event defines the `name`, `target`, and `type` properties for the event object. The target in this event is the document object.

This event does not listen to the `rc` return code.

**Page/Close**

This event happens whenever the page being viewed is no longer the current page; that is, the user switched to a new page or closed the document.

This event defines the `name`, `target`, and `type` properties for the event object. The target in this event is the document object.

This event does not listen to the `rc` return code.

**Document Event Processing**

When a document is opened, the `Doc/Open` event occurs: functions are scanned, and any exposed scripts are executed. Next, if the `NeedAppearances` key in the PDF file is set to `true` in the `AcroForm` dictionary, the formatting scripts of all form fields in the document are executed. (See Section 3.6.1 and 7.6.1 of the PDF Reference.) Finally, the `Page/Close` event occurs.

**Note:** For users who create PDF files containing form fields with the `NeedAppearances` key set to true, be sure to do a “Save As” before posting such files on the Web. Performing a “Save As” on a file generates the form appearances, which are saved with the file. This increases the performance of Reader when it loads the file within a Web browser.
Form Event Processing

The order in which the form events occur is illustrated in the state diagram below. This illustrates certain dependencies that are worth noting, for example, the Mouse Up event cannot occur if the Focus event did not occur.

*Selection change for list box only.

Event Properties

change

Specifies the change in value that the user has just typed. This is replaceable such that if the JavaScript wishes to substitute certain characters, it may. The change may take the form of an individual keystroke or a string of characters (for example if a paste into the field is performed).

Type: String

Access: R/W.

Example

Change all keystrokes to upper case.

```
// Custom Keystroke for text field
event.change = event.change.toUpperCase();
```
changeEx

Contains the export value of the change and is available only during a Field/Keystroke event for listbox and combobox.

For the listbox, the keystroke script, if any, is entered under the Selection Change tab in the properties dialog.

For the combobox, changeEx is only available if the pop-up part of the combo is used, that is, a selection (with the mouse or the keyboard) is being made from the pop-up. If the combo is editable and the user types in an entry, the Field/Keystroke event behaves as for a text field (that is, there are no changeEx or keyDown event properties).

Beginning with Acrobat 6.0, event.changeEx is defined for text fields. When event.fieldFull is true, changeEx is set to the entire text string the user attempted to enter and event.change is the text string cropped to what fits within the field. Use event.richChangeEx (and event.richChange) to handle rich text fields.

Type: various Access: R.

Example 1

This example illustrates the differences between event.value, event.changeEx and event.change. The script below is document level JavaScript used to process a custom keystroke of an editable combobox. The same script can basically be used to process a listbox as well. Try this example with field.commitOnSelChange first set to false, then set to true to compare responses.

```javascript
// convenience function for printing to console
var cp = function(msg) { console.println(msg) }

// document level script to process keystrokes of editable combo box
function customKey_Combo() {
    if (event.willCommit) {
        cp("Committed");
        // This is the face value of the committed item
        cp("event.value = " + event.value);
        // These next two values are not relevant to a committed field.
        // Each value is the empty string
        cp("event.change = " + event.change);
        cp("event.changeEx = " + event.changeEx);
    }
    else {
        cp("Not Committed");
        /* event.value is the export value of the current item, the one
           before the change. */
        cp("event.value = " + event.value);
        /* This is the change. It could be the face value of a listed
           item, or a keystroke (if typing/pasting into an edit box is
           permitted). */
    }
}
```
cp( "event.change = " + event.change );
/* This is the export value of the change. If the export value wasn't given in the UI, then this is the same as event.value. */
cp( "event.changeEx = " + event.changeEx );
/* If event.changeEx is the empty string, then a menu item has not been chosen. User is typing or pasting into the editable field. */
if ( event.changeEx == "" ) {
    /* If the length of event.change is one, then user has probably pressed a single key to input into the edit box. Or, user could have pasted a single keystroke as well. */
    switch ( event.change.length ) {
        case 0:
            cp("User has backspaced or deleted one or more " + "characters");
            break;
        case 1:
            cp("User enters data into editable field: " + event.change);
            // process keystroke, say, change to upper case.
            event.change = event.change.toUpperCase();
            cp("User has entered a single char, modified to " + event.change);
            break;
        default:
            cp("User has pasted in some data, modified to " + event.change);
            // process keystroke, say, change to upper case.
            event.change = event.change.toUpperCase();
            cp("User enters data into editable field: " + event.change);
            break;
    }
    /* Display the input so far, AFMergeChange defined in aform.js */
    cp("User Input so far: " + AFMergeChange(event) );
}
/* event.changeEx != ", so user has simply selected a menu item and not typed or pasted into the edit box. */
else {
    cp("Menu Item selected");
}
}

Understanding this example is key to successfully handling a listbox or combobox.

Example 2

For an example of the use of changeEx with text fields, see the example following fieldFull.
**commitKey**

| 4.0 |  |  |

Determines how a form field will lose focus. Values are:

0: Value was not committed (for example, escape key was pressed).
1: Value was committed because of a click outside the field using the mouse.
2: Value was committed because of hitting the enter key.
3: Value was committed by tabbing to a new field.

*Type: Number  Access: R.*

**Example**

To automatically display an alert dialog after a field has been committed add the following to the field's format script:

```javascript
if (event.commitKey != 0)
    app.alert("Thank you for your new field value.");
```

**fieldFull**

| 6.0 |  |  |

Only available in keystroke events for text fields. Set to true when the user attempts to enter text which does not fit in the field due to either a space limitation or the maximum character limit. When **fieldFull** is true, **event.changeEx** is set to the entire text string the user attempted to enter and **event.change** is the text string cropped to what fits within the field.

*Type: Boolean  Access: R  Events: Keystroke.*

**Example**

Test whether user has overfilled the text field.

```javascript
// Custom Keystroke script for a text field. Initially, the field is set
// so that text does not scroll.
if ( event.fieldFull )
{
    app.alert("You've filled the given space with text,
    + " and as a result, you've lost some text. I'll set the field to"
    + " scroll horizontally, and paste in the rest of your"
    + " missing text.");
    event.target.doNotScroll = false;
    event.change = event.changeEx;
}
```
keyDown

Available only during a keystroke event for listbox and combobox. For a listbox or the pop-up part of a combobox, the value is true if the arrow keys were used to make a selection, false otherwise.

For the combobox, keyDown is only available if the pop-up part of it is used, that is, a selection (with the mouse or the keyboard) is being made from the pop-up. If the combo is editable and the user types in an entry, the Field/Keystroke event behaves as for a text field (that is, there are no changeEx or keyDown event properties).

Type: Boolean Access: R.

modifier

Whether the modifier key is down during a particular event. The modifier key on the Microsoft Windows platform is Control and on the Macintosh platform is Option or Command. The modifier is not supported on UNIX.

Type: Boolean Access: R.

name

The name of the current event as a text string. The type and name together uniquely identify the event. Valid names are:

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

Type: String Access: R Events: all.

rc

Used for validation. Indicates whether a particular event in the event chain should succeed. Set to false to prevent a change from occurring or a value from committing. By default rc is true.
### richChange

The `richChange` property specifies the change in value that the user has just typed. The `richChange` property is only defined for rich text fields and mirrors the behavior of the `event.change` property. When `event.fieldFull` is true, `richChangeEx` is set to the entire rich formatted text string the user attempted to enter and `event.richChange` is the rich formatted text string cropped to what fits within the field. Use `event.changeEx` and `event.change` to handle (plain) text fields.

- **Type:** Array of `Span Objects`
- **Access:** R/W
- **Events:** Keystroke

**Example**

This example changes the keystroke to uppercase, alternately colors the text blue and red, and switches underlining off and on.

```javascript
var span = event.richChange;
for (var i=0; i< span.length; i++) {
    span[i].text = span[i].text.toUpperCase();
    span[i].underline = !span[i].underline;
    span[i].textColor = (span[i].underline) ? color.blue : color.red;
}
event.richChange = span;
```

### richChangeEx

The `richChangeEx` property is only defined for rich text fields and mirrors the behavior of the `event.changeEx` property for text fields. The value of `richChangeEx` is an array of `Span Objects` which specify both the text entered into the field and the formatting. Keystrokes are represented as single member arrays, while rich text pasted into a field is represented as an array of arbitrary length.

- **Type:** Array of `Span Objects`
- **Access:** R/W
- **Events:** Keystroke
When `event.fieldFull` is `true`, `richChangeEx` is set to the entire rich formatted text string the user attempted to enter and `event.richChange` is the rich formatted text string cropped to what fits within the field. Use `event.changeEx` (and `event.change`) to handle (plain) text fields.

**Type:** Array of **Span Objects**

**Access:** R/W

**Events:** Keystroke

Related objects and properties are the **Span Object**, `field/defaultStyle`, `field/richText`, `field/richValue`, `event/richChange`, `event/richValue`, and `annot/richContents`.

**Example**

If the text field is filled up by the user, allow additional text by setting the field to scroll.

```javascript
if ( event.fieldFull )
{
  app.alert("You've filled the given space with text,
+ " and as a result, you've lost some text. I'll set the field to"
+ " scroll horizontally, and paste in the rest of your"
+ " missing text.");
  event.target.doNotScroll = false;
  if ( event.target.richText )
    event.richChange = event.richChangeEx
  else
    event.change = event.changeEx;
}
```

See also `event.fieldFull`.

### richValue

This property mirrors the `field/richValue` property of the field and the `event/value` property for each event.

**Type:** Array of **Span Objects**

**Access:** R/W

**Events:** Keystroke

Related objects and properties are the **Span Object**, `field/defaultStyle`, `field/richText`, `field/richValue`, `event/richChange`, `event/richChangeEx`, and `annot/richContents`.

**Example**

This example turns all bold text into red underlined text.

```javascript
// Custom Format event for a rich text field.
var spans = event.richValue;
for ( var i = 0; i < spans.length; i++ )
{
  if( spans[i].fontWeight >= 700 )
  {
```
spans[i].textColor = color.red;
spans[i].fontWeight = 400; // change to default weight
spans[i].underline = true;
}
}
event.richValue = spans;

**selEnd**

The ending position of the current text selection during a keystroke event.

*Type: Integer*  
*Access: R/W.*

**Example**

This is the function `AFMergChange` taken from the file `AFoms.js`, in the application JavaScripts folder. This function merges the last change (of a text field) with the uncommitted change. This function uses both `selEnd` and `selStart`.

```javascript
function AFMergChange(event)
{
    var prefix, postfix;
    var value = event.value;

    if(event.willCommit) return event.value;
    if(event.selStart >= 0)
        prefix = value.substring(0, event.selStart);
    else prefix = "";
    if(event.selEnd >= 0 && event.selEnd <= value.length)
        postfix = value.substring(event.selEnd, value.length);
    else postfix = "";
    return prefix + event.change + postfix;
}
```

**selStart**

The starting position of the current text selection during a keystroke event.

*Type: Integer*  
*Access: R/W.*

**Example**

See the example following `selEnd`.

**shift**

Whether the shift key is down during a particular event.

*Type: Boolean*  
*Access: R.*
**Example**

The following is a mouse up button action.

```javascript
if (event.shift)
    this.gotoNamedDest("dest2");
else
    this.gotoNamedDest("dest1");
```

**source**

The **Field Object** that triggered the calculation event. This is usually different from the target of the event, that is, the field that is being calculated.

*Type: object  Access: R.*

**target**

The target object that triggered the event. In all mouse, focus, blur, calculate, validate, and format events it is the **Field Object** that triggered the event. In other events, such as page open and close, it is the **Doc Object** or **this Object**.

*Type: object  Access: R.*

**targetName**

Tries to return the name of the JavaScript being executed. Can be used for debugging purposes to help better identify the code causing exceptions to be thrown. Common values of **targetName** include:

- the folder-level script file name for **App/Init** events;
- the Doc-level script name for **Doc/Open** events;
- the PDF file name being processed for **Batch/Exec** events;
- the Field name for **Field/Blur**, **Field/Calculate**, **Field/Format**, **Field/Keystroke**, **Field/Mouse Down**, **Field/Mouse Enter**, **Field/Mouse Exit**, **Field/Mouse Up** and **Field/Validate** events.
- the Menu item name for **Menu/Exec** events.

If there is an identifiable name, Acrobat EScript reports **targetName** when an exception is thrown.

*Type: String  Access: R.*
**Example**

The first line of the folder level JavaScript file `conserve.js` has an error in it, when the Acrobat Viewer started, an exception is thrown. The standard message reveals quite clearly the source of the problem.

```
MissingArgError: Missing required argument.
App.alert:1:Folder-Level:App:conserve.js
  ==> Parameter cMsg.
```

**type**

The type of the current event as a text string. The type and `name` together uniquely identify the event. Valid types are:

- Batch
- External
- Console
- Bookmark
- App
- Link
- Doc
- Field
- Page
- Menu

*Type: String*  
*Access: R.*

**value**

This property has different meanings for different `field` events.

**Field/Validate event**

For the `Field/Validate` event, this is the value that the field contains when it is committed. For a `combobox`, this is the `face value`, not the `export value` (see `changeEx` for the export value).

**Example**

For example, the following JavaScript verifies that the field value is between zero and 100.

```javascript
if (event.value < 0 || event.value > 100) {
    app.beep(0);
    app.alert("Invalid value for field " + event.target.name);
    event.rc = false;
}
```
**Field/Calculate event**

For a Field/Calculate event, JavaScript should set this property. It is the value that the field should take upon completion of the event.

**Example**

For example, the following JavaScript sets the calculated value of the field to the value of the SubTotal field plus tax.

```javascript
var f = this.getField("SubTotal");
event.value = f.value * 1.0725;
```

**Field/Format event**

For a Field/Format event, JavaScript should set this property. It is the value used when generating the appearance for the field. By default, it contains the value that the user has committed. For a **combo**box, this is the face value, not the export value (see changeEx for the export value).

**Example**

For example, the following JavaScript formats the field as a currency type of field.

```javascript
event.value = util.printf("$%.2f", event.value);
```

**Field/Keystroke event**

The current value of the field. If modifying a text field, for example, this is the text in the text field before the keystroke is applied.

**Field/Blur and Field/Focus events**

The current value of the field. During these two events, `event.value` is read-only, that is, the field value cannot be changed by setting `event.value`.

Beginning with Acrobat 5.0, for a **listbox** that allows multiple selections (see `field.multipleSelection`), if the field value is an array (that is, there are multiple selections currently selected), `event.value` returns an empty string when getting, and does not accept setting.

*Type: various*  
*Access: R/W.*

**willCommit**

Verifies the current keystroke event before the data is committed. This is useful to check the target form field values and for example verify if character data instead of numeric data was entered. JavaScript sets this property to `true` after the last keystroke event and before the field is validated.

*Type: Boolean*  
*Access: R.*

**Example**

```javascript
var value = event.value
if (event.willCommit)
```
// Final value checking.
else
  // Keystroke level checking.

**FDF Object**

| 6.0 | S |

This object corresponds to a PDF-encoded data exchange file. The most familiar use of FDF files is to contain forms data that is exported from a PDF file. FDF files can also be used as general purpose data files. It is for this later purpose that the FDF object exists.

(Security S): All methods and properties marked with S in its quickbar are available only during batch, console, application initialization and menu events.

**FDF Properties**

**deleteOption**

| 6.0 | D | S | X |

Indicates whether the FDF file should be automatically deleted after it is processed. This is a generic value that may or may not be used, depending on the content of the FDF file and how it is processed. It is used for embedded files beginning in Acrobat 6.0. Allowed values are

0 (default): Acrobat will automatically delete the FDF file after processing
1: Acrobat will not delete the FDF file after processing (however a web or email browser may still delete the file).
2: Acrobat will prompt the user to determine whether to delete the FDF file after processing (however a web or email browser may still delete the file).

Type: Integer Access: R/W.

**isSigned**

| 6.0 | D | X |

Returns **true** if the FDF data file is signed.

Type: Boolean Access: R.
Example
See if the fdf is signed.

```javascript
var fdf = app.openFDF("/C/temp/myDoc.fdf");
console.println( "It is " + fdf.isSigned + " that this FDF is signed");
fdf.close();
```

See a more complete example following `fdf.signatureSign`

**numEmbeddedFiles**

The number of files embedded in the FDF file. If the FDF object is a valid FDF file, no exceptions will be thrown.

*Type: Integer*  
*Access: R.*

Example
Create a new FDF object, embed a PDF doc, save the FDF, open the FDF again, and count the number of embedded files.

```javascript
var fdf = app.newFDF()
fdf.addEmbeddedFile("/C/myPDFs/myDoc.pdf")
fdf.save("/c/temp/myDocWrapper.fdf");
fdf = app.openFDF("/c/temp/myDocWrapper.fdf");
console.println("The number of embedded files = "+ fdf.numEmbeddedFiles);
fdf.close();
```

**FDF Methods**

**addContact**

The number of files embedded in the FDF file. If the FDF object is a valid FDF file, no exceptions will be thrown.

*Type: Integer*  
*Access: R.*

Example
Create a new FDF object, embed a PDF doc, save the FDF, open the FDF again, and count the number of embedded files.

```javascript
var fdf = app.newFDF()
fdf.addEmbeddedFile("/C/myPDFs/myDoc.pdf")
fdf.save("/c/temp/myDocWrapper.fdf");
fdf = app.openFDF("/c/temp/myDocWrapper.fdf");
console.println("The number of embedded files = "+ fdf.numEmbeddedFiles);
fdf.close();
```
### Example

```javascript
var oEntity={firstName:"Fred", lastName:"Smith", fullName:"Fred Smith"};

var f = app.newFDF();
f.addContact( oEntity );
f.save( "/c/temp/FredCert.fdf" );
```

### addEmbeddedFile

Add the specified file to the end of the array of embedded files in the FDF file. Anyone opening the FDF file will be instructed to save the embedded file or files according to `nSaveOptions`. If the embedded file is a PDF file, the file will be opened and displayed in the viewer. If the embedded file is an FDF file, the file will be opened by the viewer for processing. FDF files containing embedded files were supported beginning with Acrobat 4.05. An example use for embedding PDF files is when these files are hosted on an HTTP server and it is desired that the user clicks to download and save the PDF file, rather than viewing the file in the browser. There is no relationship between these embedded files and files that are associated with forms data that is stored in an FDF file.

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cDIPath</code></td>
<td>(optional) A device-independent absolute path to a file on the user’s hard drive. If not specified, the user is prompted to locate a file. See File Specification Strings in the PDF Reference for the exact syntax of the path.</td>
</tr>
</tbody>
</table>
| `nSaveOptions`          | (optional) How the embedded file will be presented to the person opening this FDF file, where the file will be saved, and whether the file will be deleted after it is saved. Values are:  
  ● 0: The file will be automatically saved to the Acrobat document folder.  
  ● 1 (the default): The user will be prompted for a filename to which to save the embedded file.  
  ● 2: Should not be used.  
  ● 3: The file will be automatically saved as a temporary file and deleted during cleanup (when Acrobat is closed).  
In Acrobat 4.05 through 5.05, for values of 0 and 3, the user is prompted for the location of the save folder if they have not already set this value.  
For all values of `nSaveOptions`, if the file is a PDF or FDF file it is automatically opened by Acrobat once it is saved. |
Returns

Throws an exception if this operation could not be completed, otherwise returns the number of embedded files that are now in the FDF file.

Example

Create a new FDF, embed a PDF doc, then save.

```javascript
var fdf = app.newFDF();
fdf.addEmbeddedFile("/C/myPDBs/myDoc.pdf");
fdf.save("/c/temp/myDocs.fdf");
```

addRequest

Adds a request to the FDF file. There can be only one request in an FDF file. If the FDF file already contains a request, it is replaced with this new request.

Parameters

<table>
<thead>
<tr>
<th>cType</th>
<th>What is being requested. Currently the only valid value is the string “CMS”, which is a request for contact information.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cReturnAddress</td>
<td>The return address string for the request. This must begin with <code>mailto:</code>, <code>http:</code> or <code>https:</code> and be of the form &quot;<a href="http://www.acme.com/cgi.pl">http://www.acme.com/cgi.pl</a>&quot; or &quot;<a href="mailto:jdoe@adobe.com">mailto:jdoe@adobe.com</a>&quot;.</td>
</tr>
<tr>
<td>cName</td>
<td>(optional) The name of the person or organization that has generated the request.</td>
</tr>
</tbody>
</table>
Parameters
None

Returns
Throws an exception if there is an error.
See the `fdf.save` method, which also closes an FDF file.

Example
The example following `addEmbeddedFile` illustrates `fdf.close`.

mail

6.0  S  X

This method saves the FDF Object as a temporary FDF file and mails this file as an attachment to all recipients, with or without user interaction. The temporary file is deleted once it is no longer needed.
See also `mailGetAddr`, `mailMsg`, `mailDoc`, `mailForm` and `report.mail`.

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUI</td>
<td>(optional) Whether to display a user interface. If true (the default) the rest of the parameters are used to seed a compose-new-message window that is displayed to the user. If false, the cTo parameter is required and all others are optional.</td>
</tr>
<tr>
<td>cTo</td>
<td>(optional) A semicolon-separated list of recipients for the message.</td>
</tr>
<tr>
<td>cCc</td>
<td>(optional) A semicolon-separated list of CC recipients for the message.</td>
</tr>
<tr>
<td>cBcc</td>
<td>(optional) A semicolon-separated list of BCC recipients for the message.</td>
</tr>
<tr>
<td>cSubject</td>
<td>(optional) The subject of the message. The length limit is 64k bytes.</td>
</tr>
<tr>
<td>cMsg</td>
<td>(optional) The content of the message. The length limit is 64k bytes.</td>
</tr>
</tbody>
</table>

Returns
Throws an exception if there is an error.

Example
```
var fdf = app.openFDF( "/c/temp/myDoc.fdf" );
/* This will pop up the compose new message window */
```
fdf.mail();

/* This will send out the mail with the attached FDF file to
fun1@fun.com and fun2@fun.com */
fdf.mail( false, "fun1@fun.com", "fun2@fun.com", ",
  "This is the subject", "This is the body.");

**save**

Save the FDF Object as a file. A save will always occur. The file is closed when it is saved, and
the FDF object no longer contains a valid object reference.

See the fdf.close method, which also closes an FDF file.

**Parameters**

<table>
<thead>
<tr>
<th>cDIPath</th>
<th>The device-independent path of the file to be saved.</th>
</tr>
</thead>
</table>

**Note:** (Security ⑤): cDIPath must be a Safe Path and must have an
extension of .fdf.

**Returns**

Throws an exception if there is an error.

**Example**

Create a new FDF, embed a PDF doc, then save.

```
var fdf = app.newFDF()
fdf.addEmbeddedFile("/C/myPDFs/myDoc.pdf");
fdf.save("/c/temp/myDocs.fdf");
```

**signatureClear**

If the FDF Object is signed, clears the signature and returns true if successful. Does
nothing if the FDF object is not signed. Does not save the file.

**Parameters**

None

**Returns**

true on success.
signatureSign

Sign the FDF Object with the specified security object. FDF objects can be signed only once. The FDF object is signed in memory and is not automatically saved as a file to disk. Call save to save the FDF object after it is signed. Call signatureClear to clear FDF signatures.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oSig</td>
<td>The SecurityHandler Object that is to be used to sign. Security objects normally require initialization before they can be used for signing. Check the documentation for your security handler to see if it is able to sign FDF files. The signPDF property of the SecurityHandler Object will indicate whether a particular security object is capable of signing FDF files.</td>
</tr>
<tr>
<td>oInfo</td>
<td>(optional) A SignatureInfo Object containing the writable properties of the signature.</td>
</tr>
<tr>
<td>nUI</td>
<td>(optional) The type of dialog to show when signing. Values are: 0: Show no dialog. 1: Show a simplified dialog with no editable fields (fields can be provided in oInfo). 2: Show a more elaborate dialog that includes editable fields for reason, location and contact information. The default is 0.</td>
</tr>
<tr>
<td>cUISignTitle</td>
<td>(optional) The title to use for the sign dialog. This is only used if nUI is non-zero.</td>
</tr>
<tr>
<td>cUISelectMsg</td>
<td>(optional) A message to display when a user is required to select a resource for signing, such as selecting a credential. It is used only when nUI is non-zero.</td>
</tr>
</tbody>
</table>

Returns

true if the signature was applied successfully, false otherwise.

Example

Open existing FDF data file and sign.

```javascript
var eng = security.getHandler( "Adobe.PPKLite" );
eng.login("myPassword" ,"/c/test/Acme.pfx");
var myFDF = app.openFDF( "/c/temp/myData.fdf" );
if( !myFDF.isSigned ) {
    myFDF.signatureSign(eng, {}, 1, "Sign Embedded File FDF",
```
"Please select a Digital ID to use to sign your "
+ "embedded file FDF."
});
myFDF.save( "/c/temp/myData.fdf" );
});

**signatureValidate**

| 6.0 | X |

Validate the signature of an FDF Object and return a SignatureInfo Object specifying the properties of the signature.

**Parameters**

<table>
<thead>
<tr>
<th>oSig</th>
<th>(optional) The security handler to be used to validate the signature. Can be either a SecurityHandler Object or a generic object with the following properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- oSecHdlr: The SecurityHandler Object to use to validate this signature.</td>
</tr>
<tr>
<td></td>
<td>- bAltSecHdlr: A boolean. If true, an alternate security handler, selected based on user preference settings, may be used to validate the signature. The default is false, meaning that the security handler returned by the signature's handlerName property is used to validate the signature. This parameter is not used if oSecHdlr is provided.</td>
</tr>
<tr>
<td></td>
<td>- If oSig not supplied, the security handler returned by the signature's handlerName property is used to validate the signature.</td>
</tr>
<tr>
<td>bUI</td>
<td>(optional) When true, allow UI to be shown, if necessary, when validating the data file. UI may be used to select a validation handler if none is specified.</td>
</tr>
</tbody>
</table>

**Returns**

A SignatureInfo Object. The signature status is described in status property.

**Example**

```javascript
fdf = app.openFDF("/c/temp/myDoc.fdf");
eng = security.getHandler( "Adobe.PPKLite" );
if (fdf.isSigned)
{
  var oSigInfo = fdf.signatureValidate({
    oSig: eng,
    bUI: true
  });
  console.println("Signature Status: " + oSigInfo.status);
  console.println("Description: " + oSigInfo.statusText);
}
Field Object

The field object represents an Acrobat form field (that is, a field created using the Acrobat form tool or doc.addField). In the same manner that an author might want to modify an existing field’s properties like the border color or font, the Field object gives the JavaScript user the ability to perform the same modifications.

Field Access from JavaScript

Before a field can be accessed, it must be “bound” to a JavaScript variable through a method provided by the Doc Object method interface. More than one variable may be bound to a field by modifying the field’s object properties or accessing its methods. This affects all variables bound to that field.

```javascript
var f = this.getField("Total");
```

This example allows the script to now manipulate the form field Total by using the variable f.

Fields can be arranged hierarchically within a document. For example, form fields can have names like “FirstName” and “LastName”. These are called flat names, there is no association between these fields. By changing the field names slightly, a hierarchy of fields within the document can be created. For example, if "FirstName" and "LastName" are changed to "Name.First" and "Name.Last", a tree of fields is formed. The period (‘.’) separator in Acrobat Forms is used to denote a hierarchy shift. The “Name” portion of these fields is the parent, and “First” and “Last” are the children. There is no limit to the depth of a hierarchy that can be constructed but it is important that the hierarchy remain manageable. It is also important to clarify some terminology: the field “Name” is known as an internal field (that is, it has no visible manifestation) and the fields “First” and “Last” are terminal fields (and show up on the page).

A useful property about Acrobat Form fields is that fields that share the same name also share the same value. Terminal fields can have different presentations of that data; they can appear on different pages, be rotated differently, have a different font or background color, and so on, but they have the same value. This means that if the value of one presentation of a terminal field is modified, all others with the same name get updated automatically. We refer to each presentation of a terminal field as a widget.

Individual widgets do not have names. Each individual widget is identified by index (0-based) within its terminal field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). You can easily determine what the index is for a specific widget by looking at the “Fields” panel in Acrobat. It is the number that follows the ‘#’ sign in the field name shown (in Acrobat 6, the widget index is only displayed if the field has more than one widget). You can double-click an entry
in the “Fields” panel to go to the corresponding widget in the document. Alternatively, if you select a field in the document, the corresponding entry in the “Fields” panel is highlighted.

**Doc.getField() Extended to Widgets**

A new notation is available when calling `getField` which can be used to retrieve the Field object of one individual widget of a field. This new notation consists of appending a “.” followed by the widget index to the field name passed. When this approach is used, the field object returned by `getField` encapsulates only one individual widget. You can use the field objects returned this way in any place you would use a field object returned by simply passing the unaltered field name. However, the set of nodes that are affected may vary, as shown in the following table.

<table>
<thead>
<tr>
<th>Action</th>
<th>Field Object that Represents All Widgets</th>
<th>Field Object that Represents One Specific Widget</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get a widget property</td>
<td>Gets property of widget # 0</td>
<td>Gets property of that widget</td>
</tr>
</tbody>
</table>
| Set a widget property   | Sets property of all widgets that are children of that field
a                        | Sets property of that widget              |
| Get a field property    | Gets property of that field              | Gets property of parent field                  |
| Set a field property    | Sets property of that field              | Sets property of parent field                  |

a. Except for the `rect` property and the `setFocus` method. For these cases it applies to widget # 0.

The following example changes the `rect` property of the second radio button (the first would have index 0) of the field "my radio".

```javascript
var f = this.getField("my radio.1");
f.rect = [360, 677, 392, 646];
```

**Field versus Widget Attributes**

Some of the properties of the field object in JavaScript truly live at the field level, and apply to all widgets that are children of that field. A good example is `value`. Other
properties are, in fact, widget-specific. A good example is `rect`. The following table shows which attributes live at the field level and which at the widget level.

<table>
<thead>
<tr>
<th>Field Object Properties and Methods that Affect Widget-Level Attributes</th>
<th>Field Object Properties and Methods that Affect Field-Level Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>alignment, borderStyle, buttonAlignX, buttonAlignY, buttonPosition, buttonScaleHow, buttonScaleWhen, display, fillColor, hidden, highlight, lineWidth, print, rect, strokeColor, style, textColor, textField, textSize, buttonGetCaption, buttonGetIcon, buttonImportIcon, buttonSetCaption, buttonSetIcon, checkThisBox(^a), defaultIsChecked(^a), isBoxChecked(^a), isDefaultChecked(^a), setAction(^b), setFocus</td>
<td>calcOrderIndex, charLimit, comb, currentValueIndices, defaultValue, doNotScroll, doNotSpellCheck, delay, doc, editable, exportValues, fileSelect, multiline, multipleSelection, name, numItems, page, password, readonly, required, submitName, type, userName, value, valueAsString, clearItems, browseForFileToSubmit, deleteItemAt, getItemAt, insertItemAt, setAction(^b), setItems, signatureInfo, signatureSign, signatureValidate</td>
</tr>
</tbody>
</table>

\(^a\) These methods take a widget index, `nWidget`, as parameter. If you invoke these methods on a Field object "f" that represents one specific widget, then the `nWidget` parameter is optional (and is ignored if passed), and the method acts on the specific widget encapsulated by "f".

\(^b\) Some actions live at the field level, and some at the widget level. The former includes "Keystroke", "Validate", "Calculate", "Format". The latter includes "MouseUp", "MouseDown", "MouseEnter", "MouseExit", "OnFocus", "OnBlur".

### Field Properties

**alignment**

Controls how the text is laid out within the text field. Values are:

- left
- center
- right

*Type: String*  
*Access: R/W*  
*Fields: text.*

**Example**

```javascript
var f = this.getField("MyText");
f.alignment = "center";
```
**borderStyle**

The border style for a field. Valid border styles are

- `solid`
- `dashed`
- `beveled`
- `inset`
- `underline`

The border style determines how the border for the rectangle is drawn. The `border` object is a static convenience constant that defines all the border styles of a field, as shown in the following table:

<table>
<thead>
<tr>
<th>Type</th>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>solid</td>
<td><code>border.s</code></td>
<td>Strokes the entire perimeter of the rectangle with a solid line.</td>
</tr>
<tr>
<td>beveled</td>
<td><code>border.b</code></td>
<td>Equivalent to the <code>solid</code> style with an additional beveled (pushed-out appearance) border applied inside the solid border.</td>
</tr>
<tr>
<td>dashed</td>
<td><code>border.d</code></td>
<td>Strokes the perimeter with a dashed line</td>
</tr>
<tr>
<td>inset</td>
<td><code>border.i</code></td>
<td>Equivalent to the <code>solid</code> style with an additional inset (pushed-in appearance) border applied inside the solid border.</td>
</tr>
<tr>
<td>underline</td>
<td><code>border.u</code></td>
<td>Strokes the bottom portion of the rectangle's perimeter.</td>
</tr>
</tbody>
</table>

Type: `String`  
Access: `R/W`  
Fields: all.

**Example**

The following example illustrates how to set the border style of a field to `solid`:

```javascript
var f = this.getField("MyField");
f.borderStyle = border.s; /* border.s evaluates to "solid" */
```

**buttonAlignX**

Controls how space is distributed from the left of the button face with respect to the icon. It is expressed as a percentage between 0 and 100, inclusive. The default value is 50. If the
icon is scaled anamorphically (which results in no space differences) then this property is not used.

*Type: Integer  Access: R/W  Fields: button*

### buttonAlignY

5.0

Controls how unused space is distributed from the bottom of the button face with respect to the icon. It is expressed as a percentage between 0 and 100 inclusive. The default value is 50. If the icon is scaled anamorphically (which results in no space differences) then this property is not used.

*Type: Integer  Access: R/W  Fields: button*

### buttonFitBounds

6.0

When *true*, the extent to which the icon may be scaled is set to the bounds of the button field; the additional icon placement properties are still used to scale/position the icon within the button face.

In previous versions of Acrobat, the width of the field border was always taken into consideration when scaling an icon to fit a button face, even when no border color was specified. Setting this property to *true* when a border color has been specified for the button will cause an exception to be raised.

*Type: Boolean  Access: R/W  Fields: button*

### buttonPosition

5.0

Controls how the text and the icon of the button are positioned with respect to each other within the button face. The convenience *position* object defines all of the valid alternatives:

<table>
<thead>
<tr>
<th>Icon/Text Placement</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text Only</td>
<td>position.textOnly</td>
</tr>
<tr>
<td>Icon Only</td>
<td>position.iconOnly</td>
</tr>
</tbody>
</table>

*Acrobat JavaScript Scripting Reference  
Field Properties*
### Icon/Text Placement

<table>
<thead>
<tr>
<th>Icon/Text Placement</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icon top, Text bottom</td>
<td>position.iconTextV</td>
</tr>
<tr>
<td>Text top, Icon bottom</td>
<td>position.textIconV</td>
</tr>
<tr>
<td>Icon left, Text right</td>
<td>position.iconTextH</td>
</tr>
<tr>
<td>Text left, Icon right</td>
<td>position.textIconH</td>
</tr>
<tr>
<td>Text in Icon (overlaid)</td>
<td>position.overlay</td>
</tr>
</tbody>
</table>

**Type:** Integer  
**Access:** R/W  
**Fields:** button

### buttonScaleHow

| 5.0 |  |

Controls how the icon is scaled (if necessary) to fit inside the button face. The convenience scaleHow object defines all of the valid alternatives:

<table>
<thead>
<tr>
<th>How is Icon Scaled</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proportionally</td>
<td>scaleHow.proportional</td>
</tr>
<tr>
<td>Non-proportionally</td>
<td>scaleHow.anamorphic</td>
</tr>
</tbody>
</table>

**Type:** Integer  
**Access:** R/W  
**Fields:** button

### buttonScaleWhen

| 5.0 |  |

Controls when an icon is scaled to fit inside the button face. The convenience scaleWhen object defines all of the valid alternatives:

<table>
<thead>
<tr>
<th>When is Icon Scaled</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>scaleWhen.always</td>
</tr>
<tr>
<td>Never</td>
<td>scaleWhen.never</td>
</tr>
<tr>
<td>If icon is too big</td>
<td>scaleWhen.tooBig</td>
</tr>
<tr>
<td>If icon is too small</td>
<td>scaleWhen.tooSmall</td>
</tr>
</tbody>
</table>

**Type:** Integer  
**Access:** R/W  
**Fields:** button
calcOrderIndex

Changes the calculation order of fields in the document. When a computable text or combobox field is added to a document, the field’s name is appended to the calculation order array. The calculation order array determines the order fields are calculated in the document. The calcOrderIndex property works similarly to the Calculate tab used by the Acrobat Form tool.

Type: Integer  Access: R/W  Fields: combobox, text.

Example

```javascript
var a = this.getField("newItem");
var b = this.getField("oldItem");
a.calcOrderIndex = b.calcOrderIndex + 1;
```

In this example, `getField` gets the "newItem" field that was added after "oldItem" field. It then changes the `calcOrderIndex` of the "oldItem" field so that it is calculated before "newItem" field.

charLimit

Limits the number of characters that a user can type into a text field.

See also `event.fieldFull`.

Type: Integer  Access: R/W  Fields: text.

Example

Set a limit on the number of characters that can be typed into a field.

```javascript
var f = this.getField("myText");
f.charLimit = 20;
```

comb

If set to true, the field background is drawn as series of boxes (one for each character in the value of the field) and the each character of the content is drawn within those boxes. The number of boxes drawn is determined from the field.charLimit property.

It applies only to text fields. The setter will also raise if any of the following field properties are also set: multiline, password, and fileSelect. A side-effect of setting this property is that the doNotScroll property is also set.
Field Properties

**Example**

Create a comb field in the upper left corner of a newly created document.

```javascript
var myDoc = app.newDoc();  // create a blank doc
var Bbox = myDoc.getPageBox("Crop");  // get crop box
var inch = 72;

// add a text field at the top of the document
var f = myDoc.addField("Name.Last", "text", 0,
    [ inch, Bbox[1]-inch, 3*inch, Bbox[1]- inch - 14 ] )
// add some attributes to this text field
f.strokeColor = color.black;
f.textColor = color.blue;
f.fillColor = ["RGB",1,0.66,0.75]

f.comb = true  // declare this is a comb field
f.charLimit = 10;  // Max number of characters
```

**commitOnSelChange**

Controls whether a field value is committed after a selection change. When `true`, the field value is committed immediately when the selection is made. When `false`, the user can change the selection multiple times without committing the field value; the value is committed only when the field loses focus, that is, when the user clicks outside the field.

Type: Boolean  Access: R/W  Fields: combobox, listbox

**currentValueIndices**

Reads and writes single or multiple values of a `listbox` or `combobox`.

**Read**

Returns the options-array indices of the strings that are the value of a `listbox` or `combobox` field. These indices are 0-based. If the value of the field is a single string then it returns an integer. Otherwise, it returns an array of integers sorted in ascending order. If the current value of the field is not a member of the set of offered choices (as could happen in the case of an editable combobox) it returns -1.

**Write**

Sets the value of a `listbox` or `combobox`. It accepts either a single integer, or an array of integers, as an argument. To set a single string as the value, pass an integer which is the 0-
based index of that string in the options array. Note that in the case of an editable
**combobox**, if the desired value is not a member of the set of offered choices, then you
must set the **value** instead. Except for this case, **currentValueIndices** is the
preferred way to set the value of a **listbox** or **combobox**.

To set a multiple selection for a **listbox** that allows it, pass an array as argument to this
property, containing the indices (sorted in ascending order) of those strings in the options
array. This is the only way to invoke multiple selection for a **listbox** from JavaScript. The
ability for a **listbox** to support multiple selection can be set through
**multipleSelection**.

Related Field methods and properties include **numItems**, **getItemAt**, **insertItemAt**, **deleteItemAt** and **setItems**.

**Type:** Integer | Array  
**Access:** R/W  
**Fields:** **combobox**, **listbox**

**Example (Read)**

The script below is a mouse up action of a button. The script gets the current value of a list
box.

```javascript
var f = this.getField("myList");
var a = f.currentValueIndices;
if (typeof a == "number") // a single selection
    console.println("Selection: " + f.getItemAt(a, false));
else { // multiple selections
    console.println("Selection:");
    for (var i = 0; i < a.length; i++)
        console.println("   " + f.getItemAt(a[i], false));
}
```

**Example (Write)**

The following code, selects the second and fourth (0-based index values, 1 and 3,
respectively) in a listbox.

```javascript
var f = this.getField("myList");
f.currentValueIndices = [1, 3];
```

**defaultStyle**

This property defines the default style attributes for the form field. If the user clicks into an
empty field and begins entering text without changing properties using the property
toolbar, these are the properties that will be used. This property is a single **Span Object**
without a **text** property. Some of the properties in the default style span mirror the
properties of the field object. Changing these properties also modifies the
**defaultStyle** property for the field and vice versa.
The following table details the properties of the field object that are also in the default style and any differences between their values.

<table>
<thead>
<tr>
<th>Field Properties</th>
<th>defaultStyle (Span Properties)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alignment</td>
<td>alignment</td>
<td>The <code>alignment</code> property has the same values for both the default style and the field object.</td>
</tr>
<tr>
<td>textFont</td>
<td>fontFamily, fontStyle, fontWeight</td>
<td>The value of this field property is a complete font name which represents the font family, weight and style. In the default style property each property is represented separately. If an exact match for the font properties specified in the default style cannot be found a similar font will be used or synthesized.</td>
</tr>
<tr>
<td>textColor</td>
<td>textColor</td>
<td>The <code>textColor</code> property has the same values for both the default style and the field object.</td>
</tr>
<tr>
<td>textSize</td>
<td>textSize</td>
<td>The <code>textSize</code> property has the same values for both the default style and the field object.</td>
</tr>
</tbody>
</table>

**Notes:** When a field is empty, `defaultStyle` is the style used for newly entered text. If a field already contains text when when the `defaultStyle` is changed the text will not pick up any changes to `defaultStyle`; newly entered text will use the attributes of the text it is inserted into (or specified with the toolbar).

When pasting rich text into a field any unspecified attributes in the pasted rich text will be filled with those from the `defaultStyle`.

Superscript and Subscript are ignored in the `defaultStyle`.

**Type:** Span Object  
**Access:** R/W  
**Fields:** rich text.

**Example**

Change the default style for a text field.

```javascript
var style = this.getField("Text1").defaultStyle;

style.textColor = color.red;
style.textSize = 18;

// if Courier Std is not found on the user’s system, use a monospace
style.fontFamily = ["Courier Std", "monospace" ];

this.getField("Text1").defaultStyle = style;
```
**defaultValue**

Exposes the default value of a field. This is the value that the field is set to when the form is reset. For **comboboxes** and **listboxes** either an export or a user value can be used to set the default. In the case of a conflict (for example, the field has an export value and a user value with the same string but these apply to different items in the list of choices), the export value is matched against first.

*Type: String  Access: R/W  Fields: all except button, signature.*

**Example**

```javascript
var f = this.getField("Name");
f.defaultValue = "Enter your name here.";
```

**doNotScroll**

When `true`, the text field does not scroll and the user, therefore, is limited by the rectangular region designed for the field. Setting this property to `true` or `false` corresponds to checking or unchecking the “Scroll long text” field in the Options tab of the field.

*Type: Boolean  Access: R/W  Fields: text.*

**doNotSpellCheck**

When `true`, spell checking is *not* performed on this editable text field. Setting this property to `true` or `false` corresponds to unchecking or checking the "Check spelling" attribute in the Options tab of the Field Properties dialog.

*Type: Boolean  Access: R/W  Fields: combobox (editable), text.*

**delay**

Delays the redrawing of a field’s appearance. It is generally used to buffer a series of changes to the properties of the field before requesting that the field regenerate its appearance. Setting the property to `true` forces the field to wait until `delay` is set to `false`. The update of its appearance then takes place, redrawing the field with its latest settings.
There is a corresponding `doc.delay` flag if changes are being made to many fields at once.

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

**Example**
```javascript
// 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;
  f.delay = false; // force the changes now
```

### display

Controls whether the field is hidden or visible on screen and in print. Values are:

<table>
<thead>
<tr>
<th>Effect</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field is visible on screen and in print</td>
<td>display.visible</td>
</tr>
<tr>
<td>Field is hidden on screen and in print</td>
<td>display.hidden</td>
</tr>
<tr>
<td>Field is visible on screen but does not print</td>
<td>display.noPrint</td>
</tr>
<tr>
<td>Field is hidden on screen but prints</td>
<td>display.noView</td>
</tr>
</tbody>
</table>

This property supersedes the older `hidden` and `print` properties.

**Type:** Integer  **Access:** R/W  **Fields:** all.

**Example**
```javascript
// 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

Returns the **Doc Object** of the document to which the field belongs.

**Type:** object  **Access:** R/W  **Fields:** all.
**editable**

Controls whether a *combobox* is editable. When `true`, the user can type in a selection. When `false`, the user must choose one of the provided selections.

*Type: Boolean  Access: R/W  Fields: combobox*

**Example**

```javascript
var f = this.getField("myComboBox");
f.editable = true;
```

**exportValues**

The array of export values defined for the field. For radio button fields, this is necessary to make the field work properly as a group with the one button checked at any given time giving its value to the field as a whole. For checkbox fields, unless an export value is specified, the default used when the field is checked is "Yes" (or the corresponding localized string). When it is unchecked, its value is "Off" (this is also true for a radio button field when none of its buttons are checked). This property contains an array of strings with as many elements as there are annotations in the field. The elements of the array are mapped to the individual annotations comprising the field in the order of creation (unaffected by tab-order).

*Type: Array  Access: R/W  Fields: checkbox, radiobutton*

**Example**

```javascript
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**

When `true`, sets the file-select flag in the Options tab of the *text* field ("Field is Used for File Selection"). This indicates that the value of the field represents a pathname of a file whose contents may be submitted with the form.
The pathname may be entered directly into the field by the user, or the user can browse for the file. (See the `browseForFileToSubmit`.)

**NOTE:** The file select flag is mutually exclusive with the `multiline`, `charLimit`, `password`, and `defaultValue`. Also, on the Macintosh platform, when setting the file select flag, the field gets treated as read-only; hence, the user must browse for the file to enter into the field. (See the `browseForFileToSubmit`.)

**NOTE:** (Security): This property can only be set during batch, menu, or console events. See the `Event Object` for a discussion of Acrobat JavaScript events.

*Type: Boolean  Access: R/W  Fields: `text`.*

### fillColor

[Color swatch]

Specifies the background color for a field. The background color is used to fill the rectangle of the field. Values are defined by using `transparent`, `gray`, `RGB` or `CMYK` color. See Color Arrays for information on defining color arrays and how values are used with this property.

In older versions of this specification, this property was named `bgColor`. The use of `bgColor` is now discouraged but for backwards compatibility is still valid.

*Type: Array  Access: R/W  Fields: all.*

**Example**

```javascript
var f = this.getField("myField");
if (color.equal(f.fillColor, color.red))
  f.fillColor = color.blue;
else
  f.fillColor = color.yellow;
```

### hidden

[Hidden/Visible]

Controls whether the field is hidden or visible to the user. If the value is `false` the field is visible, `true` the field is invisible. The default value for `hidden` is `false`.

See also the `display` which supersedes this property in later versions.

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

**Example**

```javascript
var f = this.getField("myField");
f.hidden = true; // Set the field to hidden
```
**highlight**

Defines how a button reacts when a user clicks it. The four highlight modes supported are:

- **none**: No visual indication that the button has been clicked.
- **invert**: Click causes the region encompassing the button's rectangle to invert momentarily.
- **push**: Click displays the down face for the button (if any) momentarily.
- **outline**: Click causes the border of the rectangle to invert momentarily.

The convenience **highlight** object defines each state, as follows:

<table>
<thead>
<tr>
<th>Type</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>highlight.n</td>
</tr>
<tr>
<td>invert</td>
<td>highlight.i</td>
</tr>
<tr>
<td>push</td>
<td>highlight.p</td>
</tr>
<tr>
<td>outline</td>
<td>highlight.o</td>
</tr>
</tbody>
</table>

Type: String  
Access: R/W  
Fields: **button**

**Example**

The following example sets the **highlight** property of a button to "invert".

```javascript
// set the highlight mode on button to invert
var f = this.getField("myButton");
f.highlight = highlight.i;
```

**lineWidth**

Specifies the thickness of the border when stroking the perimeter of a field's rectangle. If the stroke color is transparent, this parameter has no effect except in the case of a beveled border. Values are:

- 0: none
- 1: thin
- 2: medium
- 3: thick

In older versions of this specification, this property was **borderWidth**. The use of **borderWidth** is now discouraged but for backwards compatibility is still valid.
Field Properties

Type: Integer  Access: R/W  Fields: all.

Example
// Change the border width of the Text Box to medium thickness
  f.lineWidth = 2

The default value for `lineWidth` is 1 (thin). Any integer value can be used; however, values beyond 5 may distort the field’s appearance.

**multiline**

Controls how the text is wrapped within the field. When `false`, the default, the text field can be a single line only. When `true`, multiple lines are allowed and wrap to field boundaries.

Type: Boolean  Access: R/W  Fields: text.

Example
See the Example 1 following `doc.getField`.

**multipleSelection**

If `true`, indicates that a listbox allows multiple selection of the items. See also `type`, `value`, and `currentValueIndices`.

Type: Boolean  Access: R/W  Fields: listbox

**name**

Allows you to access the fully qualified field name of the field as a string object. Beginning with Acrobat 6.0, if the Field Object represents one individual widget, then the returned name includes an appended "." followed by the widget index.

Type: String  Access: R  Fields: all.

Example
```javascript
var f = this.getField("myField");

// displays "myField" in console window
console.println(f.name);
```
numItems

The number of items in a combobox or listbox.

Type: Integer  Access: R  Fields: combobox, listbox

Example

```javascript
var f = this.getField("myList");
console.println("There are " + f.numItems + " in this listbox");
```

Face names and values of a combobox or listbox can be accessed through the `getItemAt` method. See that method for an additional example of `numItems`.

page

Returns the page number or an array of page numbers of a field. If the field has only one appearance in the document, the `page` property will return an integer representing the (0 based) page number of the page on which the field appears. If the field has multiple appearances, it will return an array of integers, each member of which is a (0 based) page number of an appearance of the field. The order in which the page numbers appear in the array is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). If an appearance of the field is on a hidden template page, `page` returns a value of -1 for that appearance.

Type: Integer | Array  Access: R  Fields: all.

Example

```javascript
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

Causes the field to display asterisks for the data entered into the field. Upon submission, the actual data entered is sent. Fields that have the password attribute set will not have the data in the field saved when the document is saved to disk.

Type: Boolean  Access: R/W  Fields: text.
**print**

Determines whether a given field prints or not. Set the `print` property to `true` to allow the field to appear when the user prints the document, set it to `false` to prevent printing. This property can be used to hide control buttons and other fields that are not useful on the printed page.

This property has been superseded by the `display` and its use is discourage.

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

**Example**

```javascript
var f = this.getField("myField");
f.print = false;
```

**radiosInUnison**

When `false`, even if a group of radio buttons have the same name and export value, they behave in a mutually exclusive fashion, like HTML radio buttons. The default for new radio buttons is `false`.

When `true`, if a group of radio buttons have the same name and export value, they turn on and off in unison, as in Acrobat 4.

*Type: Boolean*  
*Access: R/W*  
*Fields: radiobutton.*

**readonly**

Sets or gets the read-only characteristic of a field. If a field is read-only, the user can see the field but cannot change it.

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

**rect**

Sets or gets an array of four numbers in Rotated User Space that specifies the size and placement of the form field. These four numbers are the coordinates of the bounding
rectangle and are listed in the following order: upper-left x, upper-left y, lower-right x and lower-right y.

**NOTE:** Annot Object also has a `rect`, but: 1) the coordinates are not in Rotated User Space, and 2) they are in different order than in `field.rect`.

**Example 1**

Lay out a 2-inch-wide text field just to the right of the field "myText".

```javascript
var f = this.getField("myText"); // get the field object
var myRect = f.rect;  // and get it's rectangle

// make needed coordinate adjustments for new field
myRect[0] = f.rect[2]; // the ulx for new = lrx for old
myRect[2] += 2 * 72;  // move over two inches for lry
f = this.addField("myNextText", "text", this.pageNum, myRect);
f.strokeColor = color.black;
```

**Example 2**

Move a button field that already exists over 10 points to the right.

```javascript
var b = this.getField("myButton");
var aRect = b.rect; // make a copy of b.rect
aRect[0] += 10;  // increment first x-coordinate by 10
aRect[2] += 10; // increment second x-coordinate by 10
b.rect = aRect; // update the value of b.rect
```

### required

Sets or gets the *required* characteristic of a field. When `true`, the field’s value must be non-null when the user clicks a submit button that causes the value of the field to be posted. If the field value is `null`, the user receives a warning message and the submit does not occur.

**Type:** Boolean  **Access:** R/W  **Fields:** all except button

**Example**

```javascript
var f = this.getField("myField");
f.required = true;
```
richText

Get and sets the rich text property of the text field. If `true`, the field will allow rich text formatting. The default is `false`.

*Type:* Boolean  *Access:* R/W  *Fields:* `text`

Related objects and properties are the `Span Object`, `field.richValue`, `field.defaultStyle`, `event.richValue`, `event.richChange`, `event.richChangeEx`, and `annot.richContents`.

Example 1

Get a field object, and set it for rich text formatting.

```javascript
var f = this.getField("Text1");
f.richText = true;
```

Example 2

Count the number of rich text fields in the document.

```javascript
var count = 0;
for ( var i = 0; i < this.numFields; i++)
{
  var fname = this.getNthFieldName(i);
  var f = this.getField(fname);
  if ( f.type == "text" && f.richText ) count++
}
console.println("There are a total of "+ count + " rich text fields.");
```

richValue

This property gets the text contents and formatting of a rich text field. For field types other than rich text this property is undefined. The rich text contents are represented as an array of `Span Objects` containing the text contents and formatting of the field.

*Type:* Array of `Span Objects`  *Access:* R/W  *Fields:* `richText`

Related objects and properties are the `Span Object`, `field.richText`, `field.defaultStyle`, `event.richValue`, `event.richChange`, `event.richChangeEx`, and `annot.richContents`.

Example 1

This example turns all bold text into red underlined text.

```javascript
var f = this.getField("Text1");
var spans = f.richValue;
```
for ( var i = 0; i < spans.length; i++ )
{
    if( spans[i].fontWeight >= 700 )
    {
        spans[i].textColor = color.red;
        spans[i].underline = true;
    }
}

f.richValue = spans;

Example 2

This example creates a text field, marks it for rich text formatting, and inserts rich text.

```javascript
var myDoc = app.newDoc(); // create a blank doc
var Bbox = myDoc.getPageBox("Crop"); // get crop box
var inch = 72;

// add a text field at the top of the document
var f = myDoc.addField("Text1", "text", 0,
[72, Bbox[1]-inch, Bbox[2]-inch, Bbox[1]-2*inch ]
);
// add some attributes to this text field
f.strokeColor = color.black;
f.richText = true; // rich text
f.multiline = true; // multiline

// now build up an array of Span Objects
var spans = new Array();
spans[0] = new Object();
spans[0].text = "Attention:\r";
spans[0].textColor = color.blue;
spans[0].textSize = 18;

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

spans[2] = new Object();
spans[2].text = "will soon be here!";
spans[2].textColor = color.green;
spans[2].fontStyle = "italic";
spans[2].underline = true;
spans[2].alignment = "right";

// now give the rich field a rich value
f.richValue = spans;
```
rotation

Determines the rotation of a widget in 90 degree counter-clockwise increments. Valid values are 0, 90, 180, 270.

*Type: Integer  Access: R/W  Fields: all.*

strokeColor

Specifies the stroke color for a field which is used to stroke the rectangle of the field with a line as large as the line width. Values are defined by using transparent, gray, RGB or CMYK color. See Color Arrays for information on defining color arrays and how values are used with this property.

In older versions of this specification, this property was **borderColor**. The use of **borderColor** is now discouraged but for backwards compatibility is still valid.

*Type: Array  Access: R/W  Fields: all.*

style

Allows the user to set the glyph style of a checkbox or radiobutton. The glyph style is the graphic used to indicate that the item has been selected.

The style values are associated with keywords as follows:

<table>
<thead>
<tr>
<th>Style</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>check</td>
<td>style.ch</td>
</tr>
<tr>
<td>cross</td>
<td>style.cr</td>
</tr>
<tr>
<td>diamond</td>
<td>style.di</td>
</tr>
<tr>
<td>circle</td>
<td>style.ci</td>
</tr>
<tr>
<td>star</td>
<td>style.st</td>
</tr>
<tr>
<td>square</td>
<td>style.sq</td>
</tr>
</tbody>
</table>

*Type: String  Access: R/W  Fields: checkbox, radiobutton*
Example

The following example illustrates the use of this property and the style object:

```javascript
var f = this.getField("myCheckbox");
f.style = style.ci;
```

**submitName**

If nonempty, used during form submission instead of *name*. Only applicable if submitting in HTML format (that is, URLencoded).

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

**textColor**

Determines the foreground color of a field. It represents the text color for *text*, *button*, or *listbox* fields and the check color for *checkbox* or *radio button* fields. Values are defined the same as the *fillColor*. See *Color Arrays* for information on defining color arrays and how values are set and used with this property.

In older versions of this specification, this property was *fgColor*. The use of *fgColor* is now discouraged but for backwards compatibility is still valid.

*NOTE:* An exception is thrown if a transparent color space is used to set *textColor*.

*Type: Array*  
*Access: R/W*  
*Fields: all.*

**Example**

```javascript
var f = this.getField("myField");
f.textColor = color.red;
```

**textFont**

Determines the font that is used when laying out text in a *text field*, *combobox*, *listbox* or *button*. Valid fonts are defined as properties of the *font* object as follows:

<table>
<thead>
<tr>
<th>Text Font</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Times-Roman</td>
<td>font.Times</td>
</tr>
</tbody>
</table>
Beginning with Acrobat 5.0, an arbitrary font can be used when laying out a text field, combobox, listbox or button by setting the value of textFont to a string that represents the PostScript name of the font.

**NOTE:** Use of arbitrary fonts as opposed to those listed in the font object creates compatibility problems with older versions of the Viewer.

<table>
<thead>
<tr>
<th>Text Font</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Times-Bold</td>
<td>font.TimesB</td>
</tr>
<tr>
<td>Times-Italic</td>
<td>font.TimesI</td>
</tr>
<tr>
<td>Times-BoldItalic</td>
<td>font.TimesBI</td>
</tr>
<tr>
<td>Helvetica</td>
<td>font.Helv</td>
</tr>
<tr>
<td>Helvetica-Bold</td>
<td>font.HelvB</td>
</tr>
<tr>
<td>Helvetica-Oblique</td>
<td>font.HelvI</td>
</tr>
<tr>
<td>Helvetica-BoldOblique</td>
<td>font.HelvBI</td>
</tr>
<tr>
<td>Courier</td>
<td>font.Cour</td>
</tr>
<tr>
<td>Courier-Bold</td>
<td>font.CourB</td>
</tr>
<tr>
<td>Courier-Oblique</td>
<td>font.CourI</td>
</tr>
<tr>
<td>Courier-BoldOblique</td>
<td>font.CourBI</td>
</tr>
<tr>
<td>Symbol</td>
<td>font.Symbol</td>
</tr>
<tr>
<td>ZapfDingbats</td>
<td>font.ZapfD</td>
</tr>
</tbody>
</table>

**Example**

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

```javascript
// set the font of "myField" to Helvetica
var f = this.getField("myField");
f.textFont = font.Helv;
```

**Example (Acrobat 5.0)**

```javascript
// set the font of "myField" to Viva-Regular
var f = this.getField("myField");
f.textFont = "Viva-Regular";
```
**textSize**

Controls the text size (in points) to be used in all controls. In `checkbox` and `radiobutton` fields, the text size determines the size of the check. Valid text sizes range from 0 to 32767 inclusive. A text size of zero means to use the largest point size that will allow all text data to fit in the field’s rectangle.

*Type:* Number  
*Access:* R/W  
*Fields:* all.

**Example**

```javascript
// set the text size of myField to 28 point
this.getField("myField").textSize = 28;
```

**type**

Returns the type of the field as a string. Valid types are:

- button
- checkbox
- combobox
- listbox
- radiobutton
- signature
- text

*Type:* String  
*Access:* R  
*Fields:* all.

**userName**

Gets or sets the user name (short description string) of the field. The user name is intended to be used as tooltip text whenever the mouse cursor enters a field. It can also be used as a user-friendly name when generating error messages instead of the field name.

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

**value**

Gets the value of the field data that the user has entered. Depending on the `type` of the field, may be a String, Date, or Number. Typically, the `value` is used to create calculated fields.
Beginning with Acrobat 6.0, if a field contains rich text formatting, modifying this property will discard the formatting and regenerate the field value and appearance using the `defaultStyle` and plain text value. To modify the field value and maintain formatting use the `richValue` property.

**NOTES:** For `signature` fields, if the field has been signed then a non-null string is returned as the value.

For Acrobat 5.0 or later, if the field is a `listbox` that accepts multiple selection (see `multipleSelection`), you can pass an array to set the `value` of the field, and `value` returns an array for a `listbox` with multiple values currently selected.

The `currentValueIndices` of a listbox that has multiple selections is the preferred and most efficient way to get and set the value of this type of field.

See also `valueAsString`, and the Event Object type.

**Type:** various  
**Access:** R/W  
**Fields:** all except button

### Example

In this example, the value of the field being calculated is set to the sum of the "oil" and "filter" fields and multiplied by the state sales tax.

```javascript
var oil = this.getField("Oil");
var filter = this.getField("Filter");
event.value = (oil.value + filter.value) * 1.0825;
```

---

**valueAsString**

5.0

Returns the value of a field as a JavaScript string.

This differs from `value`, which attempts to convert the contents of a field contents to an accepted format. For example, for a field with a value of "020", `value` returns the integer 20, while `valueAsString` returns the string "020".

**Type:** String  
**Access:** R  
**Fields:** all except button

---

**Field Methods**

### browseForFileToSubmit

5.0

When invoked on a `text` field for which the `fileSelect` flag is set (checked), opens a standard file-selection dialog. The path entered through the dialog is automatically assigned as the value of the text field.
If invoked on a text field in which the `fileSelect` flag is clear (unchecked), an exception is thrown.

**Parameter**

None

**Returns**

Nothing

**Example**

The following code references a text field with the file select flag checked. This is a mouse up action of a button field.

```javascript
var f = this.getField("resumeField");
f.browseForFileToSubmit();
```

### buttonGetCaption

**5.0**

Gets the caption associated with a **button**.

**Parameter**

- `nFace` (optional) If specified, gets a caption of the given type:
  - 0: (default) normal caption
  - 1: down caption
  - 2: rollover caption

**Returns**

The caption string associated with the button.

**Example**

This example places pointing arrows to the left and right of the caption on a button field with icon and text.

```javascript
// a mouse enter event
event.target.buttonSetCaption("=> "+ event.target.buttonGetCaption() +" <=");

// a mouse exit event
var str = event.target.buttonGetCaption();
str = str.replace(/=> | <=/g, "");
event.target.buttonSetCaption(str);
```

The same effect can be created by having the same icon for rollover, and have the same text, with the arrows inserted, for the rollover caption. This approach would be slower and
cause the icon to flicker. The above code gives a very fast and smooth rollover effect because only the caption is changed, not the icon.

**buttonGetIcon**

**5.0**

Gets the *Icon Generic Object* of a specified type associated with a button.

**Parameter**

<table>
<thead>
<tr>
<th><strong>nFace</strong> (optional)</th>
<th>If specified, gets an icon of the given type:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0: (default) normal icon</td>
<td></td>
</tr>
<tr>
<td>1: down icon</td>
<td></td>
</tr>
<tr>
<td>2: rollover icon</td>
<td></td>
</tr>
</tbody>
</table>

**Returns**

The *Icon Generic Object*.

**Example**

```javascript
// Swap two button icons.
var f = this.getField("Button1");
var g = this.getField("Button2");
var temp = f.buttonGetIcon();
f.buttonSetIcon(g.buttonGetIcon());
g.buttonSetIcon(temp);
```

See also **buttonSetIcon** and **buttonImportIcon**.

**buttonImportIcon**

**4.0**

Imports the appearance of a button from another PDF file. If neither of the optional parameters are passed, the method prompts the user to select a file available on the
Field Methods

Parameter

**cPath**

(optional, version 5.0) The device-independent pathname for the file. See Section 3.10.1 of the PDF Reference for a description of the device-independent pathname format. Beginning with version 6.0, Acrobat will first attempt to open *cPath* as a PDF. On failure, Acrobat will try to convert *cPath* to PDF from one of the known graphics formats (BMP, GIF, JPEG, PCX, PNG, TIFF) and then import the converted file as a button icon.

**nPage**

(optional, version 5.0) The 0-based page number from the file to turn into an icon. The default is 0.

Returns

An integer, as follows:

1: The user cancelled the dialog
0: No error
-1: The selected file couldn’t be opened
-2: The selected page was invalid

Example (Acrobat 5.0)

It is assumed that we are connected to an employee information database. We communicate with the database using the ADBC Object and related objects. An employee's record is requested and three columns are utilized, **FirstName**, **SecondName** and **Picture**. The **Picture** column, from the database, contains a device-independent path to the employee's picture, stored in PDF format. The script might look like this:

```javascript
var f = this.getField("myPicture");
f.buttonSetCaption(row.FirstName.value + " " + row.LastName.value);
if (f.buttonImportIcon(row.Picture.value) != 0)
  f.buttonImportIcon("/F/employee/pdfs/NoPicture.pdf");
```

The button field "myPicture" has been set to display both icon and caption. The employee's first and last names are concatenated to form the caption for the picture. Note that if there is an error in retrieving the icon, a substitute icon could be imported.

**buttonSetCaption**

Sets the caption associated with a button. See **buttonAlignX**, **buttonAlignY**, and so on for details on how the icon and caption are placed on the button face.
### Field Methods

#### buttonSetCaption

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cCaption</td>
<td>The caption associated with the button.</td>
</tr>
<tr>
<td>nFace</td>
<td>(optional) If specified, sets a caption of the given type:</td>
</tr>
<tr>
<td></td>
<td>0: (default) normal caption</td>
</tr>
<tr>
<td></td>
<td>1: down caption</td>
</tr>
<tr>
<td></td>
<td>2: rollover caption</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**Example**

```javascript
var f = this.getField("myButton");
f.buttonSetCaption("Hello");
```

#### buttonSetIcon

Sets the icon associated with a button. See `buttonScaleHow`, `buttonScaleWhen`, and so on for details on how the icon is rendered on the button face. See also `buttonGetIcon`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oIcon</td>
<td>The Icon Generic Object associated with the button.</td>
</tr>
<tr>
<td>nFace</td>
<td>(optional) If specified, sets an icon of the given type:</td>
</tr>
<tr>
<td></td>
<td>0: (default) normal icon</td>
</tr>
<tr>
<td></td>
<td>1: down icon</td>
</tr>
<tr>
<td></td>
<td>2: rollover icon</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**Example**

This example takes every named icon in the document and creates a listbox using the names. Selecting an item in the listbox sets the icon with that name as the button face of the field "myPictures". What follows is the mouse up action of the button field "myButton".

```javascript
var f = this.getField("myButton");
var aRect = f.rect;
aRect[0] = f.rect[2]; // place listbox relative to the
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) {
    var f = this.getField("myPictures");
    var i = this.getIcon(event.change);
    f.buttonSetIcon(i);
  }');

The named icons themselves can be imported into the document through an interactive
scheme, such as the example given in addIcon or through a batch sequence.
See also buttonGetCaption for a more extensive example.

checkThisBox

Checks or unchecks the specified widget. Only checkboxes can be unchecked. A
radiobutton cannot be unchecked using this method, but if its default state is
unchecked (see defaultIsChecked) it can be reset to the unchecked state using
doc.resetForm.

**NOTE:** For a set of radio buttons that do not have duplicate export values, you can set
the value to the export value of the individual widget that should be checked (or
pass an empty string if none should be).

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nWidget</td>
<td>The 0-based index of an individual checkbox or radiobutton widget for this field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). Every entry in the Fields panel has a suffix giving this index; for example, MyField #0.</td>
</tr>
<tr>
<td>bCheckIt</td>
<td>(optional) Whether the widget should be checked. The default is true.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**Example**

```javascript
// check the box "ChkBox"
var f = this.getField("ChkBox");
```
f.checkThisBox(0,true);

clearItems

Clears all the values in a **listbox** or **combobox**. Related methods and properties include `numItems`, `getItemAt`, `deleteItemAt`, `currentValueIndices`, `insertItemAt`, and `setItems`.

**Parameters**
None

**Returns**
Nothing

**Example**
Clear the field “myList.”

```javascript
var f = this.getField("myList");
f.clearItems();
```

defaultIsChecked

Sets the specified widget to be checked or unchecked by default.

**NOTE:** For a set of radio buttons that do not have duplicate export values, you can set the `defaultValue` to the export value of the individual widget that should be checked by default (or pass an empty string if none should be).
 Parameters

<table>
<thead>
<tr>
<th>nWidget</th>
<th>The 0-based index of an individual <strong>Radiobutton</strong> or <strong>Checkbox</strong> widget for this field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). Every entry in the <strong>Fields</strong> panel has a suffix giving this index (for example, MyField #0).</th>
</tr>
</thead>
<tbody>
<tr>
<td>bIsDefaultChecked</td>
<td>(optional) When <strong>true</strong> (the default) the widget should be checked by default (for example, when the field gets reset). When <strong>false</strong>, it should be unchecked by default.</td>
</tr>
</tbody>
</table>

 Returns

**true** on success.

 Example

Change the default of "ChkBox" to checked.

```javascript
var f = this.getField("ChkBox");
f.defaultIsChecked(0,true);
this.resetForm(['ChkBox']);
```

**deleteItemAt**

Deletes an item in a **ComboBox** or a **ListBox**.

For a **ListBox**, if the current selection is deleted the field no longer has a current selection. Having no current selection can lead to unexpected behavior by this method if it is again invoked without parameters on this same field; there is no current selection to delete. It is important, therefore, to make a new selection so that this method will behave as documented. A new selection can be made by using the **currentValueIndices**.

 Parameters

| nIdx | (optional) The 0-based index of the item in the list to delete. If not specified, the currently selected item is deleted. |

 Returns

Nothing

 Example

```javascript
var a = this.getField("MyListBox");
a.deleteItemAt(); // delete current item, and...
a.currentValueIndices = 0; // select top item in list
```
getArray
Performs field calculations in tables where a parent field value is the sum of all of its children.

**Parameters**
None

**Returns**
An array of terminal child fields (that is, fields that can have a value) for a parent field.

**Example**
```javascript
// 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
Gets the internal value of an item in a **combobox** or a **listbox**.
The number of items in a list can be obtained from `field.numItems`. See also `insertItemAt`, `deleteItemAt`, `clearItems`, `currentValueIndices` and `setItems`.

**Parameters**

<table>
<thead>
<tr>
<th><strong>nIdx</strong></th>
<th>The 0-based index of the item in the list to obtain, or -1 for the last item in the list.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bExportValue</strong></td>
<td>(optional, version 5.0) Whether to return an export value.</td>
</tr>
</tbody>
</table>

- When `true`, (the default) if the requested item has an export value, returns the export value. If there is no export value, returns the item name.
- When `false`, the method returns the item name.

**Returns**
The export value or name of the specified item.

**Example**

In the two examples that follow, assume there are three items on "myList": "First", with an export value of 1; "Second", with an export value of 2; and "Third" with no export value.

```javascript
// returns value of first item in list, which is 1
var f = this.getField("myList");
```
var v = f.getItemAt(0);

The following example illustrates the use of the second optional parameter.

    for (var i=0; i < f.numItems; i++)
        console.println(f.getItemAt(i,true) + "": " +
            f.getItemAt(i,false));

The output to the console reads:

    1: First
    2: Second
    Third: Third

Thus, by putting the second parameter to false the item name (face value) can be obtained, even when there is an export value.

### getLock

**6.0**

Gets a Lock Object, a generic object that contains the lock properties of a signature field.

See also `setLock` of the Field Object.

**Parameters**

None

**Returns**

The Lock Object for the field.

### insertItemAt

Inserts a new item into a combobox or a listbox.

Related methods and properties include `numItems`, `getItemAt`, `deleteItemAt`, `clearItems`, `currentValueIndices` and `setItems`. 
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The item name that will appear in the form.</td>
</tr>
<tr>
<td>cExport</td>
<td>(optional) The export value of the field when this item is selected. If not provided, the <code>cName</code> is used as the export value.</td>
</tr>
<tr>
<td>nIdx</td>
<td>(optional) The index in the list at which to insert the item. If 0 (the default), the new item is inserted at the top of the list. If –1, the new item is inserted at the end of the list.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

```javascript
var l = this.getField("myList");
l.insertItemAt("sam", "s", 0); /* inserts sam to top of list l */
```

`isBoxChecked` 5.0

Determines whether the specified widget is checked.

**NOTE:** For a set of `radiobuttons` that do not have duplicate export values, you can get the `value`, which is equal to the export value of the individual widget that is currently checked (or returns an empty string, if none is).

Parameters

| nWidget | The 0-based index of an individual `radiobutton` or `checkbox` widget for this field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). Every entry in the `Fields` panel has a suffix giving this index, for example `MyField #0`. |

Returns

`true` if the specified widget is currently checked, `false` otherwise.

Example

```javascript
var f = this.getField("ChkBox");
if(f.isBoxChecked(0))
    app.alert("The Box is Checked");
else
    app.alert("The Box is not Checked");
```
**isDefaultChecked**

5.0

Determines whether the specified widget is checked by default (for example, when the field gets reset).

**NOTE:** For a set of radio buttons that do not have duplicate export values, you can get the `defaultValue`, which is equal to the export value of the individual widget that is checked by default (or returns an empty string, if none is).

**Parameters**

<table>
<thead>
<tr>
<th>nWidget</th>
</tr>
</thead>
<tbody>
<tr>
<td>The 0-based index of an individual radio button or checkbox widget for this field. The index is determined by the order in which the individual widgets of this field were created (and is unaffected by tab-order). Every entry in the <strong>Fields</strong> panel has a suffix giving this index, for example MyField #0.</td>
</tr>
</tbody>
</table>

**Returns**

*true* if the specified widget is checked by default, *false* otherwise.

**Example**

```javascript
var f = this.getField("ChkBx");
if (f.isDefaultChecked(0))
    app.alert("The Default: Checked");
else
    app.alert("The Default: Unchecked");
```

**setAction**

5.0 [ ] [ ]

Sets the JavaScript action of the field for a given trigger. See also *bookmark.setAction*, *doc.setAction*, *doc.addScript*, *doc.setPageAction*. 
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cTrigger</td>
<td>A string that sets the trigger for the action. Values are: MouseUp, MouseDown, MouseEnter, MouseExit, OnFocus, OnBlur, Keystroke, Validate, Calculate, Format. For a listbox, use the Keystroke trigger for the Selection Change event.</td>
</tr>
<tr>
<td>cScript</td>
<td>The JavaScript code to be executed when the trigger is activated.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

```javascript
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;
```

See also buttonSetIcon.

setFocus

4.05

Sets the keyboard focus to this field. This can involve changing the page that the user is currently on or causing the view to scroll to a new position in the document. This method automatically brings the document that the field resides in to the front, if it is not already there.

See also the bringToFront.

Parameters

None

Returns

Nothing
Example

Search for a certain open doc, then focus in on the field of interest. This will only work on documents with disclosed set to true

```javascript
var d = app.activeDocs;
for (var i = 0; i < d.length; i++) {
    if (d[i].info.Title == "Response Document") {
        d[i].getField("name").value="Enter your name here: ";
        // also brings the doc to front.
        d[i].getField("name").setFocus();
        break;
    }
}
```

setItems

Sets the list of items for a combobox or a listbox.

Related methods and properties include numItems, getItemAt, deleteItemAt, currentValueIndices and clearItems.

Parameters

<table>
<thead>
<tr>
<th>oArray</th>
<th>An array in which each element is either an object convertible to a string or another array.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>● For an element that can be converted to a string, the user and export values for the list item are equal to the string.</td>
</tr>
<tr>
<td></td>
<td>● For an element that is an array, the array must have two sub-elements convertible to strings, where the first is the user value, and the second is the export value.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Examples

```javascript
var l = this.getField("ListBox");
l.setItems(["One", "Two", "Three"]);

var c = this.getField("StateBox");
c.setItems(["California", "CA"],["Massachusetts", "MA"],["Arizona", "AZ"]);

var c = this.getField("NumberBox");
c.setItems(["1", 2, 3, ["PI", Math.PI]]);
```
**setLock**

Controls which fields are to be locked when a signature is applied to this `signature` field. Once the fields are locked no modifications can be done to the fields. When the signature is cleared, all the fields that were locked down are unlocked. The property settings can be obtained using `getLock`.

**NOTE:** (Security odynamically): The method can be executed during a batch, application initialization, console, or menu events. Not allowed in the Adobe Reader.

**NOTE:** This method cannot be applied to a field that is in a document that is already signed.

**Parameters**

- `oLock`: A Lock Object containing the lock properties.

**Returns**

- `true` if successful, `false` otherwise, or can throw an exception.

**Lock Object**

A generic JS object containing lock properties. This object is passed to `field.setLock` and returned by `field.getLock` for a `signature` field. It contains the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>String</td>
<td>R/W</td>
<td>The language independent name of the action. Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>All</strong>: All fields in the document are to be locked.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Include</strong>: Only the fields specified in fields are to be locked.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Exclude</strong>: All fields except those specified in fields are to be locked.</td>
</tr>
<tr>
<td>fields</td>
<td>Array of Strings</td>
<td>R/W</td>
<td>An array of strings containing the field names. Required if the value of action is <code>Include</code> or <code>Exclude</code>.</td>
</tr>
</tbody>
</table>

**signatureGetSeedValue**

Returns a SeedValue Generic Object that contains the seed value properties of a signature field. Seed values are used to control properties of the signature, including the signature appearance, reasons for signing, and the person.

See `signatureSetSeedValue`.
Parameters

None

Returns

A SeedValue Generic Object.

Example

The following illustrates accessing the seed value for a signature field.

```javascript
var f = this.getField( "sig0" );
var seedValue = f.signatureGetSeedValue();
// displays the seed value filter and flags
console.println( "Filter name:" + seedValue.filter);
console.println( "Flags:" + seedValue.flags);
// displays the certificate seed value constraints
var certSpec = seedValue.certspec;
console.println( "Issuer:" + certspec.issuer);
```

signatureInfo

Returns a SignatureInfo Object that contains the properties of the signature. The object is a snapshot of the signature that is taken at the time that this method is called. A security handler may specify additional properties that are specific to the security handler.

NOTE: (Security 5): There are no restrictions on when this method can be called, however, the specified security handler may not always be available; see `security.getHandler` for details.

NOTE: Some properties of a signature handler, for example, certificates (a property of the SignatureInfo Object), may return a null value until the signature is validated. Therefore, signatureInfo should be called again after `signatureValidate`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oSig</td>
<td>(optional) The SecurityHandler Object to use to retrieve the signature properties. If not specified, the security handler is determined by user preferences: it is usually the handler that was used to create the signature.</td>
</tr>
</tbody>
</table>

Returns

A SignatureInfo Object that contains the properties of the signature. This type of object is also used when signing signature fields, signing FDF objects, or with the `FDF.signatureValidate` method.
Example

The following illustrates how to access signature info properties.

```javascript
// get all info
var f = getField("Signature1");
f.signatureValidate();
var s = f.signatureInfo();
console.println("Signature Attributes:");
for(i in s) console.println(i + " = " + s[i]);

// get particular info
var f = this.getField("Signature1"); // uses the ppklite sig handler
var Info = f.signatureInfo();
// some standard signatureInfo properties
console.println("name = " + Info.name);
console.println("reason = " + Info.reason);
console.println("date = " + Info.date);

// additional signatureInfo properties from PPKLite
console.println("contact info = " + Info.contactInfo);

// get the certificate; first (and only) one
var certificate = Info.certificates[0];

// common name of the signer
console.println("subjectCN = " + certificate.subjectCN);
console.println("serialNumber = " + certificate.serialNumber);

// Display some information about this the distinguished name of signer
console.println("subjectDN.cn = " + certificate.subjectDN.cn);
console.println("subjectDN.o = " + certificate.subjectDN.o);
```

signatureSetSeedValue

6.0  D  S  X

Sets properties that are used when signing signature fields. The properties are stored in the signature field and are not altered when the field is signed, the signature is cleared, or when resetForm is called. Use signatureGetSeedValue to obtain the property settings.

**NOTE:** (Security): The method can be executed during a batch, application initialization, console, or menu events. Not allowed in the Adobe Reader.

**NOTE:** Seed values cannot be set for author signatures. Author signatures are signatures with a SignatureInfo Object mdp property value of allowNone, default, or defaultAndComments.
### Parameters

| oSigSeedValue | A SeedValue Generic Object containing the signature seed value properties. |

### Returns

Nothing

### SeedValue Generic Object

A generic JS object, passed to `field.signatureSetSeedValue` and returned by `field.signatureGetSeedValue`, which represents a signature seed value. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>String</td>
<td>R/W</td>
<td>The language independent name of the security handler to be used when signing.</td>
</tr>
<tr>
<td>subFilter</td>
<td>Array of Strings</td>
<td>R/W</td>
<td>An array of acceptable formats to use for the signature. Refer to the signature info object's subFilter property for a list of known formats.</td>
</tr>
<tr>
<td>version</td>
<td>Number</td>
<td>R/W</td>
<td>The minimum version of the signature format dictionary that is required when signing.</td>
</tr>
<tr>
<td>reasons</td>
<td>Array of Strings</td>
<td>R/W</td>
<td>A list of reasons that the user is allowed to use when signing.</td>
</tr>
<tr>
<td>certspec</td>
<td>object</td>
<td>R/W</td>
<td>A seed value CertificateSpecifier Generic Object.</td>
</tr>
<tr>
<td>flags</td>
<td>Number</td>
<td>R/W</td>
<td>Flags controlling which properties in this object are critical (1, required) and not critical (0, optional). The value should be set to the logical or of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1: if filter is critical,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2: if subFilter is critical,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4: if version is critical</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8: if reasons field is critical.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If this field is not present, interpretation of all attributes is optional.</td>
</tr>
</tbody>
</table>
CertificateSpecifier Generic Object

This generic JS object contains the certificate specifier properties of a signature seed value. Used in the certSpec property of the SeedValue Generic Object. This objects contains the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subject</td>
<td>Array of Certificate Object</td>
<td>R/W</td>
<td>Array of Certificate Objects that are acceptable for signing. <strong>NOTE:</strong> If specified, the signing certificate must be an exact match with one of the certificates in this array.</td>
</tr>
<tr>
<td>issuer</td>
<td>Array of Certificate Object</td>
<td>R/W</td>
<td>Array of Certificate Objects that are acceptable for signing. <strong>NOTE:</strong> If specified, the signing certificate must be issued by a certificate that is an exact match with one of the certificates in this array.</td>
</tr>
<tr>
<td>oid</td>
<td>Array of Strings</td>
<td>R/W</td>
<td>Array of strings that contain Policy OIDs that must be present in the signing certificate. This property is only applicable of the issuer property is present.</td>
</tr>
<tr>
<td>url</td>
<td>String</td>
<td>R/W</td>
<td>A URL that can be used to enroll for a new credential if a matching credential is not found.</td>
</tr>
<tr>
<td>flags</td>
<td>Number</td>
<td>R/W</td>
<td>Bit flags controlling which properties in this object are critical (1, required) and not critical (0, optional). The value should be set to the logical or of the following values: 1 if subject is critical, 2 if issuer is critical, 4 if oid is critical. If this field is not present, interpretation of all attributes is optional.</td>
</tr>
</tbody>
</table>

**Example 1**

Sets the signing handler as PPKMS and the format as "adbe.pkcs7.sha1".

```javascript
var f = this.getField( "sig0" );

f.signatureSetSeedValue( {
    filter: "Adobe.PPKMS",
    subFilter: ["adbe.pkcs7.sha1"],
    flags: 0x03
  } );
```

**Example 2**

Sets the signing handler as PPKLite and the issuer of the signer's certificate as caCert. Both are mandatory seed values and signing will fail if either of constraint is not met.

```javascript
var caCert = security.importFromFile("Certificate", "/C/CA.cer");

f.signatureSetSeedValue( {filter: "Adobe.PPKLite",
```
certspec: {
    issuer: [caCert],
    url: "http://www.ca.com/enroll.html",
    flags : 0x02 },
    flags: 0x01 });

signatureSign

Signs the field with the specified security handler. See also `security.getHandler` and `securityHandler.login`.

**NOTE:** (Security): This method can only be executed during batch, console, menu, or application initialization events. Not available in the Adobe Reader. See the Event Object for a discussion of Acrobat JavaScript events.

**NOTE:** Signature fields cannot be signed if they are already signed. Use `resetForm` to clear signature fields.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oSig</td>
<td>Specifies the SecurityHandler Object to be used to sign. Throws an exception if the specified handler does not support signing operations. Some security handlers require that the user be logged in before signing can occur. Some security handlers require that the user be logged in before signing can occur. If <code>oSig</code> is not specified then this method will select a handler based on user preferences or by prompting the user if <code>bUI</code> is <code>true</code>.</td>
</tr>
<tr>
<td>oInfo</td>
<td>(optional) A SignatureInfo Object specifying the writable properties of the signature. See also <code>signatureInfo</code>.</td>
</tr>
<tr>
<td>cDIPath</td>
<td>(optional) The device-independent path to the file to save to following the application of the signature. If not specified, the file is saved back to its original location.</td>
</tr>
<tr>
<td>bUI</td>
<td>(optional, version 6.0) Whether the security handler should show user interface when signing. If <code>true</code>, <code>oInfo</code> and <code>cDIPath</code> are used as default values in the signing dialogs. If <code>false</code> (the default), the signing occurs without any user interface.</td>
</tr>
</tbody>
</table>
cLegalAttest (optional, version 6.0) A string that can be provided when creating an author signature.

Author signatures are signatures where the mdp property of the SignatureInfo Object has a value other than allowAll. When creating an author signature, the document is scanned for legal warnings and these warnings are embedded in the document. A caller can determine what legal warnings are found by first calling doc.getLegalWarnings. If warnings are to be embedded an author may wish to provide an attestation as to why these warnings are being applied to a document.

**Returns**

true if the signature was applied successfully, false otherwise.

**Example 1**

The following example signs the "Signature" field with the PPKLite signature handler:

```javascript
var myEngine = security.getHandler( "Adobe.PPKLite" );
myEngine.login( "dps017", "/c/profile/dps.pfx" );
var f = this.getField("Signature");

// Sign the field
f.signatureSign( myEngine,
    { password: "dps017", // provide password
      location: "San Jose, CA", // ... see note below
      reason: "I am approving this document",
      contactInfo: "dpsmith@adobe.com",
      appearance: "Fancy" });
```

**Note:** In the above example, a password was provided. This may or may not have been necessary depending whether the Password Timeout had expired. The Password Timeout can be set programmatically by securityHandler.setPasswordTimeout.

**Example 2**

The following example illustrates signing an author signature field

```javascript
var myEngine = security.getHandler( "Adobe.PPKLite" );
myEngine.login( "dps017", "/c/profile/dps.pfx" );

var f = this.getField( "AuthorSigFieldName" );
var s = { reason: "I am the author of this document",
    mdp: "allowNone" };

f.signatureSign({
    oSig: myEngine,
    oInfo: s,
    bUI: false,
    cLegalAttest: "Fonts are not embedded to reduce file size"
});
```
signatureValidate

Validates and returns the validity status of the signature in a signature field. This routine can be computationally expensive and take a significant amount of time depending on the signature handler used to sign the signature.

**NOTE:** There are no restrictions on when this method can be called, however, the parameter oSig will not always be available; see security.getHandler for details.

**Parameters**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oSig</td>
<td>(optional) The security handler to be used to validate the signature. The value can be either a SecurityHandler Object or a SignatureParameters Generic Object. If this handler is not specified, the method uses the security handler returned by the signature's handlerName property.</td>
</tr>
<tr>
<td>bUI</td>
<td>(optional, version 6.0) When true, allows UI to be shown, if necessary, when validating the data file. UI may be used to select a validation handler if none is specified. The default is false.</td>
</tr>
</tbody>
</table>

**Returns**

Returns the validity status of the signature. Validity values are:

-1: Not a signature field

0: Signature is blank

1: Unknown status

2: Signature is invalid

3: Signature of document is valid, identity of signer could not be verified

4: Signature of document is valid and identity of signer is valid.

See the status and statusText properties of the SignatureInfo Object.

**SignatureParameters Generic Object**

A generic object with the following properties that specify security handlers to be used for validation by field.signatureValidate:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oSecHdlr</td>
<td>The security handler object to use to validate this signature</td>
</tr>
</tbody>
</table>
Example

```javascript
var f = this.getField("Signature1") // get signature field
var status = f.signatureValidate();
var sigInfo = f.signatureInfo();
if ( status < 3 )
    var msg = "Signature not valid! " + sigInfo.statusText;
else
    var msg = "Signature valid! " + sigInfo.statusText;
app.alert(msg);
```

**FullScreen Object**

The interface to fullscreen (presentation mode) preferences and properties. To acquire a `fullscreen` object, use `app.fs`.

### FullScreen Properties

**backgroundColor**

The background color of the screen in full screen mode. See Color Arrays for details.

*Type: Color Array  Access: R/W.*

**Example**

```javascript
app.fs.backgroundColor = color.ltGray;
```

**clickAdvances**

Whether a mouse click anywhere on the page will cause the viewer to advance one page.

*Type: Boolean  Access: R/W.*
cursor

Determines the behavior of the mouse pointer in full screen mode. The convenience cursor object defines all the valid cursor behaviors:

<table>
<thead>
<tr>
<th>Cursor Behavior</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always hidden</td>
<td>cursor.hidden</td>
</tr>
<tr>
<td>Hidden after delay</td>
<td>cursor.delay</td>
</tr>
<tr>
<td>Visible</td>
<td>cursor.visible</td>
</tr>
</tbody>
</table>

Type: Number Access: R/W.

Example

```javascript
app.fs.cursor = cursor.visible;
```

defaultTransition

The default transition to use when advancing pages in full screen mode. Use transitions to obtain list of valid transition names supported by the viewer.

No Transition is equivalent to app.fs.defaultTransition = "";

Type: Number Access: R/W.

Example

```javascript
Put document into presentation mode
app.fs.defaultTransition = "WipeDown";
app.fs.isFullScreen = true;
```

escapeExits

Whether the escape key can be used to exit full screen mode.

Type: Boolean Access: R/W.

isFullScreen

Puts the Acrobat viewer in fullscreen mode rather than regular viewing mode. This only works if there are documents open in the Acrobat viewer window.

**NOTE:** A PDF document being viewed from within a web browser cannot be put into fullscreen mode.

Type: Boolean Access: R/W.
Example

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

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

**loop**

Whether the document will loop around to the beginning of the document in response to a page advance (mouse click, keyboard, and/or timer generated) in full screen mode.

*Type: Boolean  Access: R/W.*

**timeDelay**

The default number of seconds before the page automatically advances in full screen mode. See `useTimer` to activate/deactivate automatic page turning.

*Type: Number  Access: R/W.*

Example

```javascript
app.fs.timeDelay = 5; // delay 5 seconds
app.fs.useTimer = true; // activate automatic page turning
app.fs.usePageTiming = true; // allow page override
app.fs.isFullScreen = true; // go into fullscreen
```

**transitions**

An array of strings representing valid transition names implemented in the viewer. **No Transition** is equivalent to setting `defaultTransition` to the empty string:

```javascript
app.fs.defaultTransition = "";
```

*Type: Array  Access: R.*

Example

This script produces a listing of the currently supported transition names.

```javascript
console.println("[" + app.fs.transitions + "]");
```

**usePageTiming**

Whether automatic page turning will respect the values specified for individual pages in full screen mode. Set transition properties of individual pages using `setPageTransitions`.
useTimer

Whether automatic page turning is enabled in full screen mode. Use `timeDelay` to set the default time interval before proceeding to the next page.

Type: Boolean  Access: R/W.

Global Object

This is a static JavaScript object that allows you to share data between documents and to have data be persistent across sessions. Such data is called persistent global data. Global data-sharing and notification across documents is done through a subscription mechanism, which allows you to monitor global data variables and report their value changes across documents.

Creating Global Properties

You can specify global data by adding properties to the global object. The property type can be a String, a Boolean, or a Number.

For example, to add a variable called "radius" and to allow all document scripts to have access to this variable, the script simply defines the property:

```javascript
  global.radius = 8;
```

The global variable "radius" is now known across documents throughout the current viewer session. Suppose two files, `A.pdf` and `B.pdf`, are open in the viewer, and the global declaration is made in `A.pdf`. From within either file (`A.pdf` or `B.pdf`) you can calculate the volume of a sphere using `global.radius`:

```javascript
  var V = (4/3) * Math.PI * Math.pow(global.radius, 3);
```

In either file, you obtain the same result, 2144.66058. If the value of `global.radius` changes and the script is executed again, the value of `V` changes accordingly.

Deleting Global Properties

To delete a variable or a property from the global object, use the `delete` operator to remove the defined property. For information on the reserved JavaScript keyword `delete`, see Core JavaScript 1.5 Documentation.

For example, to remove the `global.radius` property, call the following script:

```javascript
  delete global.radius
```
Global Methods

setPersistent

Controls whether a specified variable is persistent across invocations of Acrobat.

Persistent global data only applies to variables of type Boolean, Number, or String. Acrobat 6.0 places a 2-4k limit for the maximum size of the global persistent variables. Any data added to the string after this limit is dropped.

The global variables that are persistent are stored upon application exit in the glob.js file located in the user's folder for Folder Level JavaScripts, and re-loaded at application start. There is a 2-4k limit on the size of this file, for Acrobat 6.0 or later.

It is recommended that JavaScript developers building scripts for Acrobat, use a naming convention when specifying persistent global variables. For example, you could name all your variables "myCompany_name". This will prevent collisions with other persistent global variable names throughout the documents.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cVariable</td>
<td>The variable (global property) for which to set persistence.</td>
</tr>
<tr>
<td>bPersist</td>
<td>When true, the property will exist across Acrobat Viewer sessions. When false (the default) the property will be accessible across documents but not across the Acrobat Viewer sessions.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

For example, to make the "radius" property persistent and accessible for other documents you could use:

```javascript
    global.radius = 8; // declare radius to be global
    global.setPersistent("radius", true); // now say it's persistent
```

The volume calculation, defined above, will now yield the same result across viewer sessions, or until the value of `global.radius` is changed.
subscribe

5.0

Allows you to automatically update one or more fields when the value of the subscribed global variable changes. If the specified property is changed, even in another document, the specified function is called. Multiple subscribers are allowed for a published property.

Parameters

<table>
<thead>
<tr>
<th>cVariable</th>
<th>The global property.</th>
</tr>
</thead>
<tbody>
<tr>
<td>fCallback</td>
<td>The function to call when the property is changed.</td>
</tr>
</tbody>
</table>

Returns

Nothing

Example

Suppose there are two files, setRadius.pdf and calcVolume.pdf, open in Acrobat or Reader.

- In setRadius.pdf there is a single button with the code:
  ```javascript
global.radius = 2;
```
- In calcVolume.pdf there is a Document-Level JavaScript named subscribe:
  ```javascript
  // In the Advanced > JavaScripts > Document JavaScripts
  global.subscribe("radius", RadiusChanged);
  function RadiusChanged(x) // callback function
  {
      var V = (4/3) * Math.PI * Math.pow(x,3);
      this.getField("MyVolume").value = V; // put value in text field
  }
```
- Open both files in the Viewer, now, clicking on the button in setRadius.pdf file immediately gives an update in the text field "MyVolume" in calcVolume.pdf of 33.51032 (as determined by global.radius = 2).

The syntax of the callback function is as follows:

```javascript
function fCallback(newval) {
// newval is the new value of the global variable you
// have subscribed to.
< code to process the new value of the global variable >
}
```
Icon Generic Object

This generic JS object is an opaque representation of a Form XObject appearance stored in `doc.icons`. It is used with Field Objects of type `button`. The `icon` object contains the following property:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td>R</td>
<td>The name of the icon. An icon may or may not have a name depending on whether it exists in the document-level named icons tree.</td>
</tr>
</tbody>
</table>

Icon Stream Generic Object

This generic JS object represents an icon stream. It is used by `app.addToolButton` and `collab.addStateModel`. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>read(nBytes)</td>
<td>A function which takes the number of bytes to read and returns a Hex encoded string. The data should be the icon representation as a 32 bit per pixel with 4 channels (ARGB) 8 bits per channel with the channels interleaved. If the icon has multiple layers, then the function may return the pixels for the topmost layer, followed by the next layer behind it, and so on.</td>
</tr>
<tr>
<td>width</td>
<td>The icon width in pixels.</td>
</tr>
<tr>
<td>height</td>
<td>The icon height in pixels.</td>
</tr>
</tbody>
</table>

Identity Object

This is a static object that identifies the current user of the application.

**Note:** (Security 5): **Identity** object properties are only accessible during batch, console, menu, and application initialization events in order to protect the privacy of the user.
Identity Properties

**corporation**

The corporation name that the user has entered in the identity preferences panel.

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

**email**

The email address that the user has entered in the identity preferences panel.

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

**loginName**

The login name as registered by the operating system.

*Type: String*  
*Access: R.*

**name**

The user name that the user entered in the identity preferences panel.

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

**Example**

```javascript
console.println("Your name is " + identity.name);
consle.println("Your e-mail is " + identity.email);
```

**Index Object**

This is a non-creatable object returned by various methods of the Search Object and Catalog Object. The `index` object represents a Catalog-generated index. You use this object to perform various indexing operations using Catalog. You can find the status of the index with a search.
Index Properties

available

Whether the index is available for selection and searching. An index may be unavailable if a network connection is down or a CD-ROM is not inserted, or if the index administrator has brought the index down for maintenance purposes.

_Type: Boolean  Access: R._

name

The name of the index as specified by the index administrator at indexing time. See `search.indexes`, which returns an array of the index objects currently accessed by the search engine.

_Type: String  Access: R._

Example

```javascript
// 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);
}
```

path

The device-dependent path where the index resides. See Section 3.10.1, “File Specification Strings”, in the PDF Reference for exact syntax of the path.

_Type: String  Access: R._

selected

Whether the index is to participate in the search. If _true_, the index will be searched as part of the query, if _false_ it will not be. Setting or unsetting this property is equivalent to checking the selection status in the index list dialog.

_Type: Boolean  Access: R/W._
Index Methods

build

Builds the index associated with the `index` object using the Catalog plug-in.

The index is built at the same location as the index file. If the index already exists, the included directories are re-scanned for changes and the index is updated. If the index does not exist, a new index is built.

The index build is started immediately if Catalog is idle. Otherwise, it gets queued with Catalog.

Parameters

- `cExpr` (optional) An expression to be evaluated once the build operation on the index is complete. Default is no expression. See the PDF Reference, "JavaScript Action" for more details.
- `bRebuildAll` (optional) If `true`, a clean build is performed. The index is first deleted and then built. The default is `false`.

Returns

A `CatalogJob Generic Object`. The `CatalogJob` object can be used to check the job parameters and status.

Example

```javascript
/* Building an index */
if (typeof catalog != "undefined") {
  var idx = catalog.getIndex("/c/mydocuments/index.pdx");
  var job = idx.build("Done()");
  console.println("Status : ", job.status);
}
```

Link Object

This object is used to set and get the properties and to set the JavaScript action of a link. A `link` object is obtained from `doc.addLink` or `doc.getLinks`.

See also, `doc.removeLinks`.
**Link Properties**

**borderColor**

The border color of a **link** object. See Color Arrays for information on defining color arrays and how colors are used with this property.

*Type: Array  Access: R/W.*

**borderWidth**

The border width of the **link** object.

*Type: Integer  Access: R/W.*

**highlightMode**

The visual effect to be used when the mouse button is pressed or held down inside an active area of a link. The valid values are:

- None
- Invert *(default)*
- Outline
- Push

*Type: String  Access: R/W.*

**rect**

The rectangle in which the link is located on the page. Contains an array of four numbers, the coordinates in *rotated user space* of the bounding rectangle, listed in the following order: upper-left x, upper-left y, lower-right x and lower-right y.

*Type: Array  Access: R/W.*
Link Methods

**setAction**

Sets the specified JavaScript action for the MouseUp trigger for the link object.

**Parameters**

- **cScript**
  - The JavaScript action to use.

**Returns**

Nothing

OCG Object

An OCG object represents an optional-content group in a PDF file. Content in the file can be associated with one or more optional-content groups. Content belonging to one or more OCGs is referred to as optional content, and its visibility is determined by the on/off states of the OCGs to which it belongs. In the simplest case, optional content will belong to a single OCG with the content being visible when the OCG is on and hidden when the OCG is off. More advanced visibility behavior can be achieved by using multiple OCGs and different visibility mappings.

Use `doc.getOCGs` to get an array of OCG objects for a PDF document.

OCG Properties

**name**

The text string seen in the UI for this OCG. It can be used to identify OCGs, although it is not necessarily unique.

**NOTE:** This property is not necessarily unique among OCGs in a document.

**Type:** String

**Access:** R.

**Example**

```javascript
/* Toggle the Watermark OCG */
```
function ToggleWatermark(doc)
{
    var ocgArray = doc.getOCGs();
    for (var i=0; i < ocgArray.length; i++) {
        if (ocgArray[i].name == "Watermark") {
            ocgArray[i].state = !ocgArray[i].state;
        }
    }
}

state

Represents the current on/off state of this OCG.

Type: Boolean Access: R/W.

Example:

Turn on all the OCGs in the given document.

function TurnOnOCGsForDoc(doc)
{
    var ocgArray = doc.getOCGs();
    for (var i=0; i < ocgArray.length; i++) {
        ocgArray[i].state = true;
    }
}

OCG Methods

setAction

Registers a JavaScript expression to be evaluated after every state change for this OCG.

Parameters

<table>
<thead>
<tr>
<th>cExpr</th>
<th>The expression to be evaluated after the OCG state changes.</th>
</tr>
</thead>
</table>

Returns

Nothing

Example

/* Beep when the given ocg is changed */
PlugIn Object

This object gives access to information about the plug-in it represents. A plugIn object is obtained using `app.plugIns`.

PlugIn Properties

- **certified**
  
  If `true`, the plug-in is certified by Adobe. Certified plug-ins have undergone extensive testing to ensure that breaches in application and document security do not occur. The user can configure the viewer to only load certified plug-ins.

  *Type: Boolean  Access: R.

  **Example**

  ```javascript
  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**
  
  If `true`, the plug-in was loaded.

  *Type: Boolean  Access: R.

- **name**
  
  The name of the plug-in.

  *Type: String  Access: R.

  **Example**

  ```javascript
  // 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**

The device-independent path to the plug-in. See “File Specification Strings”, Section 3.10.1, in the PDF Reference for the exact syntax of the path.

*Type: String  Access: R.*

**version**

The version number of the plug-in. The integral part of the version number indicates the major version, the decimal part indicates the minor and update versions. For example, 5.11 would indicate that the plug-in is major version 5, minor version 1, and update version 1.

*Type: Number  Access: R.*

---

**printParams Object**

This object controls printing parameters that affect any document printed via JavaScript. Changing this object does not change the user preferences or make any permanent changes to the document.

In Acrobat version 6.0, `doc.print` takes a `printParams` object as its argument. You can obtain `printParams` object from `doc.getPrintParams`. The returned object can then be modified.

Many of the `printParams` properties take integer constants as values, which you can access using `constants`. For example:

```javascript
// get the printParams object of the default printer
var pp = this.getPrintParams();
// set some properties
pp.interactive = pp.constants.interactionLevel.automatic;
pp.colorOverride = pp.colorOverrides.mono;
// print
this.print(pp);
```

The `constants` object properties are all Integers, and are all Read access.
PrintParams Properties

binaryOK

true if a binary channel to the printer is supported. The default is true.

Type: Boolean Access: R/W.

bitmapDPI

The dots per inch (DPI) to use when producing bitmaps or rasterizing transparency. Valid range is 1 to 9600. If the document protections specify a maximum printing resolution, the lower of the two values is used. The default is 300. Illegal values are treated as 300. See also gradientDPI.

Type: Integer Access: R/W.

colorOverride

Whether to use color override. Values are the properties of the constants colorOverrides Object. Illegal values are treated as auto, the default value.

NOTE: This property is supported on Windows platforms only.

colorOverrides Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Let Acrobat decide color overrides. This is the default.</td>
</tr>
<tr>
<td>gray</td>
<td>Force color to grayscale.</td>
</tr>
<tr>
<td>mono</td>
<td>Force color to monochrome.</td>
</tr>
</tbody>
</table>

Type: Integer constant Access: R/W.

Example

var pp = this.getPrintParams();
pp.colorOverride = pp.constants.colorOverrides.mono;
this.print(pp);

**colorProfile**

| 6.0 |  

The color profile to use. A list of available color spaces can be obtained from the `printColorProfiles`. The default is "Printer/PostScript Color Management"

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

**constants**

| 6.0 |  

Each instance of a `printParams` object inherits this property, which is a wrapper object for holding various constant values. The `constants` object property values are all Integers, and are all Read access. The values are listed with the `printParams` properties to which they apply.

The `constants` objects are used to specify option values of some of the other properties of the `printParams` object, as shown in the following table:

<table>
<thead>
<tr>
<th>constant object</th>
<th>contains constant values for printParams property</th>
</tr>
</thead>
<tbody>
<tr>
<td>colorOverridess</td>
<td>colorOverride</td>
</tr>
<tr>
<td>fontPolicies</td>
<td>fontPolicy</td>
</tr>
<tr>
<td>handling</td>
<td>pageHandling</td>
</tr>
<tr>
<td>interactionLevel</td>
<td>interactive</td>
</tr>
<tr>
<td>printContents</td>
<td>printContent</td>
</tr>
<tr>
<td>flagValues</td>
<td>flags</td>
</tr>
<tr>
<td>rasterFlagValues</td>
<td>rasterFlags</td>
</tr>
<tr>
<td>subsets</td>
<td>pageSubset</td>
</tr>
<tr>
<td>tileMarks</td>
<td>tileMark</td>
</tr>
<tr>
<td>usages</td>
<td>usePrinterCRD</td>
</tr>
<tr>
<td></td>
<td>useT1Conversion</td>
</tr>
</tbody>
</table>

*Type: object*  
*Access: R.*
downloadFarEastFonts

When `true`, send Far East fonts to the printer if needed. Set to `false` if printer has Far East fonts but incorrectly reports it needs them. The default is `true`.

*Type: Boolean  Access: R/W.*

fileName

If not empty, the device-independent pathname for a filename to be used instead of sending the print job to the printer (Print to File). The pathname may be relative to the location of the current document. When printing to a file, if the interaction level (See `interactive`) is set to `full`, it is lowered to `automatic`. The default value is the empty string.

**Note:** Printing to a file produces output suitable for the printer, for example, Postscript or GDI commands.

**Note:** When `printerName` is an empty string and `fileName` is nonempty the current document is saved to disk as a PostScript file.

*Type: String  Access: R/W.*

**Example**

```javascript
var pp = this.getPrintParams();
pp.fileName = "/c/print/myDoc.prn";
this.print(pp);
```

**Example 2**

Save the current document as a PostScript file.

```javascript
var pp = this.getPrintParams();
pp.fileName = "/c/temp/myDoc.ps";
pp.printerName = "";
this.print(pp);
```
**firstPage**

The first 0-based page number of the document to print. The first page of any document is 0, regardless of page number labels. Values out of the document page range are treated as 0. The default value is 0.

See also [lastPage](#).

Type: Integer Access: R/W.

**Example**

```javascript
var pp = this.getPrintParams();
pp.firstPage = 0;
pp.lastPage = 9;
this.print(pp);
```

**flags**

A bit field of flags to control printing. These flags can be set or cleared using bitwise operations through the `flagValues Object`.

Zero or more flags can be set; unsupported flags are ignored. The flags default to those set by user preferences.

**flagValues Object**

Where 🗿 appears in the Reader column, the property is not available for any version of the Adobe Reader.

<table>
<thead>
<tr>
<th>Property</th>
<th>Reader</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applyOverPrint</td>
<td>🗿</td>
<td>Do overprint preview when printing, turn off if print natively supports overprinting</td>
</tr>
<tr>
<td>applySoftProofSettings</td>
<td>🗿</td>
<td>Use the softProofing settings before doing color management</td>
</tr>
<tr>
<td>applyWorkingColorSpaces</td>
<td>🗿</td>
<td>Apply working color spaces when printing</td>
</tr>
<tr>
<td>emitHalftones</td>
<td>🗿</td>
<td>Emit the halftones specified in the document</td>
</tr>
<tr>
<td>emitPostScriptXObjects</td>
<td>🗿</td>
<td>PostScript only, do include PostScript XObjects’ content in output</td>
</tr>
<tr>
<td>emitFormsAsPSForms</td>
<td>🗿</td>
<td>Converts Form XObjects to PS forms. The default is off.</td>
</tr>
<tr>
<td>Property</td>
<td>Reader</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pageSize</td>
<td></td>
<td>Enable pageSize, choose paper tray by PDF page size</td>
</tr>
<tr>
<td>suppressBG</td>
<td>X</td>
<td>Do not emit the BlackGeneration in the document</td>
</tr>
<tr>
<td>suppressCenter</td>
<td></td>
<td>Do not center the page</td>
</tr>
<tr>
<td>suppressCJKFontSubst</td>
<td>X</td>
<td>Suppress CJK Font Substitution on Printer—does not apply when \kAVEmitFontAllFonts\ is used</td>
</tr>
<tr>
<td>suppressCropClip</td>
<td></td>
<td>Do not emit the cropbox page clip</td>
</tr>
<tr>
<td>suppressRotate</td>
<td></td>
<td>Do not rotate the page</td>
</tr>
<tr>
<td>suppressTransfer</td>
<td>X</td>
<td>Do not emit the transfer functions in the document</td>
</tr>
<tr>
<td>suppressUCR</td>
<td>X</td>
<td>Do not emit the UnderColorRemovals in the document</td>
</tr>
<tr>
<td>useTrapAnnots</td>
<td>X</td>
<td>Print TrapNet and PrinterMark annotations, even if printing &quot;document only&quot;.</td>
</tr>
<tr>
<td>usePrintersMarks</td>
<td>P</td>
<td>Print PrinterMark annotations, even if printing &quot;document only&quot;.</td>
</tr>
</tbody>
</table>

**Type**: Integer  
**Access**: R/W.

**Example 1**

Check the “Apply Proof Settings” checkbox Output options in the Advanced Printing Setup dialog.

```javascript
pp = getPrintParams();
fv = pp.constants.flagValues;
// or pp.flags |= fv.applySoftProofSettings;
pp.flags = pp.flags | fv.applySoftProofSettings;
this.print(pp);
```

**Example 2**

Uncheck “Auto-Rotate and Center” (checked by default) in the Print dialog.

```javascript
pp = getPrintParams();
fv = pp.constants.flagValues;
pp.flags |= (fv.suppressCenter | fv.suppressRotate);
this.print(pp);
```
Example 3


```javascript
pp = getPrintParams();
fv = pp.constants.flagValues;
pp.flags &= ~(fv.suppressBG | fv.suppressUCR)
this.print(pp)
```

**fontPolicy**

6.0

Sets the font policy. The value of the `fontpolicy` property is set through the `constants fontPolicies Object`. The default is `pageRange`.

*Type: Integer  Access: R/W.*

**fontPolicies Object**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>everyPage</code></td>
<td>Emit needed fonts before every page, free all fonts after each page. This produces the largest, slowest print jobs, but requires the least amount of memory from the printer.</td>
</tr>
<tr>
<td><code>jobStart</code></td>
<td>Emit all fonts used at the beginning of the print job, free them at the end of the print job. This produces the smallest, fastest print jobs, but requires the most memory from the printer.</td>
</tr>
<tr>
<td><code>pageRange</code></td>
<td>(Default) Emit fonts before the first page that uses them, free them after the last page that uses them. This also produces the smallest, fastest print jobs, and can use less memory. However, the produced print job must be printed as produced due to page ordering. <strong>NOTE:</strong> <code>pageRange</code> can be a good compromise between speed and memory, but do not use it if the postscript pages will be programmatically reordered afterwards.</td>
</tr>
</tbody>
</table>

**gradientDPI**

6.0

The dots per inch to use when rasterizing gradients. This value can generally be set lower than `bitmapDPI` because it affects areas to which the eye is less sensitive. It must be set from 1 to 9600. Illegal values are treated as 150. If the document protections specify a maximum printing resolution, the lower of the two values will be used. The default value is 150.
**interactive**

Sets the level of interaction between the user and the print job. The value of the `interactive` property is set through the `constants InteractionLevel Object`. The default is `full`.

**Example**

```javascript
var pp = this.getPrintParams();
pp.interactive = pp.constants.interactionLevel.automatic;
pp.printerName = "Adobe PDF";
this.print(pp);
```

**lastPage**

The last 0-based page number of the document to print. The term “0-based” means the first page of any document is 0, regardless of page number labels. If the value is less than `firstPage` or outside the legal range of the document, this reverts to the default value. The default value is the number of pages in the document less one.

See `firstPage` for an example.
**pageHandling**

6.0

Takes one of four values. The value of the `pageHandling` property is set through the `constants handling Object`. If set to an illegal value it is treated as `shrink`. The default is `shrink`.

*Type: Integer*  
*Access: R/W.*

**handling Object**

<table>
<thead>
<tr>
<th>Property</th>
<th>Reader</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fit</td>
<td></td>
<td>Pages are enlarged or shrunk to fit the printer's paper</td>
</tr>
<tr>
<td>shrink</td>
<td></td>
<td>Small pages are printed small, large pages are shrunk to fit on the printer's paper</td>
</tr>
<tr>
<td>tileAll</td>
<td>X</td>
<td>All pages are printed using tiling settings. One use of this is to turn a normal sized page into a poster by setting tile zoom &gt; 1</td>
</tr>
<tr>
<td>tileLarge</td>
<td>X</td>
<td>Small or normal pages are printed original size, large pages are printed on multiple sheets of paper.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
var pp = this.getPrintParams();
pp.pageHandling = pp.constants.handling.shrink;
this.print(pp);
```

**pageSubset**

6.0

Select even, odd, or all the pages to print. The value of `pageSubset` is set through the `constants subsets Object`. The default is `all`.

*Type: Integer*  
*Access: R/W.*

**subsets Object**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>Print all pages in page range.</td>
</tr>
<tr>
<td>even</td>
<td>Print only the even pages. Page labels are ignored for this. The document is treated as if it were numbered 1 through n, the number of pages.</td>
</tr>
</tbody>
</table>
**PrintParams Properties**

### Example

```javascript
var pp = this.getPrintParams();
pp.pageSubset = pp.constants.subsets.even;
this.print(pp);
```

### printAsImage

6.0

Set to `true` to send pages as large bitmaps. This can be slow and more jagged looking but can work around problems with a printer’s PostScript interpreter. Set `bitmapDPI` to increase or decrease the resolution of the bitmap. If interaction (see `interactive`) is `full`, the user’s printer preferences for `printAsImage` will be used. The default is `false`.

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

### printContent

6.0

Sets the contents of the print job. The value of the `printContent` property is set through the `constants printContents Object`. The default is `doc`.

**Type:** Integer  
**Access:** R/W.

### printContents Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>doc</code></td>
<td>Emit the document contents. Document comments are not printed</td>
</tr>
<tr>
<td><code>docAndComments</code></td>
<td>Emit the document contents and comments.</td>
</tr>
<tr>
<td><code>formFieldsOnly</code></td>
<td>Emit the contents of form fields only. Useful for printing onto pre-preprinted forms.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
var pp = this.getPrintParams();
pp.interactive = pp.constants.interactionLevel.silent;
pp.printContent = pp.constants.printContent.formFieldsOnly;
this.print(pp);
```
**printerName**

Set or get the name of destination printer. The `printerName` property is a Windows-only feature; currently, the destination printer cannot be set through this property on the Mac.

By default, `printerName` is set to the name of the default printer. If set `printerName` to an empty string the default printer will be used. When `printerName` is an empty string and `fileName` is a nonempty string, the current document is saved to disk as a PostScript file. See Example 2 below.

See also `app.printerNames`.

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

**Example 1**

```javascript
var pp = this.getPrintParams();
pp.printerName = "hp officejet d series";
this.print(pp);
```

**Example 2**

Save the current document as a PostScript file.

```javascript
var pp = this.getPrintParams();
pp.fileName = "/c/temp/myDoc.ps";
pp.printerName = "";
this.print(pp);
```

**psLevel**

Level of PostScript that is emitted to PostScript printers. Level 0 indicates to use the PostScript level of the printer. Level 1 is not supported. In addition to 0, current legal values of `psLevel` are 2 and 3. If the printer only supports PostScript level 1, `printAsImage` is set to `true`. Illegal values are treated as 0. The default value for `psLevel` is 0.

*Type: Integer*  
*Access: R/W.*

**rasterFlags**

A bit field of flags. These flags can be set or cleared using bitwise operations through the `constants rasterFlagValues Object`. The default is set by user preferences.

*Type: Integer*  
*Access: R/W.*
rasterFlagValues Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Reader</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>textToOutline</td>
<td>X</td>
<td>Text converted to outlines can become thicker (especially noticeable on small fonts). If text is mixed into artwork with transparency it may be converted to outline during flattening, resulting in inconsistency with text that is not mixed into artwork. In this case turning on this option will ensure all text looks consistent.</td>
</tr>
<tr>
<td>strokesToOutline</td>
<td>X</td>
<td>Strokes converted to outlines can become thicker (especially noticeable on thin strokes). If strokes are mixed into artwork with transparency they may be converted to outlines during flattening, resulting in inconsistency with strokes that are not mixed into artwork. In this case turning on this option will ensure all strokes looks consistent.</td>
</tr>
<tr>
<td>allowComplexClip</td>
<td>X</td>
<td>Select this to ensure that the boundaries between vector artwork and rasterized artwork fall closely along object paths. Selecting this option reduces stitching artifacts that result when part of an object is flattened while another part of the object remains in vector form. However, selecting this option may result in paths that are too complex for the printer to handle.</td>
</tr>
<tr>
<td>preserveOverprint</td>
<td>X</td>
<td>Select this if you are printing separations and the document contains overprinted objects. Selecting this option generally preserves overprint for objects that are not involved in transparency and therefore improves performance. This option has no effect when printing composite. Turning this off might result in more consistent output since all overprinting will be flattened whether it is involved in transparency or not.</td>
</tr>
</tbody>
</table>

Example 1

Check the "Convert All Text to Outlines" checkbox in the Transparency Flattening option of the Advanced Print Setup.

```javascript
pp = getPrintParams();
rf = pp.constants.rasterFlagValues;
pp.rasterFlags |= rf.textToOutline;
this.print(pp);
```

Example 2

Uncheck "Complex Clip Regions" (checked by default) in the Transparency Flattening option of the Advanced Print Setup.

```javascript
pp = getPrintParams();
```
rf = pp.constants.rasterFlagValues;
pp.rasterFlags = pp.rasterFlags & ~rf.allowComplexClip;
// or pp.rasterFlags &= ~rf.allowComplexClip;
this.print(pp);

**reversePages**

Set to `true` to print pages in reverse order (last to first). The default value is `false`.

*Type: Boolean  Access: R/W.*

**tileLabel**

Label each page of tiled output. Labeled pages indicate row and column, filename, and print date. The default is `false`.

*Type: Boolean  Access: R/W.*

**tileMark**

Tile marks indicate where to cut the page and where overlap occurs. The value is set through the `constants` `tileMarks Object`. If set to an illegal value it is treated as `none`. The default is `none`.

*Type: Integer  Access: R/W.*

**tileMarks Object**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>none</code></td>
<td>No tile marks</td>
</tr>
<tr>
<td><code>west</code></td>
<td>Western style tile marks</td>
</tr>
<tr>
<td><code>east</code></td>
<td>Eastern style tile marks</td>
</tr>
</tbody>
</table>
tileOverlap

The number of points that tiled pages have in common. Value must be between 0 and 144. Illegal values are treated as 0. The default value is 0.

Type: Integer  Access: R/W.

tileScale

The amount that tiled pages are scaled. Pages that are not tiled are unaffected by this value. Default is unscaled (1.0). Larger values increase the size of the printout (for example, 2.0 is twice as large, a value of 0.5 is half as large). The value of tileScale must be between 0.01 and 99.99. Illegal values are treated as 1.0, which is the default value.

Type: Number  Access: R/W.

transparencyLevel

An integer value from 1 to 100 indicates how hard Acrobat tries to preserve high level drawing operators. A value of 1 indicates complete rasterization of the image which results in poor image quality but high speeds. A value of 100 indicates as much should be preserved as possible, but can result in slow print speeds. If set to an illegal value, 100 is used. When rasterizing, the bitmapDPI and gradientDPI values are used. The default value is 100.

Type: Integer  Access: R/W.

usePrinterCRD

Takes one of three values. The value is set through the constants usages Object. See also usePrinterCRD; the two properties use the same values, but the interpretations are different.

Type: Integer  Access: R/W.
### usages Object

<table>
<thead>
<tr>
<th>Property</th>
<th>Description for usePrinterCRD</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Let Acrobat decide if printer Color Rendering Dictionary should be used. Acrobat maintains a list of a handful of printers that have incorrect CRDs. Illegal values are treated as auto. The default is auto.</td>
</tr>
<tr>
<td>use</td>
<td>Use printer’s Color Rendering Dictionary.</td>
</tr>
<tr>
<td>noUse</td>
<td>Do not use printer’s Color Rendering Dictionary.</td>
</tr>
</tbody>
</table>

### useT1Conversion

| 6.0 |

Takes one of three values. The value of the `useT1Conversion` property is set through the `constants usages Object`. See also `usePrinterCRD`; the two properties use the same values, but the interpretations are different.

**NOTE:** This property is supported on Windows platforms only.

*Type: Integer*  
*Access: R/W.*

This property uses the `usages Object` values as follows.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description for useT1Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>Let Acrobat decide whether to disable converting Type 1 fonts to more efficient printer representations (for example, TrueType). Acrobat maintains a list of a handful of printers that have problems with these fonts. Illegal values are treated as auto. The default is auto.</td>
</tr>
<tr>
<td>use</td>
<td>Allow conversion of Type 1 fonts even if printer is known to have problems with alternative font representations.</td>
</tr>
<tr>
<td>noUse</td>
<td>Never convert Type 1 fonts to more efficient representations.</td>
</tr>
</tbody>
</table>

### RDN Generic Object

This generic object represents a Relative Distinguished Name. It is used by `securityHandler`, `newUser` and the `certificate.issuerDN` and `subjectDN` properties.
It has the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>String</td>
<td>R</td>
<td>Country or Region. Must be a two-character upper case ISO 3166 standard string (for example, 'US')</td>
</tr>
<tr>
<td>cn</td>
<td>String</td>
<td>R</td>
<td>Common name (for example, 'John Smith')</td>
</tr>
<tr>
<td>o</td>
<td>String</td>
<td>R</td>
<td>Organization name (for example, 'Adobe Systems Inc.')</td>
</tr>
<tr>
<td>ou</td>
<td>String</td>
<td>R</td>
<td>Organizational unit (for example, 'Acrobat Engineering')</td>
</tr>
<tr>
<td>e</td>
<td>String</td>
<td>R</td>
<td>Email address (for example, '<a href="mailto:jsmith@adobe.com">jsmith@adobe.com</a>')</td>
</tr>
</tbody>
</table>

**Report Object**

The Report object allows the user to programmatically generate PDF documents suitable for reporting with JavaScript. Use the `Report` constructor to create a `Report` object; for example,

```javascript
var rep = new Report();
```

The properties and methods can then be used to write and format a report.

**Report Properties**

**absIndent**

| 5.0 | ☐ | ☒ | ☒ |

Controls the absolute indentation level. It is desirable to use indent/outdent only whenever possible, as those calls correctly handle indentation overflows.

If a report is indented past the middle of the page, the effective indent is set to the middle. Note that `divide` does a little squiggly bit to indicate that it’s been indented too far.

*Type: Number  Access: R/W.*

**color**

| 5.0 | ☐ | ☒ | ☒ |

Controls the color of any text and any divisions written into the report.
Text is written to the report with `writeText` and divisions (horizontal rules) are written using `divide`.

**Type:** Color  
**Access:** R/W.

**Example**

```javascript
var rep = new Report();
rep.size = 1.2;
rep.color = color.blue;
rep.writeText("Hello World!");
```

**size**

Controls the size of any text created by `writeText`. It is a multiplier. Text size is determined by multiplying the `size` property by the default size for the given style.

**Type:** Number  
**Access:** R/W.

**Example**

```javascript
var rep = new Report();
rep.size = 1.2;
rep.writeText("Hello World!");
```

**style**

This property controls the style of the text font for the text created by `writeText`. Values of `style` are:

- DefaultNoteText
- NoteTitle

**Example**

```javascript
var rep = new Report();
rep.size = 1.2;
rep.style = "DefaultNoteText";
rep.writeText("Hello World!");
rep.open("My Report");
```
Report Methods

breakPage

Ends the current page and begins a new one.

Parameters
None

Returns
Nothing

divide

Writes a horizontal rule across the page at the current location with the given width. The rule goes from the current indent level to the rightmost edge of the bounding box. If the indent level is past the middle of the bounding box, the rule has a squiggly bit to show this.

Parameters

nWidth (optional) The horizontal rule width to use.

Returns
Nothing

indent

Increments the current indentation mark by nPoints or the default amount. If a report is indented past the middle of the page, the effective indent is set to the middle. Note that divide makes a squiggly bit to indicate that it has been indented too far.

See writeText for an example of usage.
outdent

| 5.0 | D | X | X |

The opposite of indent; that is, decrements the current indentation mark by \( n\text{Points} \) or the default amount.

See \texttt{writeText} for an example of usage.

Parameters

\( n\text{Points} \) (optional) The number of points to decrement the indentation mark.

Returns

Nothing

open

| 5.0 | D | X | X |

Ends report generation, opens the report in Acrobat and returns a \texttt{Doc Object} that can be used to perform additional processing of the report.

Parameters

\( c\text{Title} \) The report title.

Returns

A \texttt{Doc Object}.

Example

\begin{verbatim}
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";
\end{verbatim}

See \texttt{writeText} for a more complete example.
**save**

5.0

Ends report generation and saves the report to the specified path.

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cDIPath</td>
<td>The device-independent path.</td>
</tr>
<tr>
<td>cFS</td>
<td>(optional) The file system. The only value for cFS is &quot;CHTTP&quot;; in this case, the cDIPath parameter should be an URL. This parameter is only relevant if the web server supports WebDAV.</td>
</tr>
</tbody>
</table>

**Returns**

Nothing

**Example 1**

rep.save("/c/myReports/myreport.pdf");

**Example 2**

rep.save("http://www.mycompany/reports/myreport.pdf", "CHTTP");

**mail**

5.0

Ends report generation and mails the report. See also mailGetAddr, mailMsg, mailDoc, mailForm.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bUI</td>
<td>(optional) Whether to display a user interface. If true (the default) the rest of the parameters are used to seed the compose-new-message window that is displayed to the user. If false, the cTo parameter is required and all others are optional.</td>
</tr>
<tr>
<td>cTo</td>
<td>(optional) A semicolon-separated list of recipients for the message.</td>
</tr>
<tr>
<td>cCc</td>
<td>(optional) A semicolon-separated list of CC recipients for the message.</td>
</tr>
<tr>
<td>cBcc</td>
<td>(optional) A semicolon-separated list of BCC recipients for the message.</td>
</tr>
</tbody>
</table>
Returns

Nothing

Report

A constructor. Creates a new Report object with the given media and bounding boxes (values are defined in points or 1/72 of an inch). Defaults to a 8.5 x 11 inch media box and a bounding box that is indented .5 inches on all sides from the media box.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aMedia</td>
<td>(optional) The media type.</td>
</tr>
<tr>
<td>aBBox</td>
<td>(optional) The bounding box size.</td>
</tr>
</tbody>
</table>

Returns

Nothing

writeText

Writes out a block of text to the report. Every call is guaranteed to begin on a new line at the current indentation mark. Correctly wraps Roman, CJK, and WGL4 text.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>The block of text to use.</td>
</tr>
</tbody>
</table>

Example

```javascript
// Get the comments in this document, and sort by author
this.syncAnnotScan();
annots = this.getAnnots({nSortBy: ANSB_Author});

// open a new report
var rep = new Report();
rep.size = 1.2;
rep.color = color.blue;
```
rep.writeText("Summary of Comments: By Author");
rep.color = color.black;
rep.writeText(" ");
rep.writeText("Number of Comments: " + annots.length);
rep.writeText(" ");

var msg = "$200 page %s: \"%s\"";
var theAuthor = annots[0].author;
rep.writeText(theAuthor);
rep.indent(20);
for (var i=0; i < annots.length; i++) {
  if (theAuthor != annots[i].author) {
    theAuthor = annots[i].author;
    rep.writeText(" ");
    rep.outdent(20);
    rep.writeText(theAuthor);
    rep.indent(20);
  }
  rep.writeText(util.printf(msg, 1 + annots[i].page, annots[i].contents));
}

// now open the report
var docRep = rep.open("myreport.pdf");
docRep.info.Title = "End of the month report: August 2000";
docRep.info.Subject = "Summary of comments at the August meeting";

See the file Annots.js for additional examples of the Report object.

---

### Row Generic Object

This generic JS object contains the data from every column in a row. It is returned by `statement.getRow`. It contains the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>columnArray</code></td>
<td>Array</td>
<td>R</td>
<td>An array of Column Generic Objects. This is equivalent to what <code>statement.getColumnArray</code> would return if called on the same <code>statement</code> at the same time that this row object was created.</td>
</tr>
<tr>
<td><code>column properties</code></td>
<td>any</td>
<td>R</td>
<td>There is a property corresponding to each column selected by the query, containing the data for that row in that column.</td>
</tr>
</tbody>
</table>
The search object is a static object that accesses the functionality provided by the Acrobat Search plug-in. This plug-in must be installed in order to interface with the search object (see available).

See also the Index Object, which is returned by some of the methods of the search object.

The results for query calls are displayed in Acrobat's Find dialog.

**NOTE:** Acrobat 6.0 indexes are incompatible with the search engines of prior versions of Acrobat.

**NOTE:** In Acrobat 6.0, searching indexes created by versions of Acrobat prior to 6.0 is not possible on the Mac platform.

### Search Properties

<table>
<thead>
<tr>
<th>available</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns true if the Search plug-in is loaded and query capabilities are possible. A script author should check this boolean before performing a query or other search object manipulation.</td>
</tr>
</tbody>
</table>

**Type:** Boolean  
**Access:** R.

**Example**

Make sure the search object exists and is available.

```javascript
if (typeof search != "undefined" && search.available) {
    search.query("Cucumber");
}
```

<table>
<thead>
<tr>
<th>docInfo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whether the document Information is searched for the query. The default is false.</td>
</tr>
</tbody>
</table>

**Type:** Boolean  
**Access:** R/W.
**docText**

Whether the document text is searched for the query. The default is `true`.

*Type: Boolean  Access: R/W.*

**docXMP**

Whether document-level XMP metadata is searched for the query. The default is `false`.

*Type: Boolean  Access: R/W.*

**bookmarks**

Whether bookmarks are searched for the query. The default is `false`.

*Type: Boolean  Access: R/W.*

**ignoreAsianCharacterWidth**

Whether the Kana characters in the document exactly match the search query. The default is `false`.

*Type: Boolean  Access: R/W.*

**indexes**

Returns an array of all of the Index Objects currently accessible by the search engine.

*Type: Array  Access: R.*

**Example**

Enumerate all of the indexes and dump their names.

```javascript
for (var i = 0; i < search.indexes.length; i++) {
    console.println("Index[" + i + "]=" + search.indexes[i].name);
}
```
**jpegExif**

Whether EXIF data associated with JPEG images in the PDF is searched. The default is false.

*Type: Boolean  Access: R/W.*

**legacySearch**

Returns true if the Search5.api plug-in is loaded. Search5.api plug-in provides the capability to search indexes generated by Acrobat Catalog in Acrobat 5.0 (or earlier version). See the sections in the Acrobat Online Guide pertaining to searching such indexes.

*Type: Boolean  Access: R.*

**markup**

Whether markup (annotations) are searched for the query. The default is false.

*Type: Boolean  Access: R/W.*

**matchCase**

Whether the search query is case sensitive. The default is false.

*Type: Boolean  Access: R/W.*

**matchWholeWord**

Whether search finds only occurrences of complete words that are specified in the query. For example, when this option is set to true, if you search for the word "stick", the words "tick" and "sticky" will not be highlighted. The default is false.

*Type: Boolean  Access: R/W.*
maxDocs

The maximum number of documents that will be returned as part of the search query. The default is 100 documents.

*Type: Integer  Access: R/W.*

proximity

Whether the search query will reflect the proximity of words in the results ranking when performing the search that contains AND boolean clauses. The default is `false`. See the sections in the Acrobat Online Guide pertaining to Search capabilities for a more thorough discussion of proximity.

*Type: Boolean  Access: R/W.*

refine

Whether the search query will take the results of the previous query and refine the results based on the next query. The default is `false`. See the sections in the Acrobat Online Guide pertaining to Search capabilities for a more thorough discussion of refining queries.

*Type: Boolean  Access: R/W.*

soundex

Whether the search query will take the sound of words (for example, MacMillan, McMillan, McMilon) into account when performing the search. The default is `false`. See the sections in the Acrobat Online Guide pertaining to Search capabilities for a more thorough discussion of soundex.

**NOTE:** Beginning with Acrobat 6.0, the use of this property is discouraged. This property has a value of `false` and access is restricted to read only.

*Type: Boolean  Access: R.*

stem

Whether the search query will take the stemming of words (for example, run, runs, running) into account when performing the search. The default is `false`. See the sections in the Acrobat Online Guide pertaining to Search capabilities for a more thorough discussion of stemming.

*Type: Boolean  Access: R/W.*
thesaurus

Whether the search query will find similar words. For example, searching for "embellish" might yield "enhanced", "gracefully", or "beautiful". The default is false.

**NOTE:** Beginning with Acrobat 6.0, the use of this property is discouraged. This property has a value of false and access is restricted to read only.

*Type: Boolean  Access: R.*

wordMatching

How individual words in the query will be matched to words in the document. Values are:

- MatchPhrase
- MatchAllWords
- MatchAnyWord
- BooleanQuery (default)

This property is relevant only when a query has more than one word. The BooleanQuery option is ignored when searching active document.

*Type: String  Access: R/W.*

---

**Search Methods**

**addIndex**

Adds the specified index to the list of searchable indexes.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cDIPath</td>
<td>A device-independent path to an index file on the user's hard drive. See “File Specification Strings”, Section 3.10.1, in the PDF Reference for the exact syntax of the path.</td>
</tr>
<tr>
<td>bSelect</td>
<td>(optional) Whether the index should be selected for searching.</td>
</tr>
</tbody>
</table>
Returns

An Index Object.

Example

Adds the standard help index for Acrobat to the index list:

```javascript
search.addIndex("/c_program files/adobe/acrobat 5.0/help/exchhelp.pdx", true);
```

getIndexForPath

Searchs the index list and returns the index object whose path corresponds to the specified path.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cDIPath</td>
<td>A device-independent path to an index file on the user's hard drive. See &quot;File Specification Strings&quot;, Section 3.10.1, in the PDF Reference for the exact syntax of the path.</td>
</tr>
</tbody>
</table>

Returns

The Index Object whose path corresponds to the specified path.

query

Searches the specified document or index for the specified text. Properties associated with the search object (such as matchCase, matchWholeWord, stem) may affect the result.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cQuery</td>
<td>The text for which to search.</td>
</tr>
<tr>
<td>cWhere</td>
<td>(optional) Specifies where the text should be searched. Values are: ActiveDoc, Folder, Index, ActiveIndexes (default).</td>
</tr>
<tr>
<td>cDPIPath</td>
<td>(optional) A device-independent path to a folder or Catalog index on the user's computer. See &quot;File Specification Strings&quot;, Section 3.10.1, in the PDF Reference for the exact syntax of the path. When cWhere is Folder or Index, this parameter is required.</td>
</tr>
</tbody>
</table>

Returns

Nothing
Examples

Search for the word "Acrobat".

<table>
<thead>
<tr>
<th>cWhere</th>
<th>Query</th>
</tr>
</thead>
</table>
| ActiveIndexes | search.query("Acrobat"); // "ActiveIndexes" is the default.  
                 | search.query("Acrobat", "ActiveIndexes");                     |
| ActiveDoc  | search.query("Acrobat", "ActiveDoc");                              |
| Folder     | search.query("Acrobat", "Folder", "/c/myDocuments");             |
|            | search.query("Acrobat", "Folder", "/myserver/myDocuments");       |
| Index      | search.query("Acrobat", "Index", "/c/Myfiles/public/index.pdx"); |

removeIndex

Removes the specified index object from the index list.

Parameters

<table>
<thead>
<tr>
<th>index</th>
<th>The Index Object to remove from the index list.</th>
</tr>
</thead>
</table>

Returns

Nothing

Security Object

The security object is a static JavaScript object that exposes security-related PDF functions such as encryption and digital signatures. Security functions are performed using a SecurityHandler Object which is obtained from the security object using the getHandler method.

NOTE: (Security=S): The Security Object is available without restriction, including in Adobe Reader. The methods and properties of the Security Object can only be executed during batch, console, menu, or application initialization events including in Adobe Reader, except where otherwise stated. See the Event Object for a discussion of Acrobat JavaScript events.
Security Properties

handlers

Returns an array containing the language-independent names of the available security handlers that can be used for encryption or signatures. See also SecurityHandler Object. Beginning with Acrobat 6.0, access to this property is unrestricted, to allow querying to see what handlers are available.

Type: Array Access: R.

validateSignaturesOnOpen

Gets or sets the user-level preference that causes signatures to be automatically validated when a document is opened.

NOTE: (Security ☐) : The property can be used to get in all situations, but can only set new values during batch, console, application initialization and menu events.

Type: Boolean Access: R/W.

Security Methods

chooseRecipientsDialog

Opens a dialog that allows a user to choose a list of recipients. Returns an array of generic Group objects that can be used when encrypting documents or data using either encryptForRecipients or addRecipientListCryptFilter methods of the Doc Object.

NOTE: Can be executed only during console, menu, or application initialization events. Not available in Reader.
Parameters

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oOptions</td>
<td>A DisplayOptions Generic Object containing the parameters for the display options.</td>
</tr>
</tbody>
</table>

Returns

An array of generic Group Objects.

See doc. `encryptForRecipients` for a description of the generic Group Object.

DisplayOptions Generic Object

It contains the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bAllowPermGroups</td>
<td>Controls whether permissions can be set for entries in the recipient list. Default value is <code>true</code>.</td>
</tr>
<tr>
<td>bPlaintextMetadata</td>
<td>Controls whether the checkbox is displayed that allows a user to select whether meta data is plaintext or encrypted, and also the default value. If not specified, the checkbox is not shown. If specified, the checkbox is shown and the default value is the value of this property.</td>
</tr>
<tr>
<td>cTitle</td>
<td>The title to be displayed in the dialog. The default is 'Choose Recipients'.</td>
</tr>
<tr>
<td>cNote</td>
<td>A note to be displayed in the dialog. The default is to not show any note.</td>
</tr>
<tr>
<td>bAllowImportFromFile</td>
<td>Whether the option is displayed that allows a user to import recipients from a file. The default value is <code>true</code>.</td>
</tr>
<tr>
<td>bRequireEncryptionCert</td>
<td>If <code>true</code>, recipients will be required to include an encryption certificate. The default value is <code>true</code>.</td>
</tr>
<tr>
<td>bRequireEmail</td>
<td>If <code>true</code>, recipients will be required to include an email address. The default value is <code>false</code>.</td>
</tr>
<tr>
<td>bUserCert</td>
<td>If <code>true</code>, the user will be prompted to provide his or her own certificate so that he or she can be included in the list of recipients. Setting this flag to <code>true</code> results in a prompt but does not require that the user provide a certificate.</td>
</tr>
</tbody>
</table>

Example 1

Retrieve groups with permissions
var oOptions = {
    bAllowPermGroups: true,
    bPlaintextMetadata: false,
    cTitle: "Encrypt and Email",
    cNote: "Select recipients",
    bAllowImportFromFile: false,
    bRequireEncryptionCert: true,
    bRequireEmail: true
};
var groupArray = security.chooseRecipientsDialog( oOptions );
console.println("Full name = "+ groupArray[0].userEntities[0].fullName);

Example 2

Get a list of recipients for which to encrypt data and then possibly email the document once done.

var oOptions = { bAllowPermGroups: false,
    cNote: "Select the list of recipients. 
    + "Each person must have both an email address and a certificate."",
    bRequireEmail: true,
    bUserCert: true
};
var oGroups = security.chooseRecipientsDialog( oOptions );
// Display the list of recipients in an alert
// Build an email "to" mailList
var numCerts = oGroups[0].userEntities.length;
var cMsg = "The document will be encrypted for the following:
";
var mailList = new Array;
for( var g=0; g<numCerts; ++g )
{
    var ue = oGroups[0].userEntities[g];
    var oCert = ue.defaultEncryptCert;
    if( oCert == null )
        oCert = ue.certificates[0];
    cMsg += oCert.subjectCN +", " + ue.email + "\n";
    var oRDN = oCert.subjectDN;
    if( ue.email )
    {
        mailList[g] = ue.email;
    }
    else
    if ( oRDN.e )
    {
        mailList[g] = oRDN.e;
    }
}
var result = app.alert( cMsg );

Example 3

List all the entries in an array of groups
var groups = security.chooseRecipientsDialog(oOptions);
for (g in groups) {
  console.println( "Group No. " + g);
  // Permissions
  var perms = groups[g].permissions;
  console.println( "Permissions:");
  for(p in perms) console.println( p + " = " + eval("perms." +p));
  // User Entities
  for(u in groups[g].userEntities) {
    var user = groups[g].userEntities[u];
    console.println( "User No. " + u);
    for(i in user) console.println( i + " = " + eval("user." +i));
  }
}

getHandler

Obtains a SecurityHandler Object. The caller can create as many new engines as desired and each call to getHandler creates a new engine; however, there is only one UI engine.

NOTE: (Security): This method is available from batch, console, app initialization and menu events. It is also available in the Adobe Reader

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The language independent name of the security handler, as returned by the handlers property.</td>
</tr>
<tr>
<td>bUIEngine</td>
<td>(optional) If true, the method returns the existing security handler instance that is associated with the Acrobat user interface (so that, for example, a user can log in via the user interface). If false (the default), returns a new engine.</td>
</tr>
</tbody>
</table>

Returns

The SecurityHandler Object specified by cName. If the handler is not present, returns a null object.

Example

This code selects the Adobe.PPKLite SecurityHandler.

```javascript
// 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/profiles/DPSmith.pfx");

See also the example following `signatureSign` for a continuation of this example.

**exportToFile**

Exports a Certificate Object to a local disk as a raw certificate file.

**NOTE:** (Security  sổ): Data being written must be data for a valid certificate; arbitrary data types cannot be written. This method will not overwrite an existing file.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oObject</code></td>
<td>The Certificate Object that is to be exported to disk.</td>
</tr>
<tr>
<td><code>cDIPath</code></td>
<td>The device-independent save path. <strong>NOTE:</strong> (Security  sổ): The parameter <code>cDIPath</code> must be Safe Path and must end with the extension <code>.cer</code>.</td>
</tr>
</tbody>
</table>

### Returns

The path of the file that was written, if successful.

### Example

```javascript
var outPath = security.exportToFile(oCert, "/c/outCert.cer");
```

**importFromFile**

Reads a raw data file and returns the data as an object with a type specified by `cType`. The file being imported must be a valid certificate.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cType</code></td>
<td>The type of object to be returned by this method. The only supported type is “Certificate”.</td>
</tr>
<tr>
<td><code>cDIPath</code></td>
<td>(optional) When <code>bUI</code> is <code>false</code>, this parameter is a required and specifies the device-independent path to the file to be opened. If <code>bUI</code> is <code>true</code>, this is the seed path used in the open dialog.</td>
</tr>
</tbody>
</table>
SecurityHandler Object

SecurityHandler objects are used to access security handler capabilities such as signatures, encryption and directories. Different security handlers will have different properties and methods. This section documents the full set of properties and methods that security objects may have. Individual SecurityHandler objects may or may not implement these properties and methods.

SecurityHandler objects can be obtained using the `security.getHandler` method.

The JavaScript interface for `Adobe.PPKLite` signatures was introduced in Acrobat 5.0, with the remainder of the JavaScript interface being introduced in Acrobat 6.0. Prior to Acrobat 6.0 there was no support in Acrobat to enable JavaScript in third party security handlers.

Not all security handlers are JavaScript enabled. Not all JavaScript enabled handlers are enabled for all security operations. Third party public key security handlers may support JavaScript, but only if they use the new `PubSec` programming interface that was introduced in Acrobat 6.0.

JavaScript enabled handlers provided by Adobe include: the `Adobe.PPKLite` security handler, supporting signature and encryption; the `Adobe.PPKMS` security handler for the Windows operating system supporting signatures, encryption and directory access through the `Microsoft Active Directory Scripting Interface` (ADSI); and the `Adobe.AAB` security handler providing a local address book and support for directory operations. Note that the `Standard` security handler, used for password encryption of documents, is not JavaScript enabled.

**NOTE:** (Security ®): SecurityHandler Objects can only be created using the Security Object `getHandler` method. This method is available only for batch, console, application init and menu exec, and is available in the Adobe Reader.

<table>
<thead>
<tr>
<th>bUI</th>
<th>(optional) true if the user should be prompted to select the file that is to be imported. The default is false.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cMsg</td>
<td>(optional) If bUI is true, the title to use in the open dialog. If cMsg is not specified, the default title is used for the dialog.</td>
</tr>
</tbody>
</table>

**Returns**

A Certificate Object.

**Example**

```javascript
var oMyCert = security.importFromFile("Certificate", "/c/myCert.cer");
```
SecurityHandler Properties

appearances

An array containing the language-dependent names of the available user-configured appearances for the specified security handler. Appearances are used to create the on-page visual representation of a signature when signing a signature field. The name of an appearance can be specified as a signature info object property when signing a signature field using `field.signatureSign`.

Acrobat provides a standard signature appearance module that is used by Adobe signature plug-ins and that can also be used by third party signature plug-ins. This standard signature appearance module is pre-configured with one appearance and can be configured by users to contain more appearances. The name of the one pre-configured appearance, called `Standard Text` in the user interface, is not returned by this property.

If a security handler does not support selection of appearances then this property will return null.

_Type: Array Access: R._

digitalIDs

This method returns the certificates that are associated with the currently selected Digital IDs for this security handler.

_Type: Object Access: R._

The return value is a generic object with the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oEndUserSignCert</code></td>
<td>Certificate Object</td>
<td>The certificate that is associated with the currently selected Digital IDs that is to be used by this security handler object when signing. The property is undefined if there is no current selection.</td>
</tr>
</tbody>
</table>
The *Adobe.PPKLite* security handler returns the list of all Digital IDs associated with the currently accessed password-protected Digital ID file. This handler requires that the Security Handler Object has gained access to a password-protected Digital ID file before this property can return a value. Access is obtained either by logging in via the user interface and using the Security Object `getHandler` method with `bUIEngine` equal to *true*, or by using the `login` method. Both `oEndUserSignCert` and `oEndUserCryptCert` properties can be set using the user-interface, and then these settings are stored in the Digital ID file. `oEndUserSignCert` can also be set using the `login` method.

The *Adobe.PPKMS* handler returns all currently available Digital IDs in the Windows Digital ID store. Both `oEndUserSignCert` and `oEndUserCryptCert` properties can be set using the user-interface. `oEndUserSignCert` can also be set using the `login` method. This means that `oEndUserCryptCert` will only be returned when using a Security Handler object that is obtained using the `getHandler` method with `bUIEngine` set to *true*.

**Example**

```javascript
var sh = security.getHandler( "Adobe.PPKMS", true );
var ids = sh.digitalIDs;
var oCert = ids.oEndUserSignCert;
security.exportToFile( oCert, "/c/MySigningCert.cer" );
```

**directories**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oEndUserCryptCert</code></td>
<td>Certificate Object</td>
<td>The certificate that is associated with the currently selected Digital IDs that is to be used when encrypting a document with this security handler object. The property is undefined if there is no current selection.</td>
</tr>
<tr>
<td><code>certs</code></td>
<td>Array of Certificate Objects</td>
<td>An array of certificates corresponding to the list of all Digital IDs that are available for this security handler object.</td>
</tr>
</tbody>
</table>

The *Adobe.PPKLite* security handler returns the list of all Digital IDs associated with the currently accessed password-protected Digital ID file. This handler requires that the Security Handler Object has gained access to a password-protected Digital ID file before this property can return a value. Access is obtained either by logging in via the user interface and using the Security Object `getHandler` method with `bUIEngine` equal to *true*, or by using the `login` method. Both `oEndUserSignCert` and `oEndUserCryptCert` properties can be set using the user-interface, and then these settings are stored in the Digital ID file. `oEndUserSignCert` can also be set using the `login` method.

The *Adobe.PPKMS* handler returns all currently available Digital IDs in the Windows Digital ID store. Both `oEndUserSignCert` and `oEndUserCryptCert` properties can be set using the user-interface. `oEndUserSignCert` can also be set using the `login` method. This means that `oEndUserCryptCert` will only be returned when using a Security Handler object that is obtained using the `getHandler` method with `bUIEngine` set to *true*.

**Example**

```javascript
var sh = security.getHandler( "Adobe.PPKMS", true );
var ids = sh.digitalIDs;
var oCert = ids.oEndUserSignCert;
security.exportToFile( oCert, "/c/MySigningCert.cer" );
```

**directories**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oEndUserCryptCert</code></td>
<td>Certificate Object</td>
<td>The certificate that is associated with the currently selected Digital IDs that is to be used when encrypting a document with this security handler object. The property is undefined if there is no current selection.</td>
</tr>
<tr>
<td><code>certs</code></td>
<td>Array of Certificate Objects</td>
<td>An array of certificates corresponding to the list of all Digital IDs that are available for this security handler object.</td>
</tr>
</tbody>
</table>
directoryHandlers

Returns an array containing the language independent names of the available directory handlers for the specified security handler. As an example, the Adobe.PPKMS security handler has a directory handler named Adobe.PPKMS.ADSI that supports queries using the Microsoft Active Directory Script Interface (ADSI). Valid directory handler names are required when activating a new Directory Object using its info property.

Type: Array Access: R.

isLoggedln

Returns true if currently logged into this SecurityHandler Object. See the login method.

Different security handlers will have their own rules for determining the value of this property. The Adobe.PPKLite handler will return true if a user is logged in to a profile file (also called credential file, implemented as a PKCS#12 file). Adobe.PPKMS will always return true.

Type: Boolean Access: R.

Example

```javascript
var ppklite = security.getHandler("Adobe.PPKLite", true);
console.println( "Is logged in = " + ppklite.isLoggedln ); // false
ppklite.login( "dps017", "/C/signatures/DPSmith.pfx" );
console.println( "Is logged in = " + ppklite.isLoggedln ); // true
```

loginName

The name associated with the actively selected signing Digital ID for the security handler. This may require that the login method be called in order to select a signing credential. The return value is null if a signing credential is not selected or if the security handler does not support this property.

Type: String Access: R.
**loginPath**

5.0 S S

The device-independent path to the user’s profile file used to login to the security handler. The return value is null if no one is logged in, if the security handler does not support this property, or if this property is irrelevant for the currently logged in user.

*Type: String  Access: R.*

**name**

5.0 S S

The language-independent name of the security handler. Example values for the Default Certificate, Windows Certificate, and Entrust Security Handlers are `Adobe.PPKLite`, `Adobe.PPKMS`, and `Entrust.PPKEF`. All security handlers must support this property.

*Type: String  Access: R.*

**signAuthor**

6.0 S S

Whether the security handler is capable of generating certified documents. A certified document is a document that is signed with both a byte range signature and an object signature. Object signatures are generated by walking the object tree of the document and are used to detect and prevent modifications to a document. Refer to the mdp property of the `SignatureInfo Object` for details regarding modification detection and prevention (MDP) settings.

*Type: Boolean  Access: R.*

**signFDF**

6.0 S S

Indicates that the security handler is capable of signing FDF files.

*Type: Boolean  Access: R.*
signInvisible

Whether the security handler is capable of generating invisible signatures.

*Type: Boolean  Access: R.*

signValidate

Indicates whether the security handler is capable of validating signatures.

*Type: Boolean  Access: R.*

signInvisible

Whether the security handler is capable of generating visible signatures.

*Type: Boolean  Access: R.*

uiName

The language-dependent string for the security handler. This string is suitable for user interfaces. All security handlers must support this property.

*Type: String  Access: R.*

SecurityHandler Methods

login

This method provides a mechanism by which Digital IDs can be accessed and selected for a particular Security Handler. Parameters tend to be specific to a particular handler. The behaviour for Adobe.PPKLite and Adobe.PPKMS handlers is specified below.
The parameters `cPassword` and `cDIPath` for backward compatibility, or included as properties of the `oParams` object, which is the preferred calling convention beginning in Acrobat 6.0.

See also `logout`, `newUser`, and `loginName`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cPassword</code></td>
<td>(optional) The password necessary to access the password-protected Digital ID. This parameter is supported by <code>Adobe.PPKLite</code> for accessing Digital ID files.</td>
</tr>
<tr>
<td><code>cDIPath</code></td>
<td>(optional) A device independent path to the password-protected Digital ID file. This parameter is supported by <code>Adobe.PPKLite</code>. <strong>Note:</strong> (Version 6.0) When logging, the user's digital signature profile must be a <code>.pfx</code> file, not an <code>.apf</code>, as in prior versions of Acrobat. To convert an <code>.apf</code> profile to the new <code>.pfx</code> type, use the UI (<strong>Advanced &gt; Manage Digital IDs &gt; My Digital ID Files &gt; Select My Digital ID File</strong>) to import the <code>.apf</code> profile.</td>
</tr>
<tr>
<td><code>oParams</code></td>
<td>(optional, version 6.0) A <code>LoginParameters</code> Generic Object with parameters that are specific to a particular SecurityHandler Object. The common fields in this object are described below. These fields include the <code>cDIPath</code> and <code>cPassword</code> values, thus allowing the parameter list to be expressed in different ways.</td>
</tr>
<tr>
<td><code>bUI</code></td>
<td>(optional, version 6.0) Set to <code>true</code> if it is desired that user interface be used to log the user in. This attribute should be supported by all security handlers that support this method.</td>
</tr>
</tbody>
</table>

**Returns**

Returns `true` if the login succeeded, `false` otherwise.

**LoginParameters Generic Object**

This generic JS object contains parameters for the `login` method. It has the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cDIPath</code></td>
<td>String</td>
<td>The path to a file that contains the Digital ID. This file is normally password protected. Supported by <code>Adobe.PPKLite</code> security handler.</td>
</tr>
</tbody>
</table>
**Example 1**

```javascript
// Use "Adobe.PPKLite" Security Handler Object for the UI
var ppklite = security.getHandler( "Adobe.PPKLite", true );
var oParams = { cPassword: "dps017", cDlPath: "/C/DPSmith.pfx" }
ppklite.login( oParams );
<..... make a signature field and sign it ......>
ppklite.logout();

// PPKLite - Use UI to select a credential, when already logged in
ppklite.login({
  oParams:
    { oEndUserSignCert: {},
      cMsg: "Select your Digital ID",
      bUI : true
    } });

// PPKLite - Login and select signing credential
var oCert = { SHA1Hash: "00000000" }
ppklite.login({
  oParams:
    { cDlPath: "/C/test/DPSmith.pfx",
      cPassword: "dps017",
      oEndUserSignCert: oCert,
      cMsg: "Select your Digital ID"
  } );
```

---

### SecurityHandler Methods

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cPassword</td>
<td>String</td>
<td>A password that is used to authenticate the user. This password may used to access a password-protected Digital ID file. Supported by Adobe.PPKLite security handler. Note that Acrobat does not guarantee that this password is obfuscated in memory.</td>
</tr>
<tr>
<td>oEndUserSignCert</td>
<td>generic object</td>
<td>Selects a Digital ID for the purpose of performing end user signing. The value of this property is a Certificate Object, or generic object with the same property names as a Certificate Object, defining the certificate that is being selected. It may or may not be necessary to call this method for a particular handler. For example, if logged in to a PKCS#12 file containing one signing Digital ID with Adobe.PPKLite, a signing credential will not need to be selected. All security handlers must be able to process the binary and SHA1Hash properties of this object. This object can be empty if bUI is true.</td>
</tr>
<tr>
<td>cMsg</td>
<td>String</td>
<td>A message to display in the login dialog, if bUI is true.</td>
</tr>
</tbody>
</table>

---

*Acrobat JavaScript Scripting Reference*
Example 2

```javascript
// Use "Adobe.PPKMS" Security Handler Object
var ppkms = security.getHandler( "Adobe.PPKMS" );

// Select credential to use when signing
var oCert = myCerts[0];
ppkms.login( { oParams: { oEndUserSignCert: oCert } } );
```

See `signatureSign` for details on signing a PDF document.

### logout

**Parameters**

None

**Returns**

Beginning in Acrobat 6.0, returns `true` if the logout succeeded, `false` otherwise. Previous Acrobat releases did not generate a return value.

### newDirectory

**Parameters**

None

**Returns**

Returns a new `Directory Object`. The directory object must be activated using its `info` property before it is marked for persistence and can be used for searches. Existing directory objects can be discovered using the `directories` property.
newUser

This method supports enrollment with Adobe.PPKLite and Adobe.PPKMS security handlers by creating a new self-sign credential.

NOTE: (Security ☑): This method will not allow the user to overwrite an existing file.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cPassword</td>
<td>(optional) The password necessary to access the password-protected Digital ID file. This parameter is ignored by Adobe.PPKMS.</td>
</tr>
<tr>
<td>cDIPath</td>
<td>(optional) The device-independent path to the password-protected Digital ID file. This parameter is ignored by Adobe.PPKMS. &lt;br&gt;Note: (Security ☑): Beginning with Acrobat 6.0, the parameter cDIPath must be Safe Path and end with the extension .pfx.</td>
</tr>
<tr>
<td>oRDN</td>
<td>(optional) The relative distinguished name (RDN) as an RDN Generic Object containing the issuer or subject name for a certificate. The only required field is cn. If the country c is provided, it must be two characters, using the ISO 3166 standard (for example, 'US').</td>
</tr>
<tr>
<td>oCPS</td>
<td>(optional, version 6.0) A generic object containing certificate policy information that will be embedded in the Certificate Policy extension of the certificate. The object must contain property oid, which indicates the certificate policy object identifier. The other properties which may be present are url and (user) notice. The url is a URL that points to detailed information about the policy under which the certificate has been issued and user notice is a abridged version of the same, embedded in the certificate.</td>
</tr>
<tr>
<td>bUI</td>
<td>(optional, version 6.0) When true, the user interface can be used to enroll. This parameter is supported by all security handlers that support this method.</td>
</tr>
</tbody>
</table>

Returns

true if successful, throws an exception if not successful.

Example

// Create a new PPKLite self-sign credential (Acrobat 5.0 syntax) <br>var ppklite = security.getHandler("Adobe.PPKLite"); <br>var oRDN = { cn: "Fred NewUser", c: "US" };<br>var oCPS = {oid: "1.2.3.4.5",<br>    url: "http://www.myca.com/mycps.html",<br>    notice: "This is a self generated certificate, hence the "<br>        + "recipient must verify it’s authenticity through an out "
```javascript
+ "of band mechanism" });
ppklite.newUser( "testtest", "/d/temp/FredNewUser.pfx", oRDN, oCPS);

// Alternate generic object syntax, allowing additional parameters
var oParams = {
    cPassword : "myPassword",
    cDIPath : "/d/temp/FredNewUser.pfx",
    oRDN : oRDN,
    oCPS : oCPS,
    bUI : false
};
ppklite.newUser( oParams );

// Use a certificate from an existing signed, field to create the RDN
var f = this.getField( "mySignature" );
f.signatureValidate();
var sigInfo = f.signatureInfo();
var certs = sigInfo.certificates;
var oSubjectDN = certs[0].subjectDN;

ppklite.newUser({
    cPassword: "dps017",
    cDIPath: "/c/temp/DPSmith.pfx",
    oRDN: oSubjectDN
});
```

### setPasswordTimeout

Sets the number of seconds after which password should expire between signatures. This method is only supported by the Adobe.PPKLite security handler. For this handler the default timeout value for a new user is 0 (password always required).

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cPassword</strong></td>
<td>The password needed to set the timeout value.</td>
</tr>
<tr>
<td><strong>iTimeout</strong></td>
<td>The timeout value, in seconds. Set to 0 for always expire (that is, password always required). Set to 0x7FFFFFFF for never expire.</td>
</tr>
</tbody>
</table>

#### Returns

Throws an exception if the user has not logged in to the Adobe.PPKLite Security Handler, or unsuccessful for any other reason.
Example

This example logs in to the `PPKLite` security handler and sets the password timeout to 30 seconds. If the password timeout has expired—30 seconds in this example—the signer must provide a password. The password is not necessary if the password has not timed out.

```javascript
var ppklite = security.getHandler( "Adobe.PPKLite" );
ppklite.login( "dps017", "/d/profiles/DPSmith.pfx" );
ppklite.setPasswordTimeout( "dps017", 30 );
```

SignatureInfo Object

A generic JS object that contains the properties of a digital signature. Some properties are supported by all handlers, and additional properties can be supported. Writable properties can be specified when signing the object.

This object is returned by `field.signatureInfo` and `FDF.signatureValidate`, and passed to `field.signatureSign` and `FDF.signatureSign`.

SignatureInfo Object properties

All handlers define the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buildInfo</td>
<td>Object</td>
<td>R</td>
<td>6.0</td>
<td>An object containing software build and version information for the signature. The format of this object is not described in this document. An Acrobat technote may be produced that contains this information. The subject of this technote will be signature build properties dictionary.</td>
</tr>
<tr>
<td>date</td>
<td>Date</td>
<td>R</td>
<td>5.0</td>
<td>The date and time that the signature was created, in PDF date format.</td>
</tr>
</tbody>
</table>
### SignatureInfo Object properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>handlerName</strong></td>
<td>String</td>
<td>R</td>
<td>5.0</td>
<td>The language independent name of the security handler that was specified as the Filter attribute in the signature dictionary. This is usually the name of the security handler that created the signature, but can also be the name of the security handler that the creator desires to be used when validating the signature.</td>
</tr>
<tr>
<td><strong>handlerUserName</strong></td>
<td>String</td>
<td>R</td>
<td>5.0</td>
<td>The language independent name corresponding to security handler specified by handlerName. This is only available when the named security handler is available.</td>
</tr>
<tr>
<td><strong>handlerUIName</strong></td>
<td>String</td>
<td>R</td>
<td>5.0</td>
<td>The language dependent name corresponding to security handler specified by handlerName. This is only available when the named security handler is available.</td>
</tr>
<tr>
<td><strong>location</strong></td>
<td>String</td>
<td>R/W</td>
<td>5.0</td>
<td>Optional user specified location when signing. This can be a physical location (such as a city) or hostname.</td>
</tr>
<tr>
<td><strong>mdp</strong></td>
<td>String</td>
<td>R/W</td>
<td>6.0</td>
<td>The Modification Detection and Prevention (MDP) setting that was used to sign the field or FDF Object being read, or the MDP setting to use when signing. Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;allowNone&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;allowAll&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;default&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot;defaultValueAndComments&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>See Modification Detection and Prevention (MDP) Values for details.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>String</td>
<td>R</td>
<td>5.0</td>
<td>Name of the user that created the signature.</td>
</tr>
</tbody>
</table>
### SignatureInfo Object properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>numFieldsAltered</td>
<td>Number</td>
<td>R</td>
<td>5.0</td>
<td>Number of fields altered between the previous signature and this signature. Used only for signature fields.</td>
</tr>
<tr>
<td>numFieldsFilledIn</td>
<td>Number</td>
<td>R</td>
<td>5.0</td>
<td>Number of fields filled-in between the previous signature and this signature. Used only for signature fields.</td>
</tr>
<tr>
<td>numPagesAltered</td>
<td>Number</td>
<td>R</td>
<td>5.0</td>
<td>Number of pages altered between the previous signature and this signature. Used only for signature fields.</td>
</tr>
<tr>
<td>numRevisions</td>
<td>Number</td>
<td>R</td>
<td>5.0</td>
<td>The number of revisions in the document. Used only for signature fields.</td>
</tr>
<tr>
<td>reason</td>
<td>String</td>
<td>R/W</td>
<td>5.0</td>
<td>User specified reason for signing.</td>
</tr>
<tr>
<td>revision</td>
<td>Number</td>
<td>R</td>
<td>5.0</td>
<td>The signature revision to which this signature field corresponds. Used only for signature fields.</td>
</tr>
<tr>
<td>status</td>
<td>Number</td>
<td>R</td>
<td>5.0</td>
<td>The validity status of the signature, computed during the last call to the signatureValidate. Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1: Not a signature field</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0: Signature is blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1: Unknown status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2: Signature is invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3: Signature of document is valid, identity of signer could not be verified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4: Signature of document is valid and identity of signer is valid.</td>
</tr>
<tr>
<td>statusText</td>
<td>String</td>
<td>R</td>
<td>5.0</td>
<td>The language dependent text string, suitable for user display, denoting the signature validity status, computed during the last call to the signatureValidate.</td>
</tr>
</tbody>
</table>
### SignatureInfo Object properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subFilter</td>
<td>String</td>
<td>R/W</td>
<td>6.0</td>
<td>The format to use when signing. Consult the PDF Reference for a complete list of supported values. The known values used for public key signatures include adbe.pkcs7.sha1, adbe.pkcs7.detached, and adbe.x509.rsa_sha1. It is important that the caller know that a particular signature handler can support this format.</td>
</tr>
<tr>
<td>verifyHandlerName</td>
<td>String</td>
<td>R</td>
<td>6.0</td>
<td>The language independent name of the security handler that was used to validate this signature. This will be <code>null</code> if the signature has not been validated, that is, if the status property has a value of 1</td>
</tr>
<tr>
<td>verifyHandlerUIName</td>
<td>String</td>
<td>R</td>
<td>6.0</td>
<td>The language dependent name corresponding to security handler specified by verifyHandlerName. This will be null if the signature has not been validated, that is, if the status property has a value of 1.</td>
</tr>
</tbody>
</table>

### SignatureInfo Object Public Key Security Handler Properties

Public key security handlers may define the following additional properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appearance</td>
<td>String</td>
<td>W</td>
<td>5.0</td>
<td>The name of the user-configured appearance to use when signing this field. PPKLite and PPKMS use the standard appearance handler, and in this situation, the appearance names can be found in the signature appearance configuration dialog of the user interface (menu Edit &gt; Preferences &gt; Digital Signatures in Acrobat 6.0). The default, when not specified, is to use the Standard Text appearance. Used only for visible signature fields.</td>
</tr>
</tbody>
</table>
### SignatureInfo Object Public Key Security Handler Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>certificates</td>
<td>Array</td>
<td>R</td>
<td>5.0</td>
<td>Array containing a hierarchy of certificates that identify the signer. The first element in the array is the signer's certificate, and subsequent elements include the chain of certificates up to the certificate authority that issued the signer's certificate. For self-signed certificates this array will contain only one entry.</td>
</tr>
<tr>
<td>contactInfo</td>
<td>String</td>
<td>R/W</td>
<td>5.0</td>
<td>User specified contact information for determining trust. For example, a telephone number that recipients of a document can use to contact the author to establish trust. This is not recommended for a scalable solution for establishing trust.</td>
</tr>
<tr>
<td>byteRange</td>
<td>Array</td>
<td>R</td>
<td>6.0</td>
<td>An array of numbers indicating the bytes that are covered by this signature.</td>
</tr>
<tr>
<td>docValidity</td>
<td>Number</td>
<td>R</td>
<td>6.0</td>
<td>The validity status of the document byte range digest portion of the signature, computed during the last call to <code>signatureValidate</code>. All PDF document signature field signatures include a byte range digest. See Validity Values for details of the return codes.</td>
</tr>
<tr>
<td>idPrivValidity</td>
<td>Number</td>
<td>R</td>
<td>6.0</td>
<td>Returns the validity of the identity of the signer. This value is specific to the handler. See Private Validity Values for values supported by the Adobe.PPKLite and Adobe.PPKMS handlers. This value is 0 unless the signature has been validated, that is, if the <code>status</code> property has a value of 1.</td>
</tr>
</tbody>
</table>
### SignatureInfo Object Public Key Security Handler Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>idValidity</td>
<td>Number</td>
<td>R</td>
<td>6.0</td>
<td>Returns the validity of the identity of the signer as number. Values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-1: Not a signature field</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0: Signature is blank</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1: Unknown status</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2: Signature is invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3: Signature of document is valid, identity of signer could not be verified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4: Signature of document is valid and identity of signer is valid.</td>
</tr>
<tr>
<td>objValidity</td>
<td>Number</td>
<td>R</td>
<td>6.0</td>
<td>The validity status of the object digest portion of the signature, computed during the last call to <code>signatureValidate</code>. For PDF documents, signature field author signatures and document-level application rights signatures include object digests. All FDF files are signed using object digests. See Validity Values for details of the return codes.</td>
</tr>
<tr>
<td>trustFlags</td>
<td>Number</td>
<td>R</td>
<td>6.0</td>
<td>The bits in this number indicate what the signer is trusted for. The value is valid only when the value of the status property is 4. These trust settings are derived from trust setting in the recipient's trust database, for example the Acrobat Address Book (Adobe.AAB). Bit assignments are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1- trusted for signatures</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2- trusted for certifying documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3- trusted for dynamic content such as multimedia</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4- Adobe internal use</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5- the javascript in the PDF file is trusted to operate outside the normal PDF restrictions</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>W</td>
<td>5.0</td>
<td>Password required as authentication when accessing a private key that is to be used for signing. This may or may not be required, dependent on the policies of the security handler.</td>
</tr>
</tbody>
</table>
Validity Values

The following codes are returned by the `docValidity` and `objValidity` (See SignatureInfo Object Public Key Security Handler Properties), allowing a finer granularity of understanding of the validity of the signature than the `status` property.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kDSSigValUnknown</td>
<td>Validity not yet determined.</td>
</tr>
<tr>
<td>kDSSigValUnknownTrouble</td>
<td>Validity could not be determined because of errors encountered during the validation process.</td>
</tr>
<tr>
<td>kDSSigValUnknownBytesNotReady</td>
<td>Validity could not be determined because all bytes are not available, for example when viewing a file in a web browser. Even when bytes are not immediately available, this value may not be returned if the underlying implementation blocks when bytes or not ready. Adobe makes no commitment regarding whether validation checks will block or not block, however the implementation in Acrobat 6.0 will block when validating docValidity and not block when validating objValidity.</td>
</tr>
<tr>
<td>kDSSigValInvalidTrouble</td>
<td>Validity for this digest was not computed because there were errors in the formatting or information contained in this signature. There is sufficient evidence to conclude that the signature is invalid.</td>
</tr>
<tr>
<td>kDSSigValInvalidTrouble</td>
<td>Validity for this digest is not used (e.g., no doc validity if no byte range).</td>
</tr>
<tr>
<td>kDSSigValJustSigned</td>
<td>The signature was just signed, so implicitly valid.</td>
</tr>
<tr>
<td>kDSSigValFalse</td>
<td>The digest or validity is invalid</td>
</tr>
<tr>
<td>kDSSigValTrue</td>
<td>The digest or validity is valid</td>
</tr>
</tbody>
</table>

Private Validity Values

Verification of the validity of the signer's identity is specific to the handler that is being used to validate the identity. This value may contain useful information regarding an identity. The identity is returned in the `idPrivValidity` property. Values for Adobe.PPKMS and
Adobe.PPKLite security handlers are shown here. This value is also mapped to an idValidity value that is common across all handlers.

<table>
<thead>
<tr>
<th>Status Code</th>
<th>idValidity Mapping</th>
<th>Security Handler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kIdUnknown</td>
<td>1 (unknown)</td>
<td>PPKMS, PPKLite</td>
<td>Validity not yet determined.</td>
</tr>
<tr>
<td>kIdTrouble</td>
<td>1 (unknown)</td>
<td>PPKMS, PPKLite</td>
<td>Could not determine validity because of errors, for example internal errors, or could not build the chain, or could not check basic policy.</td>
</tr>
<tr>
<td>kIdInvalid</td>
<td>2 (invalid)</td>
<td>PPKMS, PPKLite</td>
<td>Certificate is invalid: not time nested, invalid signature, invalid/unsupported constraints, invalid extensions, chain is cyclic.</td>
</tr>
<tr>
<td>kIdNotTimeValid</td>
<td>2 (invalid)</td>
<td>PPKMS, PPKLite</td>
<td>Certificate is outside its time window (too early, too late).</td>
</tr>
<tr>
<td>kIdRevoked</td>
<td>2 (invalid)</td>
<td>PPKMS</td>
<td>Certificate has been revoked.</td>
</tr>
<tr>
<td>kIdUntrustedRoot</td>
<td>1 (unknown)</td>
<td>PPKMS, PPKLite</td>
<td>Certificate has an untrusted root certificate.</td>
</tr>
<tr>
<td>kIdBrokenChain</td>
<td>2 (invalid)</td>
<td>PPKMS, PPKLite</td>
<td>Could not build a certificate chain up to a self-signed root certificate.</td>
</tr>
<tr>
<td>kIdPathLenConstraint</td>
<td>2 (invalid)</td>
<td>PPKLite</td>
<td>Certificate chain has exceeded the specified length restriction. The restriction was specified in Basic Constraints extension of one of the certificates in the chain.</td>
</tr>
<tr>
<td>kIdCriticalExtension</td>
<td>1 (unknown)</td>
<td>PPKMS</td>
<td>One of the certificates in the chain has an unrecognized critical extension.</td>
</tr>
<tr>
<td>kIdJustSigned</td>
<td>4 (valid)</td>
<td>PPKMS, PPKLite</td>
<td>Just signed by user (similar to kIdIsSelf)</td>
</tr>
<tr>
<td>kIdAssumedValid</td>
<td>3 (idunknown)</td>
<td>PPKMS</td>
<td>Certificate is valid to a trusted root, but revocation could not be checked and was not required.</td>
</tr>
<tr>
<td>kIdIsSelf</td>
<td>4 (valid)</td>
<td>PPKMS, PPKLite</td>
<td>Certificate is my credential (no further checking was done).</td>
</tr>
</tbody>
</table>
Modification Detection and Prevention (MDP) Values

Modification detection and prevention (MDP) settings control what changes are allowed to occur in a document before the signature becomes invalid. Changes are recorded outside of the byte range, for signature fields, and can include changes that have been incrementally saved as part of the document or changes that have occurred in memory between the time that a document is opened and when the signature is validated. MDP settings may only be applied to the first signature in a document. Use of MDP will result in an author signature. MDP has one of the following four values:

- **allowAll**: Allow all changes to a document without any of these changes invalidating the signature. This results in MDP not being used for the signature. This was the behavior for Acrobat 4.0 through 5.1.
- **allowNone**: Do not allow any changes to the document without invalidating the signature. Note that this will also lock down the author’s signature.
- **default**: Allow form field fill-in if form fields are present in the document, otherwise do not allow any changes to the document without invalidating the signature.
- **defaultAndComments**: Allow form field fill-in if form fields are present in the document, and allow annotations (comments) to be added, deleted or modified, otherwise do not allow any changes to the document without invalidating the signature. Note that annotations can be used to obscure portions of a document and thereby affect the visual presentation of the document.

### SOAP Object

The SOAP object allows remote procedure calls to be made to a remote server from JavaScript. The SOAP 1.1 protocol (see [http://www.w3.org/TR/SOAP/](http://www.w3.org/TR/SOAP/)) is used to marshall JavaScript parameters to a remote procedure call (either synchronously or asynchronously) and to unmarshall the result as a JavaScript object. The SOAP object also has the ability to communicate with Web Services described by the Web Services Description Language (WSDL—see [http://www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl)).

<table>
<thead>
<tr>
<th>Status Code</th>
<th>idValidity Mapping</th>
<th>Security Handler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kIdValid</td>
<td>4 (valid)</td>
<td>PPKMS, PPKLite</td>
<td>Certificate is valid to a trusted root (in the Windows or Acrobat Address Book).</td>
</tr>
<tr>
<td>kIdRevocationUnknown</td>
<td>?</td>
<td>PPKMS, PPKLite</td>
<td>Certificate is valid to a trusted root, but revocation could not be checked and was required by the user.</td>
</tr>
</tbody>
</table>

Private Validity Values

<table>
<thead>
<tr>
<th>Status Code</th>
<th>idValidity Mapping</th>
<th>Security Handler</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kIdValid</td>
<td>4 (valid)</td>
<td>PPKMS, PPKLite</td>
<td>Certificate is valid to a trusted root (in the Windows or Acrobat Address Book).</td>
</tr>
<tr>
<td>kIdRevocationUnknown</td>
<td>?</td>
<td>PPKMS, PPKLite</td>
<td>Certificate is valid to a trusted root, but revocation could not be checked and was required by the user.</td>
</tr>
</tbody>
</table>
NOTE: SOAP methods `connect`, `request` and `response` are available only for documents open in Acrobat Pro and Acrobat Std., and for documents with Form Export Rights(🔗) open in Adobe Reader 6.0.

### SOAP Properties

**wireDump**

If `true`, synchronous SOAP requests will cause the XML Request and Response to be dumped to the JavaScript Console. This is useful for debugging SOAP problems.

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

### SOAP Methods

**connect**

Takes the URL of a WSDL document (`cURL`) and converts it to a JavaScript object with callable methods corresponding to the web service.

The parameters to the method calls and the return values obey the rules specified for the `SOAP . request` method.

**Parameters**

| `cURL` | The URL of a WSDL document. The `cURL` parameter must be an HTTP or HTTPS URL. |

**Returns**

The result value from `SOAP . connect` is a WSDL Service Proxy object with a JavaScript method corresponding to each operation in the WSDL document provided at the URL.

The parameters required for the method depend on the WSDL operation you are calling and how the operation encodes its parameters.

If the WSDL operation is using the SOAP RPC encoding (as described in Section 7 of the SOAP 1.1 Specification) then the arguments to the service method are the same as the parameter order in the WSDL document.
If the WSDL service is using the SOAP document/literal encoding then the function will have a single argument indicating the request message. The argument may be a JavaScript object literal describing the message or it may be either a string or a ReadStream Object with an XML fragment describing the message.

The return value of the service method will correspond to the return value of the WSDL operation.

Exceptions

SOAP Faults will cause a SOAPError exception to be thrown. If there is a problem at the networking level, such as an unavailable endpoint, a NetworkError will be thrown.

Example

A service WSDL Document URL is needed. These can be obtained from the "Round 2 Interop Services - using SOAP 1.2" section at the following URL: http://www.whitemesa.com/interop.htm.

```javascript
var cURL = <get a URL for this service from http://www.whitemesa.com/interop.htm>

// Connect to the test service
var service = SOAP.connect(cURL);

// Print out the methods this service supports to the console
for(var i in service) console.println(i);

var cTestString = "This is my test string";

// Call the echoString service -- it is an RPC Encoded method
var result = service.echoString(cTestString);

// This should be the same as cTestString
console.println(result + " == " + cTestString);

// Call the echoInteger service -- JavaScript doesn't support integers
// so we make our own integer object.
var oTestInt =
{
    soapType: "xsd:int",
    soapValue: "10"
};
var result = service.echoInteger(oTestInt);

// This should be the same as oTestInt.soapValue
console.println(result + " == " + oTestInt.soapValue);
```

This produces the following output:

echoBase64
echoBoolean
echoDate
echoDecimal
request

6.0 F

Initiates a remote procedure call (RPC) against the SOAP HTTP endpoint.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cURL</td>
<td>The URL for a SOAP HTTP Endpoint.</td>
</tr>
<tr>
<td>oRequest</td>
<td>An object literal that specifies the remote procedure name and the parameters to call.</td>
</tr>
<tr>
<td>oAsync</td>
<td>(optional) An object literal indicating that the method invocation will occur asynchronously.</td>
</tr>
<tr>
<td>cAction</td>
<td>(optional) The SOAPAction header for the method.</td>
</tr>
<tr>
<td>bEncoded</td>
<td>(optional) Encode the request using the SOAP Encoding described in Section 5 of the SOAP 1.1 specification (the default is to use SOAP Encoding)</td>
</tr>
<tr>
<td>cNamespace</td>
<td>(optional) A namespace for the message schema when not using the SOAP Encoding (the bEncoded flag is false). The default is to have no namespace.</td>
</tr>
</tbody>
</table>

See the **Additional Notes on the Parameters of SOAP.request**.

**Returns**

An object literal. See the **Additional Notes on the Return Value**

**Exceptions**

SOAPError, NetworkError
SOAP Faults will cause a **SOAPError** to be thrown. If there is a problem at the networking level, such as an unavailable endpoint, a **NetworkError** will be thrown.

### Additional Notes on the Parameters of **SOAP.request**

- **cURL** is the URL for a SOAP HTTP Endpoint. The URL method must be one of:
  - http—Connect to a server at a URI on a port. For example, `http://serverName:portNumber/URI`
  - https—Connect to a secured (SSL) server at a URI on a port. For example, `https://serverName:portNumber/URI`

- **oRequest** is an object literal that specifies the remote procedure name and the parameters to call. The object literal uses the fully qualified method name of the remote procedure as the key. The namespace should be separated from the method name by a colon; for example, if the namespace for the method is `http://mydomain/methods` and the method name is `echoString()` then the fully qualified name would be `http://mydomain/methods:echoString`. The value of this key is an object literal, each key is a parameter of the method, and the value of each key is the value of the corresponding parameter of the method. For example:

  ```javascript
  oRequest: {
    "http://soapinterop.org/:echoString":{inputString: "Echo!"}
  }
  ```

When passing parameters to a remote procedure, JavaScript types are bound to SOAP types automatically as listed in the table:

<table>
<thead>
<tr>
<th>JavaScript Type</th>
<th>SOAP Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>xsd:string</td>
</tr>
<tr>
<td>Number</td>
<td>xsd:float</td>
</tr>
<tr>
<td>Date</td>
<td>xsd:dateTime</td>
</tr>
<tr>
<td>Boolean</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td><strong>ReadStream Object</strong></td>
<td>SOAP-ENC:base64</td>
</tr>
<tr>
<td>Array</td>
<td>SOAP-ENC:Array</td>
</tr>
<tr>
<td>Other</td>
<td>No type information</td>
</tr>
</tbody>
</table>


To pass parameters with a non-supported type, the parameter should be passed as an object literal. The keys and description of this object literal follow:

- **soapType:** This is the SOAP Type that will be used for the value when generating the SOAP message; this is useful when a datatype is needed other than the automatic
datatype binding described above. The type should be namespace qualified using the `<namespace>:<type>` notation, for example

   http://mydomain/types:myType

However the `xsd` (the XMLSchema Datatypes namespace), `xsi` (the XMLSchema Instance namespace) and SOAP-ENC (the SOAP Encoding namespace) namespaces are implicitly defined in the SOAP message so the `soapType` can use these, as in `xsd:int` for the XMLSchema Datatype Integer type.

- `soapValue`: This is the value that will be used when generating the SOAP message. It can be a string or a `ReadStream Object`. The `soapValue` is passed unescaped (i.e., will not be XML Entity escaped); for example `"<"` is not converted to `"&lt;"` in the XML Message. Consequently the `soapValue` parameter can be a raw XML fragment which will be passed to the XML Message.

- `soapName`: This is the element name that will be used when generating the SOAP message instead of the key name in the object literal.

For example, integers are not supported in JavaScript, but an integer parameter to a SOAP method can be constructed as follows:

```javascript
var oIntParameter = {
    soapType: "xsd:int",
    soapValue: "1"
};
```

Later, the `oRequest` parameter for the `SOAP.request` method might be

```javascript
oRequest:
    "http://soapinterop.org/:echoInteger": {
        inputInteger: oIntParameter
    }
```

The Example that follows the description of the `SOAP.request` illustrates this technique.

- The `oAsync` object literal must have a function called `response` which will be called with two parameters (`oResult` and `cURI`) when the response returns. `oResult` is the same result object that would have been returned from the request call if it was called synchronously. `cURI` is the URI of the endpoint that the request was made to.

- `cAction` is the SOAPAction header for the method. The SOAPAction is a URN written to an HTTP header used by firewalls and servers to filter SOAP requests. The WSDL file for the SOAP service or the SOAP service description will usually describe the SOAPAction header required (if any).

**ReadStream Object**

A `ReadStream` Object is an object literal that represents a stream of data. The object literal should contain a function called `read`, which takes the form:

```javascript
var readSteamObject = {
    read: function(nBytes) {...}
};
```
The `read()` method takes the number of bytes to read and returns a hex encoded string with the data from the stream. The `read()` method returns a zero length string to indicate end of stream. Alternatively, you can use the `SOAP.streamFromString` function to create a read stream.

**Additional Notes on the Return Value**

If there is no `async` parameter (that is, a synchronous request) then `request` returns the result from the SOAP method. Otherwise, nothing is returned. The SOAP types in the result are mapped to JavaScript types as follows:

<table>
<thead>
<tr>
<th>SOAP Type</th>
<th>JavaScript Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd:string</td>
<td>String</td>
</tr>
<tr>
<td>xsd:integer</td>
<td>Number</td>
</tr>
<tr>
<td>xsd:float</td>
<td>Number</td>
</tr>
<tr>
<td>xsd:dateTime</td>
<td>Date</td>
</tr>
<tr>
<td>xsd:boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>xsd:hexBinary</td>
<td>ReadStream Object</td>
</tr>
<tr>
<td>xsd:base64Binary</td>
<td>ReadStream Object</td>
</tr>
<tr>
<td>SOAP-ENC:base64</td>
<td>ReadStream Object</td>
</tr>
<tr>
<td>SOAP-ENC:Array</td>
<td>Array</td>
</tr>
<tr>
<td>No Type Information</td>
<td>String</td>
</tr>
</tbody>
</table>

**Example**

A service WSDL Document URL is needed. These can be obtained from the "Round 2 Interop Services - using SOAP 1.2" section at the following URL: http://www.whitemesa.com/interop.htm.

```javascript
var cURL = '<get a URL for this service from http://www.whitemesa.com/interop.htm>;

var cTestString = "This is my test string";

// Call the echoString SOAP method -- it is an RPC Encoded method
var response = SOAP.request(
{
    cURL: cURL,
    oRequest: {
        "http://soapinterop.org/:echoString": {
            inputString: cTestString
        }
    },
    cAction: "http://soapinterop.org/
```
```javascript
var result = response["http://soapinterop.org/:echoStringResponse"]["return"];

// This should be the same as cTestString
console.println(result + " == " + cTestString);

// Call the echoInteger SOAP method -- JavaScript doesn't support integers so we make our own integer object.
var oTestInt =
{
    soapType: "xsd:int",
    soapValue: "10"
};

var response = SOAP.request(
{
    cURL: cURL,
    oRequest: {
        "http://soapinterop.org/:echoInteger": {
            inputInteger: oTestInt
        }
    },
    cAction: "http://soapinterop.org/"
});

var result = response["http://soapinterop.org/:echoIntegerResponse"]["return"];

// This should be the same as oTestInt.soapValue
console.println(result + " == " + oTestInt.soapValue);

This produces the following output:

This is my test string == This is my test string
10 == 10

**response**

6.0

Behaves analogously to **request**, however no response is returned. This is useful for sending a message when a reply is not required.
SOAP Methods

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
</table>
| cURL      | The URL for a SOAP HTTP Endpoint. The URL method must be one of:
  - http—Connect to a server at a URI on a port. For example, http://serverName:portNumber/URI
  - https—Connect to a secured (SSL) server at a URI on a port. For example, https://serverName:portNumber/URI |
| oRequest  | An object literal describing the request. It should be specified in the same way as for the request() method. |
| cAction   | (optional) The SOAPAction header for the method. The SOAPAction is a URN written to an HTTP header used by firewalls and servers to filter SOAP requests. The WSDL file for the SOAP service or the SOAP service description will usually describe the SOAPAction header required (if any). |
| bEncoded  | (optional) Encode the request using the SOAP Encoding described in Section 5 of the SOAP 1.1 specification (the default is to use SOAP Encoding) |
| cNamespace| (optional) A namespace for the message schema when not using the SOAP Encoding (the bEncoded flag is false). The default is to have no namespace. |

Returns

Boolean

Exceptions

If there is a problem at the networking level, such as an unavailable endpoint, a NetworkError will be thrown.

Example

See the example that follows the SOAP.request method.

streamDecode

6.0

This function allows the oStream object to be decoded with the specified encoding type, cEncoder. It returns a ReadStream Object (see request) which will have been decoded appropriately. Typically this be would used to access data returned as part of a SOAP method when it was encoded in Base64 or Hex encoding.
streamEncode

This function allows the `oStream` object to be encoded with the specified encoding type, `cEncoder`. It returns a ReadStream Object (see `request`) which will have the appropriate encoding applied. Typically this would be used to pass data as part of a SOAP method when it must be encoded in Base64 or Hex encoding.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>oStream</code></td>
<td>A stream object to be encoded with the specified encoding type.</td>
</tr>
<tr>
<td><code>cEncoder</code></td>
<td>Permissible values for this string are &quot;hex&quot; (for Hex encoded) and &quot;base64&quot; (Base 64 encoded).</td>
</tr>
</tbody>
</table>

**Returns**

ReadStream Object

streamFromString

This function converts a string to a ReadStream Object (see `request`). Typically this would be used to pass data as part of a SOAP method.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cString</code></td>
<td>The string to be converted</td>
</tr>
</tbody>
</table>

**Returns**

ReadStream Object
stringFromStream

This function converts a ReadStream Object (see request) to a string. Typically, this would be used to examine the contents of a stream object returned as part of a response to a SOAP method.

**Parameters**

| oStream | ReadStream Object to be converted. |

**Returns**

String

Sound Object

This object is the representation of a sound that is stored in the document. The array of all sound objects can be obtained from doc.sounds. See also doc methods getSound, importSound, and deleteSound.

Sound Properties

name

The name associated with this sound object.

*Type: String*  
*Access: R.*

**Example**

```javascript
console.println("Dumping all sound objects in this document.");
var s = this.sounds;
for (var i = 0; i < this.sounds.length; i++)
    console.println("Sound[" + i + "]=" + s[i].name);
```
Sound Methods

play

Plays the sound asynchronously.

Parameters
None

Returns
Nothing

pause

Pauses the currently playing sound. If the sound is already paused then the sound play is resumed.

Parameters
None

Returns
Nothing

stop

Stops the currently playing sound.

Parameters
None

Returns
Nothing

Span Object

A span object is used to represent a length of text and its associated properties in a rich text form field or annotation. The rich text value of a form field or annotation consists of an array of span objects representing the text and formatting of the annotation. It is important to note that the span objects are a copy of the rich text value of the field or annotation. Use the field.richValue, event.richValue (and richChange, richChangeEx), or annot.richContents to modify and reset the rich text value to update the field.
Span Properties

alignment

The horizontal alignment of the text. Alignment for a line of text is determined by the first span on the line. The values of **alignment** are

- left
- center
- right

The default value is **left**.

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

The example following **superscript** illustrates the usage of **alignment**.

fontFamily

The font family used to draw the text. It is an array of family names to be searched for in order. The first entry in the array is the font name of the font to use; the second entry is an optional generic family name to use if an exact match of the first font is not found. The generic family names are

- symbol, serif, sans-serif, cursive, monospace, fantasy

The default generic family name is **sans-serif**.

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

**Example**

Set the **defaultStyle** font family for a rich text field.

```javascript
f = this.getField("Text1");
style = f.defaultStyle;

// if Courier Std is not found on the user’s system, use a monospace
style.fontFamily = ["Courier Std", "monospace"];
f.defaultStyle = style;
```

fontStretch

Specifies the normal, condensed or extended face from a font family to be used to draw the text. The values of **fontStretch** are

- ultra-condensed, extra-condensed, condensed, semi-condensed, normal, 
- semi-expanded, expanded, extra-expanded, ultra-expanded

The default value is **normal**.

*Type: String*  
*Access: R/W.*
**fontStyle**

Specifies the text is drawn with an italic or oblique font.

- italic
- normal

The default is `normal`.

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

**fontWeight**

The weight of the font used to draw the text. For the purposes of comparison, normal is anything under 700 and bold is greater than or equal to 700. The values of `fontWeight` are

100, 200, 300, 400, 500, 600, 700, 800, 900

The default value is 400.

*Type: Number*  *Access: R/W.*

**text**

The text within the span.

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

The example following `superscript` illustrates the usage of `text`.

**textColor**

The RGB color to be used to draw the text. The value of `textColor` is a color array, see the Color Object for a description of color array. The default color is black.

*Type: Color Array*  *Access: R/W.*

The example following `superscript` illustrates the usage of `textColor`.

**textSize**

The point size of the text. The value of `textSize` can be any number between 0 and 32767 inclusive. A text size of zero means to use the largest point size that will allow all text data to fit in the field’s rectangle.

The default text size is 12.0.

*Type: Number*  *Access: R/W.*

The example following `field.richValue` illustrates the usage of `textSize`. 

---

**Acrobat JavaScript Scripting Reference**

*Span Properties*
**strikethrough**

If `strikethrough` is `true`, the text is drawn with a strikethrough. The default is `false`.

Type: Boolean          Access: R/W.

**subscript**

Specifies the text is subscript. If `true`, subscript text is drawn with a reduced point size and a lowered baseline. The default is `false`.

Type: Boolean          Access: R/W.

**superscript**

Specifies the text is superscript. If `true`, superscript text is drawn with a reduced point size and a raised baseline. The default is `false`.

Type: Boolean          Access: R/W.

**Example**

Write rich text to a rich text field using various properties. See `field.richValue` for more details and examples.

```javascript
var f = this.getField("myRichField");

// need an array to hold the span objects
var spans = new Array();

// each span object is an object, so we must create one
spans[0] = new Object();
spans[0].alignment = "center";
spans[0].text = "The answer is x";

spans[1] = new Object();
spans[1].text = "2/3";
spans[1].superscript = true;

spans[2] = new Object();
spans[2].superscript = false;
spans[2].text = ". ";

spans[3] = new Object();
spans[3].underline = true;
spans[3].text = "Did you get it right?";
spans[3].fontStyle = "italic";
spans[3].textColor = color.red;
```
// now assign our array of span objects to the field using
// field.richValue
f.richValue = spans;

**underline**

If **underline** is **true**, the text is underlined. The default is **false**.

*Type: Boolean  Access: R/W.*

The example following **superscript** illustrates the usage of **underline**.

---

**Spell Object**

This object allows users to check the spelling of Comments and Form Fields and other spelling domains. To be able to use the **spell** object, the user must have installed the Acrobat Spelling plug-in and the spelling dictionaries.

---

**Spell Properties**

**available**

true if the **spell** object is available.

*Type: Boolean  Access: R.*

**Example**

```javascript
console.println("Spell checking available: " + spell.available);
```

**dictionaryNames**

An array of available dictionary names. A subset of this array can be passed to **check**, **checkText**, and **checkWord**, and to **spellDictionaryOrder** to force the use of a specific dictionary or dictionaries and the order they should be searched.

A listing of valid dictionary names for the user’s installation can be obtained by executing **spell.dictionaryNames** from the console.
Type: Array  Access: R.

**dictionaryOrder**

The dictionary array search order specified by the user on the Spelling Preferences panel. The Spelling plug-in will search for words first in the `doc.spellDictionaryOrder` array if it has been set for the document, and then it will search this array of dictionaries.

Type: Array  Access: R.

**domainNames**

The array of spelling domains that have been registered with the Spelling plug-in by other plug-ins. A subset of this array can be passed to `check` to limit the scope of the spell check.

Depending on the user's installation, valid domains can include:

- Everything
- Form Field
- All Form Fields
- Comment
- All Comments

Type: Array  Access: R.

**languages**

This property returns the array of available ISO 639-2, 3166 language codes. A subset of this array can be passed to the `check`, `checkText`, `checkWord`, and `customDictionaryCreate` methods, and to the `doc.spellLanguageOrder` property to force the use of a specific language or languages and the order they should be searched.

Type: Array  Access: R.

Depending on the user's installation, valid language codes can include:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ca</td>
<td>Catalan</td>
<td>it</td>
<td>Italian</td>
</tr>
</tbody>
</table>
### Spell Properties

#### languageOrder

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>da</td>
<td>Danish</td>
<td>no</td>
<td>Norwegian</td>
</tr>
<tr>
<td>nl</td>
<td>Dutch</td>
<td>nn</td>
<td>Nynorsk</td>
</tr>
<tr>
<td>en</td>
<td>English</td>
<td>pl</td>
<td>Polish</td>
</tr>
<tr>
<td>en-GB</td>
<td>English – UK</td>
<td>pt</td>
<td>Portuguese</td>
</tr>
<tr>
<td>fi</td>
<td>Finish</td>
<td>pt-BR</td>
<td>Portuguese-Brazilian</td>
</tr>
<tr>
<td>fr</td>
<td>French</td>
<td>es</td>
<td>Spanish</td>
</tr>
<tr>
<td>fr-CA</td>
<td>French-Canadian</td>
<td>ru</td>
<td>Russian</td>
</tr>
<tr>
<td>de</td>
<td>German</td>
<td>sv</td>
<td>Swedish</td>
</tr>
<tr>
<td>de-CH</td>
<td>German-Swiss</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example**

List all available language codes.

```javascript
console.println( spell.languages.toSource() );
```

#### languageOrder

This property returns the dictionary search order as an array of ISO 639-2, 3166 language codes. This is the order specified by the user on the Spelling Preferences panel. The Spelling plug-in will search for words first in the `doc.spellLanguageOrder` array if it has been set for the document, and then it will search this array of languages.

**Type:** Array  
**Access:** R.

**Example**

Get a listing of the dictionary search order.

```javascript
console.println( spell.languageOrder.toSource() );
```
Spell Methods

addDictionary

Adds a dictionary to the list of available dictionaries.

A dictionary actually consists of four files: `DDDxxxxx.hyp`, `DDDxxxxx.lex`, `DDDxxxxx.clx`, and `DDDxxxxx.env`. The `cFile` parameter must be the device-independent path of the `.hyp` file. For example, `/c/temp/testdict/TST.hyp`.

Spelling will look in the parent directory of the TST.hyp file for the other three files. All four file names must start with the same unique 3 characters to associate them with each other, and they must end with the dot three extensions listed above, even on a Macintosh.

**Note:** Beginning with Acrobat 6.0, this method is no longer supported. The return value of this method is always `false`. Use the `customDictionaryOpen` method.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cFile</strong></td>
<td>The device-independent path to the dictionary files.</td>
</tr>
<tr>
<td><strong>cName</strong></td>
<td>The dictionary name used in the spelling dialog and can be used as the input parameter to the <code>check</code>, <code>checkText</code>, and <code>checkWord</code>.</td>
</tr>
<tr>
<td><strong>bShow</strong></td>
<td>(optional) When <code>true</code> (the default), Spelling combines the <code>cName</code> value with &quot;User: &quot; and shows that name in all lists and menus. For example if <code>cName</code> is &quot;Test&quot;, Spelling adds &quot;User: Test&quot; to all lists and menus. When <code>false</code>, Spelling does not show this custom dictionary in any lists or menus.</td>
</tr>
</tbody>
</table>

**Returns**

`false`

addWord

Adds a new word to a dictionary. See also the `removeWord`.

**Note:** Internally, the Spell Check Object scans the user "Not-A-Word" dictionary and removes the word if it is listed there. Otherwise, the word is added to the user dictionary. The actual dictionary is not modified.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cWord</td>
<td>The new word to add.</td>
</tr>
<tr>
<td>cName</td>
<td>(optional) The dictionary name or language code. An array of the currently installed dictionaries can be obtained using <code>dictionaryNames</code> or <code>languages</code>.</td>
</tr>
</tbody>
</table>

Returns

`true` if successful, otherwise, `false`.

check

Presents the Spelling dialog to allow the user to correct misspelled words in form fields, annotations, or other objects.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aDomain</td>
<td>(optional) An array of document objects that should be checked by the Spelling plug-in, for example form fields or comments. When you do not supply an array of domains the &quot;EveryThing&quot; domain will be used. An array of the domains that have been registered can be obtained using <code>domainNames</code>.</td>
</tr>
<tr>
<td>aDictionary</td>
<td>(optional) The array of dictionary names or language codes that the spell checker should use. The order of the dictionaries in the array is the order the spell checker will use to check for misspelled words. An array of the currently installed dictionaries can be obtained using <code>spell.dictionaryNames</code> or <code>spell.languages</code>. When this parameter is omitted the <code>spellDictionaryOrder</code> list will be searched followed by the <code>dictionaryOrder</code> list.</td>
</tr>
</tbody>
</table>

Returns

`true` if the user changed or ignored all of the flagged words. When the user dismisses the dialog before checking everything the method returns `false`.

Example

```javascript
var dictionaries = ["de", "French", "en-GB"];
var domains = ["All Form Fields", "All Annotations"];
if (spell.check(domains, dictionaries) )
    console.println("You get an A for spelling.");
else
    console.println("Please spell check this form before you submit.");
```
checkText

checkText

5.0

Presents the spelling dialog to allow the user to correct misspelled words in the specified string.

Parameters

<table>
<thead>
<tr>
<th>cText</th>
<th>The string to check.</th>
</tr>
</thead>
<tbody>
<tr>
<td>aDictionary</td>
<td>(optional) The array of dictionary names or language codes that the spell checker should use. The order of the dictionaries in the array is the order the spell checker will use to check for misspelled words. An array of the currently installed dictionaries can be obtained using spell.dictionaryNames or spell.languages. When this parameter is omitted the spellDictionaryOrder list will be searched followed by the dictionaryOrder list.</td>
</tr>
</tbody>
</table>

Returns

The result from the spelling dialog in a new string.

Example

```javascript
var f = this.getField("Text Box") // a form text box
f.value = spell.checkText(f.value); // let the user pick the dictionary
```

checkWord

checkWord

5.0

Checks the spelling of a specified word.

Parameters

<table>
<thead>
<tr>
<th>cWord</th>
<th>The word to check.</th>
</tr>
</thead>
<tbody>
<tr>
<td>aDictionary</td>
<td>(optional) The array of dictionary names or language codes that the spell checker should use. The order of the dictionaries in the array is the order the spell checker will use to check for misspelled words. An array of the currently installed dictionaries can be obtained using spell.dictionaryNames or spell.languages. When this parameter is omitted the spellDictionaryOrder list will be searched followed by the dictionaryOrder list.</td>
</tr>
</tbody>
</table>
Returns

A null object if the word is correct, otherwise an array of alternative spellings for the unknown word.

Example 1

```javascript
var word = "subpinna"; /* misspelling of "subpoena" */
var dictionaries = ["English"];
var f = this.getField("Alternatives") // alternative spellings listbox
f.clearItems();
f.setItems(spell.checkWord(word, dictionaries));
```

Example 2

The following script goes through the document and marks with a squiggle annot any misspelled word. The contents of the squiggle annot contains the suggested alternative spellings. The script can be executed from the console, as a mouse up action within the document, a menu, or as a batch sequence.

```javascript
var ckWord, numWords;
for (var i = 0; i < this.numPages; i++)
{
    numWords = this.getPageNumWords(i);
    for (var j = 0; j < numWords; j++)
    {
        ckWord = spell.checkWord(this.getPageNthWord(i, j));
        if ( ckWord != null )
        {
            this.addAnnot({
                page: i,
                type: "Squiggly",
                quads: this.getPageNthWordQuads(i, j),
                author: "A. C. Acrobat",
                contents: ckWord.toString()
            });
        }
    }
}
```

customDictionaryClose

Closes a custom dictionary that was opened using customDictionaryOpen or customDictionaryCreate.
Parameters

**cName**

Dictionary name used when this dictionary was opened or created.

**Returns**

**true** if successful, **false** on failure.

**customDictionaryCreate**

Use this method to create a new custom dictionary file and add it to the list of available dictionaries.

**NOTE:** (Security ☑): This method is allowed only during console, menu or batch events.

**Parameters**

<table>
<thead>
<tr>
<th><strong>cName</strong></th>
<th>Dictionary name used in the spelling dialog and can be used as the input parameter to <code>check</code>, <code>checkText</code>, and <code>checkWord</code> methods.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cLanguage</td>
<td>(optional) Use this parameter to associate this dictionary with a language. A list of available languages can be obtained from the <code>spell.languages</code> property.</td>
</tr>
<tr>
<td>bShow</td>
<td>(optional) If <strong>true</strong>, the default, spelling will combine the cName parameter with &quot;User: &quot; and show that name in all lists and menus. For Example, if cName is &quot;Test&quot;, spelling will add &quot;User: Test&quot; to all lists and menus. When bShow is <strong>false</strong>, Spelling will not show this custom dictionary in any lists or menus.</td>
</tr>
</tbody>
</table>

**Returns**

**true** if successful, **false** on failure. This method will fail if the user does not have read and write permission to this directory.

**Example**

Open this document, the *Acrobat JavaScript Scripting Reference*, in Acrobat and execute the following script in the console. This script goes through the bookmarks and extracts the first word of each bookmark. If that word is already in a dictionary, it is discarded. An unknown word—assumed to be the name of an Acrobat JavaScript object, property or method—is added into a newly created dictionary called "JavaScript".

```javascript
spell.customDictionaryCreate("JavaScript", "en", true);
function GetJSTerms(bm, nLevel)
{
    var newWord = bm.name.match(re);
    var ckWord = spell.checkWord( newWord[0] );
```
if ( ckWord != null )
{
    var cWord = spell.addWord( newWord[0], "JavaScript");
    if ( cWord ) console.println( newWord[0] );
}
if (bm.children != null)
for (var i = 0; i < bm.children.length; i++)
    GetJSTerms(bm.children[i], nLevel + 1);
}
console.println("\nAdding New words to the \"JavaScript\" "
+ "dictionary:");
var re = /^\w+$/;
GetJSTerms(this.bookmarkRoot, 0);

**customDictionaryDelete**

Use this method to close and delete a custom dictionary file that was opened via `customDictionaryOpen` or `customDictionaryCreate`.

**NOTE:** (Security ☐): This method is allowed only during console, menu or batch events.

**Parameters**

<table>
<thead>
<tr>
<th>cName</th>
</tr>
</thead>
<tbody>
<tr>
<td>The name of the dictionary to be deleted. This is the name used when this dictionary was opened or created.</td>
</tr>
</tbody>
</table>

**Returns**

`true` if successful, `false` on failure. This method will fail if the user does not have sufficient file system permission.

**Example**

Delete a custom dictionary.

```
spell.customDictionaryDelete("JavaScript");
```

**customDictionaryExport**

This method will export a custom dictionary to a new file that was opened using the spell methods `customDictionaryOpen` or `customDictionaryCreate`.

The user will be prompted for an export directory. The custom dictionary will be saved there as a `.clam` file using the dictionary name and the language specified on `customDictionaryCreate`. For example if the dictionary name is "JavaScript" and the
"en" language as specified when it was created then the export file name will be JavaScript-eng.clam.

Exported custom dictionaries can be used in subsequent `customDictionaryOpen` calls.

**NOTE:** (Security ☭): This method is allowed only during console, menu or batch events.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>The dictionary name used when this dictionary was opened or created.</td>
</tr>
</tbody>
</table>

### Returns

- **true** if successful, **false** on failure. This method will fail if the user does not have sufficient file system permission.

### Example

Export a custom dictionary for distribution to other users. The exported dictionary can then be sent to other users. (See the example that follows `customDictionaryCreate`.)

```javascript
spell.customDictionaryExport("JavaScript");
```

---

### `customDictionaryOpen`

<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0</td>
</tr>
</tbody>
</table>

Use this method to add an custom export dictionary to the list of available dictionaries. See `customDictionaryExport`.

**NOTE:** A custom dictionary file can be created using the `customDictionaryCreate` and `customDictionaryExport` methods.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cDIPath</td>
<td>The device independent path to the custom dictionary file.</td>
</tr>
<tr>
<td>cName</td>
<td>Dictionary name used in the spelling dialog and can be used as the input parameter to <code>check</code>, <code>checkText</code>, and <code>checkWord</code> methods.</td>
</tr>
<tr>
<td>bShow</td>
<td>(optional) If <strong>true</strong>, the default, Spelling will combine the <strong>cName</strong> parameter with &quot;User: &quot; and show that name in all lists and menus. For Example if <strong>cName</strong> is &quot;Test&quot;, Spelling will add &quot;User: Test&quot; to all lists and menus. When <strong>bShow</strong> is <strong>false</strong>, Spelling will not show this custom dictionary in any lists or menus.</td>
</tr>
</tbody>
</table>

---
Returns

true if successful, false on failure. This method will fail if the user does not have read permission for the file.

Example

This example continues the ones begun following customDictionaryCreate and customDictionaryExport.

Add an custom export dictionary to the list of available dictionaries. The user places the custom export dictionary any any folder for which there is read/write permission. One particular choice is the user dictionaries folder. This location of this folder can be obtained from the app.getPath method.

    app.getPath("user", "dictionaries");

Once the export dictionary has been placed, listing it can be made automatic by adding some folder level JavaScript. The path to the user JavaScripts can be obtained by executing

    app.getPath("user", "javascript");

Finally, create an .js file in this folder and add the line

    var myDictionaries = app.getPath("user", "dictionaries");
    spell.customDictionaryOpen( myDictionaries, "JavaScripts", true);

The next time Acrobat is started, the "JavaScript" dictionary will be open and available.

ignoreAll

6.0

Adds or removes a word from the Spelling ignored-words list of the current document.

NOTE: A document must be open in the viewer or this method will throw an exception.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cWord</td>
<td>The word to be added or removed from the ignored list.</td>
</tr>
<tr>
<td>bIgnore</td>
<td>(optional) If true (the default), the word is added to the document ignored word list; if false, the word is removed from the ignored list.</td>
</tr>
</tbody>
</table>

Returns

true if successful. An exception is thrown if there is no document open in the viewer when this method is executed.

Example

    var bIgnored = spell.ignoreAll("foo");
    if (bIgnored) console.println("\"foo\" will be ignored");
removeDictionary

Removes a user dictionary that was added via `addDictionary`.

**NOTE:** Beginning with Acrobat 6.0, this method is no longer supported. The return value of this method is always `false`. Use the `customDictionaryClose` method.

**Parameters**

| cName | The name of the dictionary to remove. Must be the same name as was used with `addDictionary`. |

**Returns**

`false`

removeWord

Removes a word from a dictionary. Words cannot be removed from user dictionaries that were created using either `customDictionaryCreate` or `customDictionaryExport`.

See also `addWord`.

**NOTE:** Internally the Spell Check object scans the user dictionary and removes the previously added word if it is there. Otherwise the word is added to the user's "Not-A-Word" dictionary. The actual dictionary is not modified.

**Parameters**

| cWord | The word to remove. |
| cName | (optional) The dictionary name or language code. An array of currently installed dictionaries can be obtained using `dictionaryNames` or `languages`. |

**Returns**

`true` if successful, `false` otherwise
userWords

5.0  X

Gets the array of words a user has added to or removed from a dictionary. See also addWord and checkWord.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cName</td>
<td>(optional) The dictionary name or language code. An array of currently installed dictionaries can be obtained using dictionaryNames or languages. If cName is not specified, the current default dictionary will be used. The current default dictionary is the first dictionary specified in the Spelling preferences dialog.</td>
</tr>
<tr>
<td>bAdded</td>
<td>(optional) When true, return the user's array of added words. When false, return the user's array of removed words. The default is true.</td>
</tr>
</tbody>
</table>

Returns

The user's array of added or removed words.

Example

List the words added to the "JavaScript" dictionary. (See the example that follows the description of customDictionaryCreate.)

```javascript
var aUserWords = spell.userWords({cName: "JavaScript"});
aUserWords.toSource();
```

Statement Object

5.0  X

Use statement objects to execute SQL updates and queries, and retrieve the results of these operations. To create a statement object, use connection.newStatement.

See also:

- The Connection Object.
- The ADBC Object.
- Column Generic Object, ColumnInfo Generic Object, Row Generic Object, TableInfo Generic Object
Statement Properties

**columnCount**

The number of columns in each row of results returned by a query. It is undefined in the case of an update operation.

_Type: Number  Access: R._

**rowCount**

The number of rows affected by an update. It is _not_ the number of rows returned by a query. Its value is undefined in the context of a query.

_Type: Number  Access: R._

Statement Methods

**execute**

Executes an SQL statement through the context of the Statement object. On failure, _execute_ throws an exception.

*NOTE:* There is no guarantee that a client can do anything on a statement if an execute has neither failed nor returned all of its data.

**Parameters**

<table>
<thead>
<tr>
<th>cSQL</th>
<th>The SQL statement to execute.</th>
</tr>
</thead>
</table>

**Returns**

Nothing

**Example**

```javascript
statement.execute("Select * from ClientData");
```

If the name of the database table or column name contains spaces, they need to be enclosed in escaped quotes. For example:

```javascript
var execStr1 = "Select firstname, lastname, ssn from "Employee Info";";
var execStr2 = "Select "First Name" from "Client Data";"
statement.execute(execStr1);
statement.execute(execStr2);
```
A cleaner solution would be to enclose the whole SQL string with single quotes, then table names and column names can be enclosed with double quotes.

```javascript
var execStr3 = 'Select "First Name","Second Name" from "Client Data" ;
statement.execute(execStr3);
```

See `getRow` and `nextRow` for extensive examples.

### getColumn

Obtains a `column` object representing the data in the specified column.

**NOTE:** Once a column is retrieved with one of these methods, future calls attempting to retrieve the same column may fail.

#### Parameters

<table>
<thead>
<tr>
<th>nColumn</th>
<th>The column from which to get the data. May be a column number or a string, the name of the desired column (see the ColumnInfo Generic Object).</th>
</tr>
</thead>
<tbody>
<tr>
<td>nDesiredType</td>
<td>(optional) Which of the ADBC JavaScript Types best represents the data in the column.</td>
</tr>
</tbody>
</table>

#### Returns

A `Column Generic Object` representing the data in the specified column, or `null` on failure.

### getColumnArray

Obtains an array of `column` objects, one for each column in the result set. A “best guess” is used to decide which of the ADBC JavaScript Types best represents the data in the column.

**NOTE:** Once a column is retrieved with one of these methods, future calls attempting to retrieve the same column may fail.

#### Parameters

None

#### Returns

An array of `column` objects, or `null` on failure as well as a zero-length array.

### getRow

Obtains a `Row Generic Object` representing the current row. This object contains information from each column. As for `getColumnArray`, column data is captured in the “best guess” format.
A call to `nextRow` should precede a call to `getRow`. Calling `getRow` twice, without an intervening call to `nextRow` yields a `null` return value for the second call to `getRow`.

**Parameters**

None

**Returns**

A Row Generic Object.

**Example 1**

Every Row object contains a property for each column in a row of data. Consider the following example:

```javascript
var execStr = "SELECT firstname, lastname, ssn FROM "Employee Info"";
statement.execute(execStr);
statement.nextRow();
row = statement.getRow();
console.println("The first name of the first person retrieved is: " + row.firstname.value);
console.println("The last name of the first person retrieved is: " + row.lastname.value);
console.println("The ssn of the first person retrieved is: " + row.ssn.value);
```

**Example 2**

If the column name contains spaces, then the above syntax for accessing the row properties (for example, `row.firstname.value`) does not work. Alternatively,

```javascript
Connect = ADBC.newConnection("Test Database");
statement = Connect.newStatement();
var execStr = 'Select  "First Name","Second Name"  from "Client Data" ';
statement.execute(execStr);
statement.nextRow();

// Populate this PDF file
this.getField("name.first").value = row["First Name"].value;
this.getField("name.last").value = row["Second Name"].value;
```

**nextRow**

Obtains data about the next row of data generated by a previously executed query. This must be called following a call to `execute` to acquire the first row of results.

**Parameters**

None

**Returns**

Nothing. Throws an exception on `failure` (if, for example, there is no next row).
Example

The following example is a rough outline of how to create a series of buttons and Document Level JavaScripts to browse a database and populate a PDF form.

For the `getNextRow` button, defined below, the `nextRow()` is used to retrieve the next row from the database, unless there is an exception thrown (indicating that there is no next row), in which case, we reconnect to the database, and use `nextRow()` to retrieve the first row of data (again).

```javascript
/* 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;
}
```javascript
this.getField("address").value = row.Address.value;
this.getField("city").value = row.City.value;
this.getField("state").value = row.State.value;
this.getField("zip").value = row.Zipcode.value;
this.getField("telephone").value = row.Telephone.value;
this.getField("income").value = row.Income.value;
```

### TableInfo Generic Object

This generic JS object contains basic information about a table, and is returned by `connection.getTableList`. It contains the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>name</code></td>
<td>String</td>
<td>R</td>
<td>The identifying name of a table. This string could be used in SQL statements to identify the table that the <code>TableInfo</code> object is associated with.</td>
</tr>
<tr>
<td><code>description</code></td>
<td>String</td>
<td>R</td>
<td>A string that contains database-dependent information about the table.</td>
</tr>
</tbody>
</table>

### Template Object

Template objects are named pages within the document. These pages may be hidden or visible and can be copied or spawned. They are typically used to dynamically create content (for example, to add pages to an invoice on overflow).

See also the `Doc Object templates` property, and methods `createTemplate`, `getTemplate`, and `removeTemplate`.

### Template Properties

**hidden**

Whether the template is hidden or not. Hidden templates cannot be seen by the user until they are spawned or are made visible. When an invisible template is made visible it is appended to the document.
NOTE: Setting this property in Adobe Reader (before 5.1) generates an exception. For Adobe 5.1 Reader and beyond, setting this property depends on Advanced Forms Feature document rights.

Type: Boolean Access: R/W.

name

The name of the template which was supplied when the template was created.

Type: String Access: R.

Template Methods

spawn

Creates a new page in the document based on the template.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nPage</td>
<td>(optional) The 0-based index of the page number after which or on which the new page will be created, depending on the value of bOverlay. The default is 0.</td>
</tr>
<tr>
<td>bRename</td>
<td>(optional) Whether form fields on the page should be renamed. The default is true.</td>
</tr>
<tr>
<td>bOverlay</td>
<td>(optional) When true (the default), the template is overlaid on the specified page. When false, it is inserted as a new page before the specified page. To append a page to the document, set bOverlay to false and set nPage to the number of pages in the document.</td>
</tr>
<tr>
<td>oXObject</td>
<td>(optional, version 6.0) The value of this parameter is the return value of an earlier call to spawn.</td>
</tr>
</tbody>
</table>

NOTE: For certified documents, or documents with “Advanced Form Features rights” (F), the bOverlay parameter is disabled; this means that a template cannot be overlaid for these types of documents.
Returns

Prior to Acrobat 6.0, this method returned nothing. Now, `spawn` returns an object representing the page contents of the page spawned. This return object can then be used as the value of the optional parameter `oXObject` for subsequent calls to `spawn`.

**NOTE:** Repeatedly spawning the same page can cause a large inflation in the file size. To avoid this file size inflation problem, `spawn` now returns an object that represents the page contents of the spawned page. This return value can be used as the value of the `oXObject` parameter in subsequent calls to the `spawn` method to spawn the same page.

Example 1

This example spawns all templates and appends them one by one to the end of the document.

```javascript
var a = this.templates;
for (i = 0; i < a.length; i++)
    a[i].spawn(this.numPages, false, false);
```

Example 2 (version 6.0)

The following example spawns the same template 31 times using the `oXObject` parameter and return value. Using this technique avoids overly inflating the file size.

```javascript
var t = this.templates;
var T = t[0];
var XO = T.spawn(this.numPages, false, false);
for (var i=0; i<30; i++) T.spawn(this.numPages, false, false, XO);
```

**Thermometer Object**

This object is a combined status window/progress bar that indicates to the user that a lengthy operation is in progress. To acquire a `thermometer` object, use `app.thermometer`.

**Example**

The following is a general example that illustrates how to use all properties and methods of the `thermometer` object.

```javascript
var t = app.thermometer; // acquire a thermometer object
t.duration = this.numPages;
t.begin();
for ( var i = 0; i < this.numPages; i++)
{
    t.value = i;
    t.text = "Processing page " + (i + 1);
    if (t.cancelled) break; // break if operation cancelled
```
... process the page ...
}
t.end();

---

### Thermometer Properties

#### cancelled

Whether the user wants to cancel the current operation. The user can indicate to the script the desire to terminate the operation by pressing the escape key on the Windows and Unix platforms and command-period on the Macintosh platform.

*Type: Boolean  Access: R.*

#### duration

Sets the value that corresponds to a full thermometer display. The thermometer is subsequently filled in by setting its `value`. The default duration is 100.

*Type: Number  Access: R/W.*

#### value

Sets the current value of the thermometer and updates the display. The allowed value ranges from 0 (empty) to the value set in the `duration`. For example, if the thermometer’s duration is 10, the current value must be between 0 and 10, inclusive. If value is less than zero, it is set to zero. If value is greater than duration, it is set to duration.

*Type: Number  Access: R/W.*

#### text

Sets the text string that is displayed by the thermometer.

*Type: String  Access: R/W.*

---

### Thermometer Methods

#### begin

Initializes the thermometer and displays it with the current value as a percentage of the duration.
Parameters
None

Returns
Nothing

Example
Count words on each page of current document, report running total and use thermometer to track progress.

```javascript
var t = app.thermometer; // acquire a thermometer object
  t.duration = this.numPages;
  t.begin();
  var cnt=0;
  for ( var i = 0; i < this.numPages; i++)
  {
    t.value = i;
    t.text = "Processing page " + (i + 1);
    cnt += getPageNumWords(i);
    console.println("There are " + cnt + "words in this doc.");
    if (t.cancelled) break;
  }
  t.end();
end
```

Draws the thermometer with its current value set to the thermometer's duration (a full thermometer), then removes the thermometer from the display.

Parameters
None

Returns
Nothing

TTS Object

The JavaScript TTS object allows users to transform text into speech. To be able to use the TTS object, the user's machine must have a Text-To-Speech engine installed on it. The Text-To-Speech engine will render text as digital audio and then “speak it”. It has been implemented mostly with accessibility in mind but it could potentially have many other applications, bringing to life PDF documents.
This is currently a Windows-only feature and requires that the MicroSoft Text to Speech engine be installed in the operating system.

The **TTS** object is present on both the Windows and Mac platforms (since it is a JavaScript object); however, it is disabled on the Mac.

**NOTE:** Acrobat 5.0 has taken a very different approach to providing accessibility for disabled users by integrating directly with popular screen readers. Some of the features and methods defined in 4.05 for the TTS object have been deprecated as a result as they conflict with the screen reader. The TTS object remains, however, as it still has useful functionality in its own right that might be popular for multi-media documents.

---

### TTS Properties

#### available

*true* if the TTS object is available and the Text-To-Speech engine can be used.

*Type: Boolean*  
*Access: R.*

**Example**

```javascript
console.println("Text to speech available: " + tts.available);
```

#### numSpeakers

The number of different speakers available to the current text to speech engine. See also the **speaker** and the **getNthSpeakerName**.

*Type: Integer*  
*Access: R.*

#### pitch

Sets the baseline pitch for the voice of a speaker. The valid range for pitch is from 0 to 10, with 5 being the default for the mode.

*Type: Integer*  
*Access: R/W.*

#### soundCues

Deprecated. Now returns only **false**.

*Type: Boolean*  
*Access: R/W.*
**speaker**

Allows users to specify different speakers with different tone qualities when performing text-to-speech. See also the `numSpeakers` and the `getNthSpeakerName`.

*Type: String  Access: R/W.*

**speechCues**

Deprecated. Now returns only `false`.

*Type: Boolean  Access: R/W.*

**speechRate**

Sets the speed at which text will be spoken by the Text-To-Speech engine. The value for `speechRate` is expressed in number of words per minute.

*Type: Integer  Access: R/W.*

**volume**

Sets the volume for the speech. Valid values are from 0 (mute) to 10 (loudest).

*Type: Integer  Access: R/W.*

---

**TTS Methods**

**getNthSpeakerName**

Gets the nth speaker name in the currently installed text to speech engine (see also `numSpeakers` and `speaker`).

**Parameters**

- `nIndex`  
  The index of the desired speaker name.

**Returns**

The name of the specified speaker.
Example

Enumerate through all of the speakers available.

```javascript
for (var i = 0; i < tts.numSpeakers; i++) {
    var cSpeaker = tts.getNthSpeakerName(i);
    console.println("Speaker[" + i + "] = " + cSpeaker);
    tts.speaker = cSpeaker;
    tts.qText ("Hello");
    tts.talk();
}
```

pause

Immediately pauses text-to-speech output on a TTS object. Playback of the remaining queued text can be resumed via `resume`.

Parameters

None

Returns

Nothing

qSilence

Queues a period of silence into the text.

Parameters

- `nDuration` (The amount of silence in milliseconds)

Returns

Nothing

qSound

Puts the specified sound into the queue in order to be performed by `talk`. It accepts one parameter, `cSound`, from a list of possible sound cue names. These names map directly to sound files stored in the SoundCues folder, if it exists.

```javascript
tts.qSound("DocPrint"); // Plays DocPrint.wav
```

The SoundCues folder should exist at the program level for the viewer, for example, `C:\Program Files\Adobe\Acrobat 5.0\SoundCues`.

**Note:** Windows only—`qSound` can handle only 22KHz,16 bit PCM .wav files. These should be at least one second long in order to avoid a queue delay problem in MS SAPI. In case the sound lasts less than one second, it should be edited and have a silence added to the end of it.
TTS Methods

**qText**

Puts text into the queue in order to be performed by `talk`.

**Parameters**

```
cText The text to convert to speech.
```

**Returns**

Nothing

**Example**

```javascript
tts.qText("Hello, how are you?");
```

**reset**

Stops playback of current queued text and flushes the queue. Playback of text cannot be resumed via `resume`. Additionally, it resets all the properties of the TTS object to their default values.

**Parameters**

None

**Returns**

Nothing

**resume**

Resumes playback of text on a paused TTS object.

**Parameters**

None

**Returns**

Nothing
**stop**

Stops playback of current queued text and flushes the queue. Playback of text cannot be resumed with `resume`.

**Parameters**

None

**Returns**

Nothing

**talk**

Sends whatever is in the queue to be spoken by the Text-To-Speech engine. If text output had been paused, `talk` resumes playback of the queued text.

**Parameters**

None

**Returns**

Nothing

**Example**

```javascript
  tts.qText("Hello there!");
  tts.talk();
```

---

**this Object**

In JavaScript the special keyword `this` refers to the current object. In Acrobat the current object is defined as follows:

- In an object method, it is the object to which the method belongs.
- In a constructor function, it is the object being constructed.
- In a function defined in one of the Folder Level JavaScripts files, it is undefined. It is recommended that calling functions pass the document object to any function at this level that needs it.
- In a Document level script or Field level script it is the document object and therefore can be used to set or get document properties and functions.

For example, assume that the following function was defined at the Plug-in folder level:

```javascript
function PrintPageNum(doc)
{
  /* Print the current page number to the console. */
  console.println("Page = " + doc.page);
}
```
The following script outputs the current page number to the console (twice) and then prints the page:

```javascript
/* Must pass the document object. */
PrintPageNum(this);
/* Same as the previous call. */
console.println("Page = " + this.pageNum);
/* Prints the current page. */
this.print(false, this.pageNum, this.pageNum);
```

### Variable and Function Name Conflicts

Variables and functions that are defined in scripts are parented off of the `this` object. For example:

```javascript
var f = this.getField("Hello");
```

is equivalent to

```javascript
this.f = this.getField("Hello");
```

with the exception that the variable `f` can be garbage collected at any time after the script is run.

Acrobat JavaScript programmers should avoid using property and method names from the `Doc Object` as variable names. Use of method names when after the reserved word "var" will throw an exception, as the following line illustrates:

```javascript
var getField = 1; // TypeError: redeclaration of function getField
```

Use of property names will not throw an exception, but the value of the property may not be altered if the property refers to an object:

```javascript
// "title" will return "1", but the document will now be named "1".
var title = 1;

// property not altered, info still an object
var info = 1; // "info" will return [object Info]
```

The following is an example of avoiding variable name clash.

```javascript
var f = this.getField("mySignature"); // uses the ppklite sig handler

// use "Info" rather than "info" to avoid a clash
var Info = f.signatureInfo();

// some standard signatureInfo properties
console.println("name = " + Info.name);
```

### Util Object

A static JavaScript object that defines a number of utility methods and convenience functions for string and date formatting and parsing.
Util Methods

printf

Formats one or more values as a string according to a format string. This is similar to the C function of the same name. This method converts and formats incoming arguments into a result string according to a format string (cFormat).

The format string consists of two types of objects:
- Ordinary characters, which are copied to the result string
- Conversion specifications, each of which causes conversion and formatting of the next successive argument to printf().

Each conversion specification is constructed as follows:

`%[,nDecSep][cFlags][nWidth][.nPrecision]cConvChar`

The following table describes the components of a conversion specification.

<table>
<thead>
<tr>
<th>nDecSep</th>
<th>Preceded by a comma character (,), is a digit from 0 to 3 which indicates the decimal/separater format:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>comma separated, period decimal point.</td>
</tr>
<tr>
<td>1</td>
<td>no separator, period decimal point.</td>
</tr>
<tr>
<td>2</td>
<td>period separated, comma decimal point.</td>
</tr>
<tr>
<td>3</td>
<td>no separator, comma decimal point.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cFlags</th>
<th>Only valid for numeric conversions and consists of a number of characters (in any order), which will modify the specification:</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>specifies that the number will always be formatted with a sign.</td>
</tr>
<tr>
<td>space</td>
<td>if the first character is not a sign, a space will be prefixed.</td>
</tr>
<tr>
<td>0</td>
<td>specifies padding to the field with leading zeros.</td>
</tr>
<tr>
<td>#</td>
<td>- which specifies an alternate output form. For f the output will always have a decimal point.</td>
</tr>
</tbody>
</table>

| nWidth        | A number specifying a minimum field width. The converted argument will be formatted in so that it is at least this many characters wide, including the sign and decimal point, and may be wider if necessary. If the converted argument has fewer characters than the field width it will be padded on the left to make up the field width. The padding character is normally a space, but is 0 if zero padding flag is present. |

| nPrecision    | A number, preceded by a period character (.), which specifies the number of digits after the decimal point for float conversions. |
printd

Formats a date. Valid string format values for the cFormat parameter are as follows:

<table>
<thead>
<tr>
<th>String</th>
<th>Effect</th>
<th>Example</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>mmmm</td>
<td>Long month</td>
<td>September</td>
<td></td>
</tr>
<tr>
<td>mmm</td>
<td>Abbreviated month</td>
<td>Sept</td>
<td></td>
</tr>
<tr>
<td>mm</td>
<td>Numeric month with leading zero</td>
<td>09</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>Numeric month without leading zero</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>ddd</td>
<td>Long day</td>
<td>Wednesday</td>
<td></td>
</tr>
<tr>
<td>ddd</td>
<td>Abbreviated day</td>
<td>Wed</td>
<td></td>
</tr>
</tbody>
</table>
A variety of addition “quick” formats are possible using numeric values.

<table>
<thead>
<tr>
<th>String</th>
<th>Effect</th>
<th>Example</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>dd</td>
<td>Numeric date with leading zero</td>
<td>03</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>Numeric date without leading zero</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>yyyy</td>
<td>Long year</td>
<td>1997</td>
<td></td>
</tr>
<tr>
<td>yy</td>
<td>Abbreviate Year</td>
<td>97</td>
<td></td>
</tr>
<tr>
<td>HH</td>
<td>24 hour time with leading zero</td>
<td>09</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>24 hour time without leading zero</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>hh</td>
<td>12 hour time with leading zero</td>
<td>09</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>12 hour time without leading zero</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>minutes with leading zero</td>
<td>08</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>minutes without leading zero</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>ss</td>
<td>seconds with leading zero</td>
<td>05</td>
<td></td>
</tr>
<tr>
<td>s</td>
<td>seconds without leading zero</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>tt</td>
<td>am/pm indication</td>
<td>am</td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>single digit am/pm indication</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>j</td>
<td>Japanese Emperor Year (abbreviated)</td>
<td></td>
<td>6.0</td>
</tr>
<tr>
<td>jj</td>
<td>Japanese Emperor Year</td>
<td></td>
<td>6.0</td>
</tr>
<tr>
<td>\</td>
<td>use as an escape character</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Example</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>PDF date format</td>
<td>D:20000801145605+07'00'</td>
<td>5.0</td>
</tr>
<tr>
<td>1</td>
<td>Universal</td>
<td>2000.08.01 14:56:05 +07'00'</td>
<td>5.0</td>
</tr>
<tr>
<td>2</td>
<td>Localized string</td>
<td>2000/08/01 14:56:05</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cFormat</td>
<td>The format to use; a string or a number.</td>
</tr>
<tr>
<td>oDate</td>
<td>The date to format.</td>
</tr>
</tbody>
</table>

Returns

The formatted date string.

Example

To format the current date in long format, for example, you would use the following script:

```javascript
var d = new Date();
console.println(util.printd("MMMM dd, yyyy", d));
```

Example (Version 5.0)

// display date in a local format
```javascript
var d = new Date();
console.println(util.printd(2, d));
```

Example (Version 6.0)

```javascript
var d = new Date();
console.println(util.printd("jj", d));
```

printx

Forms a source string, `cSource`, according to a formatting string, `cFormat`. A valid format for `cFormat` is any string which may contain special masking characters:

<table>
<thead>
<tr>
<th>Value</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Copy next character.</td>
</tr>
<tr>
<td>X</td>
<td>Copy next alphanumeric character, skipping any others.</td>
</tr>
<tr>
<td>A</td>
<td>Copy next alpha character, skipping any others.</td>
</tr>
<tr>
<td>9</td>
<td>Copy next numeric character, skipping any others.</td>
</tr>
<tr>
<td>*</td>
<td>Copy the rest of the source string from this point on.</td>
</tr>
<tr>
<td>\</td>
<td>Escape character.</td>
</tr>
<tr>
<td>&gt;</td>
<td>Uppercase translation until further notice.</td>
</tr>
<tr>
<td>&lt;</td>
<td>Lowercase translation until further notice.</td>
</tr>
<tr>
<td>=</td>
<td>Preserve case until further notice (default).</td>
</tr>
</tbody>
</table>
Parameters

- **cFormat**: The formatting string to use.
- **cSource**: The source string to use.

Returns

The formatted string.

Example

To format a string as a U.S. telephone number, for example, use the following script:

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

**scand**

Converts the supplied date, **cDate**, into a JavaScript date object according to rules of the supplied format string, **cFormat**. This routine is much more flexible than using the date constructor directly.

**Note**: Given a two-digit year for input, **scand** resolves the ambiguity as follows: if the year is less than 50 then it is assumed to be in the 21st century (that is, add 2000), if it is greater than or equal to 50 then it is in the 20th century (add 1900). This heuristic is often known as the **Date Horizon**.

Parameters

- **cFormat**: The rules to use for formatting the date. **cFormat** uses the same syntax as found in **printd**.
- **cDate**: The date to convert.

Returns

The converted **date** object.

Example

```javascript
/* 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 mm/dd", d));
```
spansToXML

This method converts an array of Span Objects into an XML(XFA) String as described in the PDF 1.5 Specification.

**Parameters**

| An array of Span Objects | An array of span objects to be converted into an XML string. |

**Returns**

String

xmlToSpans

This method converts an XML(XFA) String as described in the PDF 1.5 Specification to an array of span objects suitable for specifying as the richValue or richContents of a field or annotation.

**Parameters**

| a string | An XML (XFA) string to be converted to an array of Span Object. |

**Returns**

The converted date object.

**Example**

This example gets the value of a rich text field, turns all of the text blue, converts it to an XML string and then prints it to the console

```javascript
var f = getField("Text1");
var spans = f.richValue;
for(var index = 0; index < spans.length; index++)
{
    spans[index].textColor = color.blue;
}
console.println(util.spansToXML(spans));
```
The XFA object corresponds to the appModel in the XFA Scripting reference. All the XFA documents are located at

New Features and Changes

This section summarizes the new features and changes introduced in Acrobat 6.0 and in Acrobat 5.0.

Acrobat 6.0 Changes

Safe Path

An important new security posture Acrobat has taken concerns all JavaScript methods that write data to the local hard drive based on a path passed to it by one of its parameters. All paths are required to a safe path: For windows, the path cannot point to a system critical folder, for example, a root, windows or system directory. However, this is not the only requirement for a path to be safe; a path is also subject to certain, unspecified tests as well.

For many of the methods in question, the file name must have an extension appropriate to the type of data that is to be saved; some methods may have a no-overwrite restriction. These additional restrictions are noted in the documentation.

Generally, when a path is judged to be “not safe”, a NotAllowedError (see the Error Objects) exception is thrown and the method fails.

Introduced in Acrobat 6.0

The following properties and methods are introduced in Acrobat 6:

<table>
<thead>
<tr>
<th>ADBC Object</th>
<th>SQL Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlternatePresentation Object</td>
<td>properties:</td>
</tr>
<tr>
<td></td>
<td>active type</td>
</tr>
<tr>
<td></td>
<td>methods:</td>
</tr>
<tr>
<td></td>
<td>start</td>
</tr>
<tr>
<td></td>
<td>stop</td>
</tr>
<tr>
<td>Annot Object</td>
<td>properties:</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>borderEffectIntensity</td>
<td></td>
</tr>
<tr>
<td>borderEffectStyle</td>
<td></td>
</tr>
<tr>
<td>inReplyTo</td>
<td></td>
</tr>
<tr>
<td>richContents</td>
<td></td>
</tr>
<tr>
<td>toggleNoView</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td>getStateInModel</td>
</tr>
<tr>
<td>transitionToState</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>App Object</th>
<th>properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromPDFConverters</td>
<td></td>
</tr>
<tr>
<td>printColorProfiles</td>
<td></td>
</tr>
<tr>
<td>printerNames</td>
<td></td>
</tr>
<tr>
<td>runtimeHighlight</td>
<td></td>
</tr>
<tr>
<td>runtimeHighlightColor</td>
<td></td>
</tr>
<tr>
<td>thermometer</td>
<td></td>
</tr>
<tr>
<td>viewerType</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td>addToolButton</td>
</tr>
<tr>
<td>getPath</td>
</tr>
<tr>
<td>mailGetAddr</td>
</tr>
<tr>
<td>newPDF</td>
</tr>
<tr>
<td>openPDF</td>
</tr>
<tr>
<td>popUpMenuEx</td>
</tr>
<tr>
<td>removeToolButton</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bookmark Object</th>
<th>methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.setAction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Catalog Object</th>
<th>properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td>isIdle</td>
<td></td>
</tr>
<tr>
<td>jobs</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td>getIndex</td>
</tr>
<tr>
<td>remove</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Certificate Object</th>
<th>properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyUsage</td>
<td></td>
</tr>
<tr>
<td>usage</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collab Object</th>
<th>methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>addStateModel</td>
</tr>
<tr>
<td></td>
<td>removeStateModel</td>
</tr>
</tbody>
</table>

---

396 Acrobat JavaScript Scripting Reference
## Acrobat 6.0 Changes

**Connection Object**
- **Methods:**
  - `close`

**Dbg Object**
- **Properties:**
  - `bps`
- **Methods:**
  - `c`
  - `cb`
  - `q`
  - `sb`
  - `si`
  - `sn`
  - `so`
  - `sv`

**Directory Object**
- **Properties:**
  - `info`
- **Methods:**
  - `connect`

**DirConnection Object**
- **Properties:**
  - `canList`
  - `canDoCustomSearch`
  - `canDoCustomUISearch`
  - `canDoStandardSearch`
  - `groups`
  - `name`
  - `uiName`
- **Methods:**
  - `search`
  - `setOutputFields`
## New Features and Changes

### Acrobat 6.0 Changes

**Doc Object**

- **properties:**
  - alternatePresentations
  - documentFileName
  - metadata
  - permStatusReady

- **methods:**
  - addLink
  - addRecipientListCryptFilter
  - addScript
  - encryptForRecipients
  - exportAsText
  - exportXFADATA
  - getLegalWarnings
  - getLinks
  - getOCGs
  - getPrintParams
  - importXFADATA
  - newPage
  - removeLinks
  - setAction
  - setPageAction
  - setPageTabOrder

**Error Objects**

- **properties:**
  - fileName
  - lineNumber
  - message
  - name

- **methods:**
  - toString

**Event Object**

- **properties:**
  - fieldFull
  - richChange
  - richChangeEx
  - richValue
### Acrobat 6.0 Changes

#### FDF Object

**Properties:**
- `deleteOption`
- `isSigned`
- `numEmbeddedFiles`

**Methods:**
- `addContact`
- `addEmbeddedFile`
- `addRequest`
- `close`
- `mail`
- `save`
- `signatureClear`
- `signatureSign`
- `signatureValidate`

#### Field Object

**Properties:**
- `buttonFitBounds`
- `comb`
- `commitOnSelChange`
- `defaultStyle`
- `radiosInUnison`
- `richText`
- `richValue`
- `rotation`

**Methods:**
- `getLock`
- `setLock`
- `signatureGetSeedValue`
- `signatureSetSeedValue`

#### Index Object

**Methods:**
- `build`

#### Link Object

**Properties:**
- `borderColor`
- `borderWidth`
- `highlightMode`
- `rect`

**Methods:**
- `setAction`
### OCG Object
- **properties:**
  - `name`
  - `state`
- **methods:**
  - `setAction`

### printParams Object
- **properties:**
  - `binaryOK`
  - `bitmapDPI`
  - `colorOverride`
  - `colorProfile`
  - `constants`
  - `downloadFarEastFonts`
  - `fileName`
  - `firstPage`
  - `flags`
  - `fontPolicy`
  - `gradientDPI`
  - `interactive`
  - `lastPage`
  - `pageHandling`
  - `pageSubset`
  - `printAsImage`
  - `printContent`
  - `printerName`
  - `psLevel`
  - `rasterFlags`
  - `reversePages`
  - `tileLabel`
  - `tileMark`
  - `tileOverlap`
  - `tileScale`
  - `transparencyLevel`
  - `usePrinterCRD`
  - `useT1Conversion`

### Report Object
- **properties:**
  - `style`
### Search Object

**properties:**
- `docInfo`
- `docText`
- `docXMP`
- `bookmarks`
- `ignoreAsianCharacterWidth`
- `jpegExif`
- `legacySearch`
- `markup`
- `matchWholeWord`
- `wordMatching`

### Security Object

**methods:**
- `chooseRecipientsDialog`
- `exportToFile`
- `importFromFile`

### SecurityHandler Object

**properties:**
- `digitalIDs`
- `directories`
- `directoryHandlers`
- `signAuthor`
- `signPDF`

**methods:**
- `newDirectory`

### SOAP Object

**properties:**
- `wireDump`

**methods:**
- `connect`
- `request`
- `response`
- `streamDecode`
- `streamEncode`
- `streamFromString`
- `stringFromStream`
Span Object

- properties:
  - alignment
  - fontFamily
  - fontStyle
  - fontWeight
  - fontStretch
  - text
  - textColor
  - textSize
  - strikethrough
  - subscript
  - superscript
  - underline

Spell Object

- properties:
  - languages
  - languageOrder

- methods:
  - customDictionaryClose
  - customDictionaryCreate
  - customDictionaryExport
  - customDictionaryOpen
  - ignoreAll

Thermometer Object

- properties:
  - cancelled
  - duration
  - value
  - text

- methods:
  - begin
  - end

Util Object

- methods:
  - printd
  - spansToXML
  - xmlToSpans
Modified in Acrobat 6.0

**Changed or Enhanced Objects, Methods, and Properties**

The following properties and methods have been changed or enhanced:

<table>
<thead>
<tr>
<th>App Object</th>
<th>methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>addMenuItem</code></td>
</tr>
<tr>
<td></td>
<td><code>alert</code></td>
</tr>
<tr>
<td></td>
<td><code>listMenuItems</code></td>
</tr>
<tr>
<td></td>
<td><code>listToolbarButtons</code></td>
</tr>
<tr>
<td></td>
<td><code>response</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Doc Object</th>
<th>properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>layout</code></td>
</tr>
<tr>
<td></td>
<td><code>zoomType</code></td>
</tr>
<tr>
<td>methods:</td>
<td><code>createDataObject</code></td>
</tr>
<tr>
<td></td>
<td><code>exportAsFDF</code></td>
</tr>
<tr>
<td></td>
<td><code>exportAsXFDF</code></td>
</tr>
<tr>
<td></td>
<td><code>exportDataObject</code></td>
</tr>
<tr>
<td></td>
<td><code>flattenPages</code></td>
</tr>
<tr>
<td></td>
<td><code>getField</code> (see Extended Methods)</td>
</tr>
<tr>
<td></td>
<td><code>getURL</code></td>
</tr>
<tr>
<td></td>
<td><code>importDataObject</code></td>
</tr>
<tr>
<td></td>
<td><code>importIcon</code></td>
</tr>
<tr>
<td></td>
<td><code>print</code></td>
</tr>
<tr>
<td></td>
<td><code>saveAs</code></td>
</tr>
<tr>
<td></td>
<td><code>spawnPageFromTemplate</code></td>
</tr>
<tr>
<td></td>
<td><code>submitForm</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Event Object</th>
<th>properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>changeEx</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Object</th>
<th>properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>name</code></td>
</tr>
<tr>
<td>methods:</td>
<td><code>buttonImportIcon</code></td>
</tr>
<tr>
<td></td>
<td><code>signatureInfo</code></td>
</tr>
<tr>
<td></td>
<td><code>signatureSign</code></td>
</tr>
<tr>
<td></td>
<td><code>signatureValidate</code></td>
</tr>
</tbody>
</table>
New Features and Changes

Acrobat 6.0 Changes

Extended Methods

The `doc.getField` method has been extended in Acrobat 6.0 so that it retrieves the `field` object of individual widgets. See Field Access from JavaScript for a discussion of widgets and how to work with them.
## Deprecated in Acrobat 6.0

<table>
<thead>
<tr>
<th>Search Object</th>
<th>properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>soundex</td>
</tr>
<tr>
<td></td>
<td>thesaurus</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Spell Object</th>
<th>methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>addDictionary</td>
</tr>
<tr>
<td></td>
<td>removeDictionary</td>
</tr>
</tbody>
</table>

## Introduced in Acrobat 6.0.2

The following properties and methods are introduced in Acrobat 6.0.2:

| XFA Object |
# Acrobat 5.0 Changes

## Introduced in Acrobat 5.0

<table>
<thead>
<tr>
<th>ADBC Object</th>
<th>methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>getDataSourceList</code></td>
</tr>
<tr>
<td></td>
<td><code>newConnection</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Annot Object</th>
<th>properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>alignment</code></td>
</tr>
<tr>
<td></td>
<td><code>AP</code></td>
</tr>
<tr>
<td></td>
<td><code>arrowBegin</code></td>
</tr>
<tr>
<td></td>
<td><code>arrowEnd</code></td>
</tr>
<tr>
<td></td>
<td><code>author</code></td>
</tr>
<tr>
<td></td>
<td><code>contents</code></td>
</tr>
<tr>
<td></td>
<td><code>doc</code></td>
</tr>
<tr>
<td></td>
<td><code>fillColor</code></td>
</tr>
<tr>
<td></td>
<td><code>hidden</code></td>
</tr>
<tr>
<td></td>
<td><code>modDate</code></td>
</tr>
<tr>
<td></td>
<td><code>name</code></td>
</tr>
<tr>
<td></td>
<td><code>noView</code></td>
</tr>
<tr>
<td></td>
<td><code>page</code></td>
</tr>
<tr>
<td></td>
<td><code>point</code></td>
</tr>
<tr>
<td></td>
<td><code>points</code></td>
</tr>
<tr>
<td></td>
<td><code>popupRect</code></td>
</tr>
<tr>
<td></td>
<td><code>print</code></td>
</tr>
<tr>
<td></td>
<td><code>rect</code></td>
</tr>
<tr>
<td></td>
<td><code>readOnly</code></td>
</tr>
<tr>
<td></td>
<td><code>rotate</code></td>
</tr>
<tr>
<td></td>
<td><code>strokeColor</code></td>
</tr>
<tr>
<td></td>
<td><code>textFont</code></td>
</tr>
<tr>
<td></td>
<td><code>type</code></td>
</tr>
<tr>
<td></td>
<td><code>soundIcon</code></td>
</tr>
<tr>
<td></td>
<td><code>width</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>methods:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>destroy</code></td>
</tr>
<tr>
<td></td>
<td><code>getProps</code></td>
</tr>
<tr>
<td></td>
<td><code>setProps</code></td>
</tr>
</tbody>
</table>
**New Features and Changes**

**Acrobat 5.0 Changes**

### App Object
**properties:**
- activeDocs
- fs
- plugIns
- viewerVariation

**methods:**
- addMenuItem
- addSubMenu
- clearInterval
- clearTimeout
- listMenuItems
- listToolbarButtons
- newDoc
- openDoc
- popUpMenu
- setInterval
- setTimeout

### Bookmark Object
**properties:**
- children
- color
- doc
- name
- open
- parent
- style

**methods:**
- createChild
- execute
- insertChild
- remove

### Color Object

**methods:**
- convert
- equal

### Connection Object

**methods:**
- newStatement
- getList
- getColumnList
Data Object

properties:
  - creationDate
  - modDate
  - MIMEType
  - name
  - path
  - size

Doc Object

properties:
  - bookmarkRoot
  - disclosed (5.0.5)
  - icons
  - info
  - layout
  - securityHandler
  - selectedAnnots
  - sounds
  - templates
  - URL

methods:
  - addAnnot
  - addField
  - addIcon
  - addThumbnails
  - addWeblinks
  - bringToFront
  - closeDoc
  - createDataObject
  - createTemplate
  - deletePages
  - deleteSound
  - exportAsXFDF
  - exportDataObject
  - extractPages
  - flattenPages
  - getAnnot
  - getAnnots
  - getDataObject
  - getIcon
  - getPageBox
  - getPageLabel
New Features and Changes

Acrobat 5.0 Changes

getPageNthWord
getPageNthWordQuads
getPageRotation
getPageTransition
getSound
importAnXFDF
importDataObject
importIcon
importSound
importTextData
insertPages
movePage
print
removeDataObject
removeField
removeIcon
removeTemplate
removeThumbnails
removeWebLinks
replacePages
saveAs
selectPageNthWord
setPageBoxes
setPageLabels
setPageRotations
setPageTransitions
submitForm
syncAnnotScan

Event Object properties:

changeEx
down
targetName
### Field Object

**properties:**

- `buttonAlignX`
- `buttonAlignY`
- `buttonPosition`
- `buttonScaleHow`
- `buttonScaleWhen`
- `currentValueIndices`
- `doNotScroll`
- `doNotSpellCheck`
- `exportValues`
- `fileSelect`
- `multipleSelection`
- `rect`
- `strokeColor`
- `submitName`
- `valueAsString`

**methods:**

- `browseForFileToSubmit`
- `buttonGetCaption`
- `buttonGetIcon`
- `buttonSetCaption`
- `buttonSetIcon`
- `checkThisBox`
- `defaultIsChecked`
- `isBoxChecked`
- `isDefaultChecked`
- `setAction`
- `signatureInfo`
- `signatureSign`
- `signatureValidate`

### FullScreen Object

**properties:**

- `backgroundColor`
- `clickAdvances`
- `cursor`
- `defaultTransition`
- `escapeExits`
- `isFullScreen`
- `loop`
- `timeDelay`
- `transitions`
- `usePageTiming`
- `useTimer`
### Global Object

**methods:**

- `subscribe`

### Identity Object

**properties:**

- `corporation`
- `email`
- `loginName`
- `name`

### Index Object

**properties:**

- `available`
- `name`
- `path`
- `selected`

### Plugin Object

**properties:**

- `certified`
- `loaded`
- `name`
- `path`
- `version`

### PPKLite Signature Handler Object (now listed under the SecurityHandler Object)

**properties**

- `appearances`
- `isLoggedin`
- `loginName`
- `loginPath`
- `name`
- `signInvisible`
- `signVisible`
- `uiName`

**methods:**

- `login`
- `logout`
- `newUser`
- `setPasswordTimeout`
<table>
<thead>
<tr>
<th>Object</th>
<th>Properties</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Report Object</strong></td>
<td>absIndent, color, absIndent</td>
<td>breakPage, divide, indent, outdent, open, mail, Report, save, writeText</td>
</tr>
<tr>
<td><strong>Search Object</strong></td>
<td>available, indexes, markup, maxDocs, proximity, refine, soundex, stem</td>
<td>addIndex, getIndexForPath, query, removeIndex</td>
</tr>
<tr>
<td><strong>Security Object</strong></td>
<td>handlers, validateSignaturesOnOpen</td>
<td>getHandler</td>
</tr>
</tbody>
</table>
Modified in Acrobat 5.0

- The console can act as an editor and can execute JavaScript code.
- The following properties and methods have been changed or enhanced:

<table>
<thead>
<tr>
<th>Object</th>
<th>Language</th>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spell Object</td>
<td>available</td>
<td>addDictionary, addWord,</td>
</tr>
<tr>
<td></td>
<td>dictionaryNames</td>
<td>check, checkText, checkWord,</td>
</tr>
<tr>
<td></td>
<td>dictionaryOrder</td>
<td>removeDictionary, removeWord,</td>
</tr>
<tr>
<td></td>
<td>domainNames</td>
<td>userWords</td>
</tr>
<tr>
<td>Statement Object</td>
<td>columnCount</td>
<td>execute, getColumn, getColumnArray,</td>
</tr>
<tr>
<td></td>
<td>rowCount</td>
<td>getRow, nextRow</td>
</tr>
<tr>
<td>Template Object</td>
<td>hidden</td>
<td>spawn</td>
</tr>
<tr>
<td></td>
<td>name</td>
<td></td>
</tr>
<tr>
<td>App Object</td>
<td>language</td>
<td></td>
</tr>
<tr>
<td></td>
<td>execMenuItem</td>
<td></td>
</tr>
<tr>
<td>Doc Object</td>
<td>exportAsPDF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>print</td>
<td></td>
</tr>
<tr>
<td></td>
<td>submitForm</td>
<td></td>
</tr>
<tr>
<td>Event Object</td>
<td>type</td>
<td></td>
</tr>
</tbody>
</table>
The section related to Event Object has been greatly enhanced to facilitate better understanding of the Acrobat JavaScript Event model.

**Deprecated in Acrobat 5.0**

The following properties and methods have been deprecated:

- **App Object**
  - fullscreen
  - numPlugIns
  - getNthPlugInName

- **Doc Object**
  - author
  - creationDate
  - creationDate
  - keywords
  - modDate
  - numTemplates
  - producer
  - title
  - getNthTemplate
  - spawnPageFromTemplate

- **Field Object**
  - hidden

- **TTS Object**
  - soundCues
  - speechCues

**Modified in Acrobat 5.05**

- A new symbol has been added to the quick bar denoting which methods are missing from Acrobat™ Approval™.
- In the Doc Object, the property disclosed has been added.
## Modified in Adobe 5.1 Reader

A new column has been added to the Quick Bars that summarize availability, and the meanings of the fourth and fifth columns has changed. They now indicate the availability of a property or method in the Adobe Reader and Acrobat Approval respectively.

- The symbols that appear in the fourth column indicate whether a property or method is available in Adobe Reader, and also whether access depends on document rights in the Acrobat 5.1 Reader.
- The fifth column indicates whether a property or method is available in Acrobat Approval.

Access to the following properties and methods has changed for the Adobe 5.1 Reader:

<table>
<thead>
<tr>
<th>Annot Object</th>
<th>properties:</th>
<th>Doc Object</th>
<th>properties:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>alignment</td>
<td>modDate</td>
<td>rect</td>
</tr>
<tr>
<td></td>
<td>AP</td>
<td>name</td>
<td>readOnly</td>
</tr>
<tr>
<td></td>
<td>arrowBegin</td>
<td>noView</td>
<td>rotate</td>
</tr>
<tr>
<td></td>
<td>arrowEnd</td>
<td>page</td>
<td>strokeColor</td>
</tr>
<tr>
<td></td>
<td>author</td>
<td>point</td>
<td>textFont</td>
</tr>
<tr>
<td></td>
<td>contents</td>
<td>points</td>
<td>type</td>
</tr>
<tr>
<td></td>
<td>doc</td>
<td>popupRect</td>
<td>soundIcon</td>
</tr>
<tr>
<td></td>
<td>fillColor</td>
<td>print</td>
<td>width</td>
</tr>
<tr>
<td></td>
<td>hidden</td>
<td></td>
<td></td>
</tr>
<tr>
<td>methods:</td>
<td>destroy</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>getProps</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>setProps</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|          | properties:          |            | importAnXFDF |
|          |                       |            | importDataObject |
|          |                       |            | mailDoc |
|          |                       |            | mailForm |
|          |                       |            | spawnPageFromTemplate |
|          |                       |            | submitForm |
|          |                       |            | syncAnnotScan |

<table>
<thead>
<tr>
<th>Template Object</th>
<th>methods:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>spawn</td>
<td></td>
</tr>
</tbody>
</table>
New Features and Changes

Acrobat 5.0 Changes