JScripter

Manual

PRERELEASE



Version 2027 (V12) — November 2026

Contents

Version 2027 (V12) — November 2026

Introduction

Scripting Features

Variable declaration....4
Object creation....4
Special Variable Types....4
UnitValueString....4
LineStyleString....5
FontName....5
ThreeStates....5
Exception Processing....5
Error Status....6
Functions and Recursion....6
Public Commands....7
Command: include....7
Command: quit....7
Command: pause....7
Command: repaint....8
Command: execute....8
Command: convertUnitValue....8
Command: dump....8

Application, Document Structure Objects

Main object “application”....12
Class: Document....15
Class: Collection....21
Class: Page....22
Class: DocumentPage....25
Class: AliasPage....27
Class: Spread....28
Class: AliasSpread....30
Class: Layer....31
Class: Guide....33
Class: PreviewInstructions....33

Object Types and Structure

Class: Objects....38
Class: Object....40
Class: GraphicObject....44
Class: LineObject....47
Class: FrameObject....49
Class: TextObject....52
Class: TextContent....54
Class: PictureObject....57
Class: PictureContent....58
Class: TextChains....64
Class: TextChain....65
Class: TableObject....65
Class: TableRow....70
Class: TableColumn....71
Class: TableCell....72
Class: TableSeparator....76
Class: GroupObject....78
Class: ChartObject....80
Class: ObjectAlias....82
Class: TextObjectAlias....83
Class: PictureObjectAlias....83

FormattedText Objects

Class: FormattedText....86
Class: TextRanges....119
Class: TextRange....119
Class: TextCursor....120

Object shapes

Class: Point....122
Class: PointList....123

Class: BoundingBox....125
Class: Matrix....126

Brushes and Colors

Introduction....128
Class: Brushes....128
Class: Brush....128
Class: ColorBrush....129
Class: BlendBrush....130
Class: Stop....131
Class: Rgb....131
Class: Cmyk....132
Class: Hsv....132
Class: Lab....133
Examples....133

StyleSheets

Class: StyleSheets....136
Class: StyleSheet....137
Class: CharacterStyleSheet....138
Class: ParagraphStyleSheet....139
Class: LayoutStyleSheet....143
Class: PictureStyleSheet....144
Class: GraphicStyleSheet....144
Class: TableStyleSheet....147
Class: RowColumnStyle....148
Class: RowColumnStyleAlt....148
Class: TableRowStyleSheet....149
Class: TableColumnStyleSheet....149
Class: TableCellStyleSheet....150

Document and Page Settings

Class: DocumentSettings....154
Class: PageSettings....156
Class: ChapterInfo....156
Class: DocumentStatistics....157
Class: SearchPaths....158

Preferences

Class: EmbeddedFontsPreferences....162
Class: LanguageInfo....162
Class: LanguagePreferences....162
Class: LayoutPreferences....163
Class: ObjectPreferences....163
Class: PagePreferences....164
Class: PreflightPreferences....164
Class: TextPreferences....165

Helped Data Objects

Class: Annotation....170
Class: Attributes....171
Class: AutomaticCharacterSpacing....171
Class: AutomaticCharacterWidth....171
Class: AutomaticWordSpacing....172
Class: BaselineGrid....172
Class: BaselinePosition....172
Class: ColorOverlay....173
Class: ColumnLines....173
Class: Comment....174
Class: DropCaps....174
Class: Enumerations....175
Class: FontSize....178
Class: GraphicFootnotes....178

Class: HeaderFooter....178
Class: Hyphenation....178
Class: ChangePictureColorSpaceSettings....179
Class: CharacterBackground....180
Class: CharacterFrame....181
Class: CharacterKerning....182
Class: CharacterPositionCorrection....182
Class: CharacterRule....182
Class: Image....183
Class: InnerGlow....184
Class: LayoutColumnsInfo....184
Class: LayoutFootnotes....185
Class: LineStyle....186
Class: LineNumbers....186
Class: Link....186
Class: Note....187
Class: OpenType....188
Class: OptimizePictureSettings....189
Class: OuterGlow....190
Class: OutlineOptions....190
Class: PackageSettings....191
Class: ParagraphBackground....191
Class: ParagraphFrame....192
Class: ParagraphRule....194
Class: PDFExportSettings....194
Class: Reflection....196
Class: Runaround....196
Class: Shadow....197
Class: SpellLang....198
Class: StrikethroughOptions....199
Class: TablesAbbreviation....199
Class: TablesBibliography....200

Class: TablesContents....200
Class: TablesIndex....201
Class: TablesPicture....201
Class: TablesRunningTitle....201
Class: Tabulator....202
Class: Tabulators....202
Class: TaggedPdfOptions....203
Class: TextEndnotes....203
Class: TextFootnotes....204
Class: TransparencyStop....205
Class: Transparency....205
Class: UnderlineOptions....205
Class: Variable....207
Class: WidowsOrphans....207

User Interface Objects

Class: MessageBox....210
Class: Directory....210
Class: File....211
Class: TextFile....212
Class: DataTable....214
Class: FileDialog....215
Class: ProgressReporter....216
Class: Dialog....216
Class: Label....218
Class: CheckBox....219
Class: ComboBox....219
Class: GroupBox....221
Class: RadioButton....222
Class: LineEdit....222
Class: TextEdit....223
Class: Button....224

Enumeration Objects Index

Index



Introduction

Version 2027 (V12) — November 2026

Introduction

JScripter is a VivaDesigner tool that allows you to manipulate documents using the JavaScript programming language. The scripts run in the JavaScript engine of Qt 6, which implements the ECMA-Script language standard (7th edition). This document describes JScripter's available functions and how to use them.

Examples
The document contains many examples that demonstrate the principles of the scripting language. Do not assume that all of the examples are complete and ready to run. They are intended to illustrate the feature mentioned in the text. To experiment with the examples in the real script, you may need to add code that is important for running the code fragment but not for explaining the appropriate feature.

Legend
The documentation uses various fonts to differentiate the text's various meanings. The following legend defines the text's meaning and the font used for it:

Font	Meaning of the text
Normal font	General description
`Courier`	Script examples
`Courier Bold`	Script commands
`Courier Italic`	Names of objects and classes, Jscripter keywords



Version 2027 (V12) — November 2026

Scripting Features

JScripter manipulates objects of various classes. JScripter uses a common syntax that is well known from other scripting languages.

Variable declaration

The declaration of a variable is realized by the keyword var followed by the variable name. The type of variable is specified by the type of value that is assigned to the variable.

```
var variableName1 = 10; // numeric variable
var variableName2 = new Rgb(0,0,0); // object variable
var variableName3 = [1,2,3,4,5]; // array of numbers
var variableName4 = [new Rgb(0,0,0), new Rgb(25,255,255)]; // array of
objects
```

Object creation

An object can be created by calling a constructor:

```
var myRgb = new Rgb(255, 100, 25);
```

or via the object provider (i.e. a function which creates an object inside and returns it as a result of the function call):

```
var myObjGroup = doc.objects().create(ObjectType.Group);
```

The parameter of the function ***create()*** is a value of the enumeration ObjectType, the type of the created object.

Each class of this manual is a global object with the name of the class. Classes with a constructor, e.g. Rgb, PageSettings, PDFExportSettings or TextFile, are created with ***new***. Each enumeration type is a global object as well; its values are written as ***Enumeration.Value***, e.g. ***ObjectType.Text*** or ***ErrorStatus.Ok***.

Further global objects are ***application*** (see the chapter “Main object application”), ***messageBox*** (see the chapter “Class: MessageBox”), ***appScriptsPath*** and ***userScriptsPath*** and the constants ***on***, ***off*** and ***undef*** of the type ThreeStates.

Special Variable Types

UnitValueString

The type ***UnitValueString*** defines the distance (length, width, space, …) with a unit. If the ***UnitValueString*** value is returned without a unit, the “pt” unit is presumed. The following units are supported:

```
Base:  “dtp”
Metric:“mm”, “cm”, “m”, “in”, “dz” (decimal inch), “q” (quart)
Didot system: “dd”, “c”
Pica system:  “pt”,”p”
Pixel: "px" (1 px = 1 pt; not for convertUnitValue)
```

The conversion function ***convertUnitValue*** is used for the conversion of a value to another unit.

Example 1:

```
"12mm", "10pt", "2in", ….
```

Example 2:

```
textObject.pagex = textObject.pagex + "+10mm";
textObject.pagey = "(5mm * 2) + 22pt + 1cm";

var res = convertUnitValue("125", "m", true, "pt");
```

LineStyleString

The ***LineStyleString*** is the name of a line style: ***"solid"***, ***"dotted"***, ***"dash-dot"***, ***"dash-dot-dot"***, ***"dashed-3-2"***, ***"dashed-3-1"***, ***"dashed-1-1"***, ***"dashed-1-2"***, ***"dashed-2-2"***, ***"dashed-2-3"***, ***"dashed-4-4"***, ***"dashed-2-1-1-1"***, ***"dashed-2-2-1-2"***, ***"thick-thick"***, ***"thick-thin"***, ***"thin-thick"***, ***"thin-thin"***, ***"thick-thin-thick"***, ***"thin-thick-thin"***, ***"triple"*** or ***"wavy"***. Upper and lower case do not matter. The names ***SolidLine***, ***DashLine*** (***"dashed-3-2"***), ***DotLine***, ***DashDotLine*** and ***DashDotDotLine*** are accepted as well. A property returns the name in lower case; an unknown name gives ***"solid"***.

FontName

The ***FontName*** is composed of the family name (i.e. "Arial") and font face (i.e. "Italic", "Bold") separated by a hyphen .

Examples:

```
"Arial-Regular", "Times New Roman-Bold Italic"
```

ThreeStates

The type ***ThreeStates*** takes three logical values, written as the global constants ***on*** (1), ***off*** (0) and ***undef*** (-1):

on - it is set

off - it is unset

undef - it is not defined

Exception Processing

Some methods generate exceptions when an error occurred. The error handling code has the following structure:

```
try {
  block of code to try
}
catch(err) {
  block of code to handle errors
}
finally {
  block of code to be executed regardless of the try / catch result
}
```

The *try* statement lets you test a block of code for errors.

The *catch* statement lets you handle the error.

The *finally* statement lets you execute code, after try and catch, regardless of the result.

Examples:

```
  try {
       var cb = doc.brushes.addColor("[Green]");
       // Color named [Green] exists than an exception occurs!
  } catch (err) {
       messageBox.information("Error:", "Error Desc: " + err.name + ', ' +
err.message);
  } finally {
       messageBox.warning("Warning:", "Finally after exception!");
  }
```

Error Status

Each object has a property *errorStatus* which is set after calling each method or property. The property *errorStatus* returns an error that occurred when the method or the property was processed. The method or the property returns default value in case of the error.

The property *errorStatus* returns following values of the enumeration ErrorStatus:

Ok – calling the property or the method has been successfully done without the error

IsNotProcessed – calling the property or the method has not been done

IsNotValid – the result value is not valid. The default value has been returned.

IsNotSet – the property is not set yet. The value is returned when the attribute is not set at all. For example, in the case of creating new stylesheet profile.

IsNotUsable – the property or the method is not usable in the context.

IsNotAccessToFileSystem – the property or the method is not accessible because the file system is protected.

The property *errorStatus* belongs to the object whose method or property was called, the object *application* included.

Example:

```
  var doc = application.currentDocument;
  var page = doc.documentPageAt(1000);
  if (doc.errorStatus == ErrorStatus.IsNotValid)
    messageBox.warning("Error", "The document has no page 1001.");
```

Functions and Recursion

Functions can be used for writing more complicated scripts. The function can be called with input parameters and can return a result. The functions can be saved in separate files and can be included in the script using the 'include' command. The functions can be called recursively.

Examples:

```
function function1(parameter1, parmeter2, ...)
{
  … function body

  if( … condition)
    // recursively calling
    function1(par1, par2, …);
  else
    // function returns result
    return result;
}

function function2(text)
{
  … function body
  function1( par1, par2, …);
}
```

Public Commands

Command: include

The command allows the inclusion of the content of another script file. The content is processed and all functions from this file are accessed after processing. The include command must be inserted in the file before the script code and be processed before the script is running. The scripts are searched for in two directories: application and user scripts directories.

Using:

include *(String fileName)*

where:

fileName is the name of the script file; for a file in a subfolder of a scripts folder its path relative to that folder, e.g. *"Library/tools.js"*. Each file is included only once.

Command: quit

Quits the running of the script prematurely.

Using:

quit ( ) ;

Command: pause

Suspends the running of the script for a time in milliseconds.

Using:

pause (long time);

where:

time is the number of milliseconds to wait

Command: repaint

Repaints the whole window with the document.

Using:

```
repaint();
```

Command: execute

The command starts an external program and returns at once; the script does not wait for the program. The first parameter is the program path, the next ones (up to five) are program parameters.

Using:

execute *(String programPath, String arg1, ..., String arg5)*

Command: convertUnitValue

Converts the value of the *UnitValueStrin*g type from one unit to another.

Using:

convertUnitValue*(String value, String toUnit, bool withUnit=false*, *String fromUnit = "pt")*

where:

value is the value that is converted. If the value contains a unit, the unit takes precedence over the *fromUnit* parameter.

toUnit – result unit name ("mm", "in", "pt", …)

fromUnit – source unit name ("mm", "in", "pt", …). Optional, the default value is "pt"

withUnit – flag, true = the unit will be added to the result after value. Optional, the default value is false.

Command: dump

Returns a value as text: objects and arrays as indented JSON text, other values as their text. It is useful for plain JavaScript objects, e.g. the results of *application.commands()* or *application.documentExportFormats*. For objects of the classes of this manual it returns only an internal class name.

Using:

dump *(value)*

where:

value is any JavaScript value.

Properties: appScriptsPath, userScriptsPath

Return the full paths to the application scripts or to the user's scripts. Both paths are searched automatically for scripts.

appScriptsPath *String* *Read*

userScriptsPath *String* *Read*

Examples

content of the file js_library.js:

```
include("js_library_2.js");

function myFunction(a, b)
{
     return (a * b) - 10;
}
```

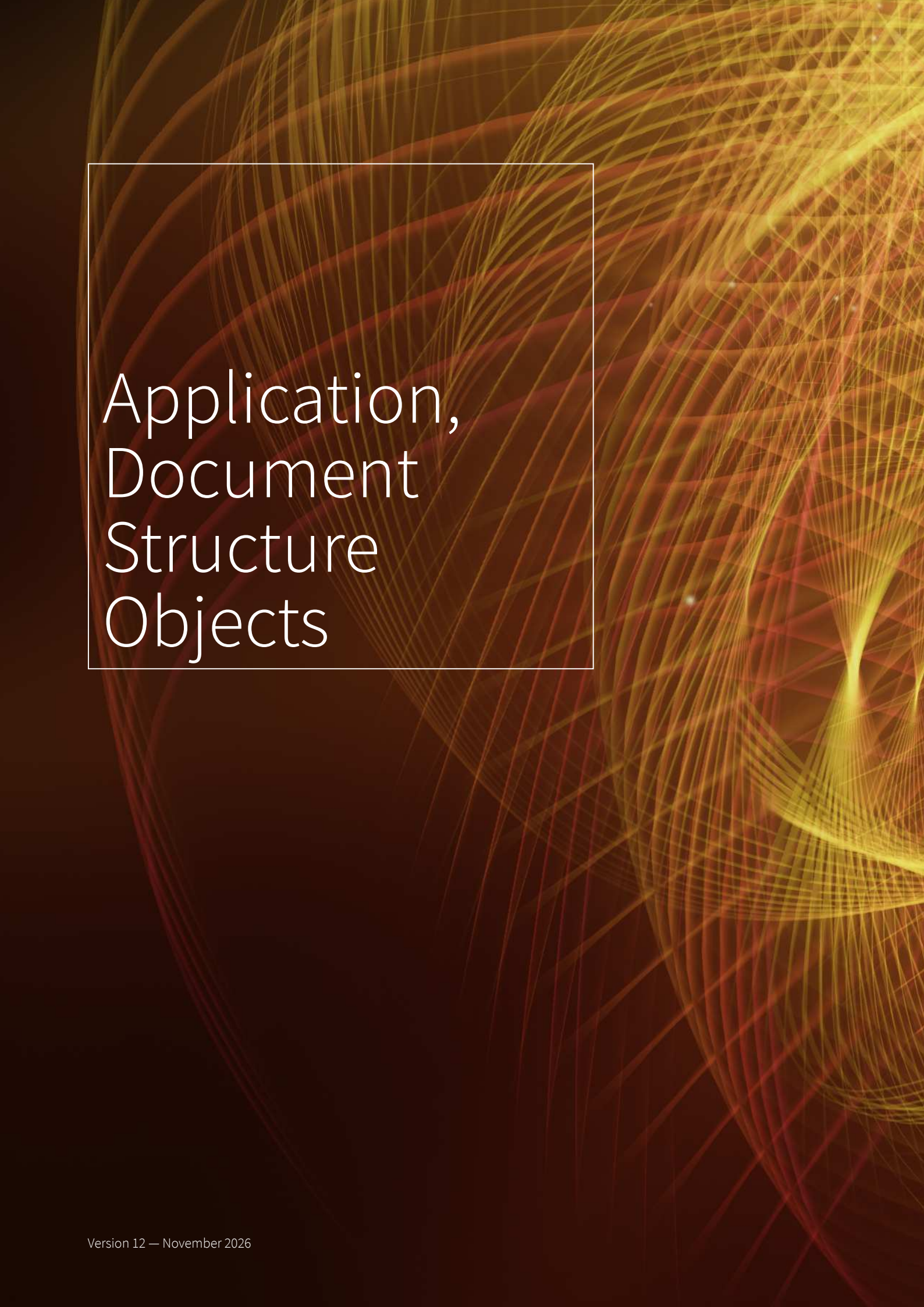
content of base script file:

```
include("js_library.js");
include("js_library_1.js");
pause(2000);

messageBox.information("Result", myFunction(11, 22));
messageBox.information("Msg:",
convertUnitValue(ds.columnDistance, "m", true, "pt"));
messageBox.information("Formats",
dump(application.documentExportFormats));

repaint();
quit();

messageBox.information("Both Scripts Paths",
appScriptsPath + "\n" + userScriptsPath);
```



Application, Document Structure Objects

Version 12 — November 2026

Application, Document Structure Objects

This chapter describes the hierarchy of object classes in relation to the application and documents, spreads, layers and pages in the application to understand the structure of object relationships, and shows examples of how to use them.

Main object “application”

VIVA Designer has just one built-in root object application. This object always exists. This object must be used for contained documents, pages, spreads etc.

Properties:

`identifier` *`String`* *`Read`*

Returns identifier of application VivaDesigner.exe.

`filePath` *`String`* *`Read`*

Returns full path to the VivaDesigner.exe file.

`absolutePath` *`String`* *`Read`*

Returns full path to the working folder of the application.

`documentCount` *`int`* *`Read`*

Returns number of opened documents in the application

`currentDocument` *`Document`* *`Read`*

Returns current Document object or *`null`* if no object exists.

`dictionariesPath` *`String`* *`Read`*

Returns the full path to the dictionaries folder of the application.

`language` *`String`* *`Read`*

Returns the language of the program interface as a language code, e.g. *`"en"`* or *`"de"`*.

`collectionCount` *`int`* *`Read`*

Returns the number of open collections (.cold files). See the chapter “Class: Collection”.

`currentCollection` *`Collection`* *`Read`*

Returns the collection in front or *`null`* if no collection is open.

`documentOpenFormats` *`Array`* *`Read`*

Returns the file formats that VivaDesigner opens as a document. Each entry is an object with the keys *`name`* (name of the format in the program language), *`extensions`* (array of the file extensions without “*.”, e.g. *`["idml"]`*), *`flags`* (array of texts) and *`owner`* (four-character ID of the add-in that provides the format; missing for the formats of the program itself).

`documentImportFormats` *`Array`* *`Read`*

Returns the file formats whose import filter turns the file into a new document when it is opened, e.g. PDF. The entries are the same as in *`documentOpenFormats`*.

documentSaveFormats *Array* *Read*

Returns the file formats in which a document can be saved. The entries are the same as in ***documentOpenFormats***.

documentExportFormats *Array* *Read*

Returns the file formats of the export filters, see ***Document.exportToXML()***. The entries are the same as in ***documentOpenFormats***.

Methods:

documentAt *(int index)* *Document*

Returns the document specified by its index or ***null*** if the document doesn't exist.

index: index of required document. The value must be in the range 0–(documentCount–1).

newDocument *(DocumentSettings settings, bool show)* *Document*

Creates and returns a new Document object. Returns ***null*** if a problem occurs when creating.

settings: Parameters of the new document.

show: Use true or false to show or hide the new document.

newDocument() *Document*

Creates and returns a new document using default document parameters The new document is set as a current document. Returns ***null*** if problem occurs.

openDocument *(String filePath, bool show, Object customParameters)* *Document*

Opens a document specified by its file path. Returns the opened Document object or ***null*** if document cannot be opened. An exception is caused when the document is open already.

filePath: Path to the opened document.

show: Use true or false to show or hide the new document.

customParameters: optional. Options for the import filter of the file format, e.g. for an IDML file ***{ImportFootnotes: "true", ApplyStyles: "WithStyleSheets"}***. The names depend on the filter. When options are given, the import filters for IDML and Word files do not show their options dialog.

setQrCodeToImage (*String url, String qrPictureId)* *bool*

Creates a QR code for the text ***url*** and places it as the picture of a picture object. Returns ***false*** if ***url*** or ***qrPictureId*** is empty.

qrPictureId: ID of the picture object as a hexadecimal number (the last part of ***Object.id***).

collectionAt *(int index)* *Collection*

Returns the open collection specified by its index or ***null*** if the collection doesn't exist.

index: Index of the collection. The value must be in the range 0 to (collectionCount - 1).

openCollection *(String filePath, bool show = true)* *Collection*

Opens a collection file (.cold) and returns the Collection object. A collection that is open already is returned as it is. Returns ***null*** for a missing file or a file of another format.

filePath: Path to the collection file.

show: Use true or false to show or hide the collection window. The default is true.

---

commands *(String filter = "")* *Array*

Returns the commands of the program that menus and keyboard shortcuts run, as an array of CommandInfo objects. Each command has an ID, the ID of the keyboard shortcut sets, e.g. *"edit.select-all"* or *"document.page.go"*.

filter: optional. Only commands whose ID, title or menu path contains this text.

A CommandInfo object has the following keys:

id: ID of the command.

title: Name of the command in the program language, as the menu shows it.

area, *section*: Where the menus show the command.

mode: The mode in which the command works: *"all"*, *"object"*, *"text"*, *"table"* or *"image"* (picture mode).

params: Array of the parameters (CommandParameter objects) with the keys *name*, *type* (*"string"*, *"number"*, *"boolean"*, *"object"*, *"array"* or *"any"*), *required* (bool), *default* (the value used when the parameter is left out), *values* (the allowed values, if they are limited), *path* (true for a file that the command reads; an absolute path) and *description*.

enabled: true if the command is possible right now (most commands need a selection).

available: true if the menus of the user offer the command (working mode, edition).

dialog: true if the command opens a dialog.

assistant: true if the command is released for the VIVA AI Assistant.

owner: Four-character ID of the add-in that registered the command; *""* for the program.

toolbarGroup: The group of the command in the control bar of its mode; *""* for none.

keys: Array of the current keyboard shortcuts, e.g. *["Ctrl+A"]*.

---

runCommand *(String id, Object params = {})* *Object*

Runs the command with the ID *id* on the active document, the same command that the menu and the keyboard shortcut run. Returns a CommandResult object. The method throws no exception.

params: optional. The parameters by the names that *params* of the CommandInfo object gives. A command without parameters runs as the menu runs it, with its dialog if it has one.

A CommandResult object has the following keys:

success: true if the command ran.

result: What the command returns, e.g. the page number for *"document.page.go"*.

errorCode: *""* in case of success, otherwise *"no-such-command"*, *"unknown-parameter"*, *"missing-parameter"*, *"bad-params"* (wrong type or value), *"file-denied"*, *"command-unavailable"*, *"command-disabled"*, *"command-refused"*, *"command-failed"*, *"no-document"* or *"not-supported"*.

error: *""* in case of success, otherwise an English text for *errorCode*.

The property *errorStatus* of the application is *Ok* in case of success, *IsNotValid* for *"no-such-command"*, *IsNotUsable* for *"command-unavailable"*, *"command-disabled"*, *"no-document"* and *"not-supported"*, *IsNotAccessToFileSystem* for *"file-denied"* and *IsNotProcessed* otherwise.

Examples:

```
var docCount = application.documentCount;
var docCurr = application.currentDocument;
var docFirst = application.documentAt(0);

// Find a command and run it with its parameter
var cmds = application.commands("page.go");
var r = application.runCommand("document.page.go", {page: 2});
if (!r.success)
  messageBox.warning("Command", r.errorCode + ": " + r.error);
```

Class: Document

Objects of the class *Document* are only objects which can be owned by the object *application*. One document object represents the whole document with its pages, spreads, texts, pictures, graphic objects etc., which is stored in one file.

Properties:

pageMode *PageMode* *Read/Write*

Contains information about the page mode in the document.

documentPageCount *int* *Read*

Contains the number of document pages. See the chapter “Class: DocumentPage” for more info about document pages. Returns -1 if a problem occurs.

aliasPageCount *int* *Read*

Contains the number of Alias pages. See the chapter “Class: AliasPage” for more info about Alias pages. Returns -1 if an error occurs.

documentSpreadCount *int* *Read*

Contains the number of document spreads. See the chapter “Class: Spread” for more info about document spreads. Returns -1 if an error occurs.

aliasSpreadCount *int* *Read*

Contains the number of Alias facing pages/spreads. Returns -1 if an error occurs.

layerCount *int* *Read*

Contains the number of layers. See the chapter “Class: Layer” for more info about document layers. Returns -1 if an error occurs.

brushes *Brushes* *Read*

Keeps an object of the class Brushes, which is a repository of all available brushes. New brushes can be added, unnecessary ones can be removed. See chapter “Class: Brushes” for more info about brushes.

styleSheets *StyleSheets* *Read*

Keeps an object of the class StyleSheets, which is a repository of all available style sheets. New style sheets can be added, unnecessary ones can be removed. See chapter “Class: StyleSheets” for more info about style sheets.

textCursor *TextCursor* *Read*

Keeps an object of the class TextCursor. This object contains information about the current position of the text cursor and/or the current selection. It also gives opportunity to set a new cursor position. See the chapter “Class: TextCursor” for more information about the text cursor.

textPreferences *TextPreferences* *Read/Write*

Keeps an object of the class TextPreferences. See the chapter “Class: TextPreferences” for more information about text preferences.

preflightPreferences *PreflightPreferences* *Read/Write*

Keeps an object of the class PreflightPreferences. See the chapter “Class: PreflightPreferences” for more information about preflight preferences.

languagePreferences *LanguagePreferences* *Read/Write*

Keeps an object of the class LanguagePreferences. See the chapter “Class: LanguagePreferences” for more information about language preferences.

objectPreferences *ObjectPreferences* *Read/Write*

Keeps an object of the class ObjectPreferences. See the chapter “Class: ObjectPreferences”.

layoutPreferences *LayoutPreferences* *Read/Write*

Keeps an object of the class LayoutPreferences. See the chapter “Class: LayoutPreferences”.

pagePreferences *PagePreferences* *Read/Write*

Keeps an object of the class PagePreferences. See the chapter “Class: PagePreferences”.

embeddedFontsPreferences *EmbeddedFontsPreferences* *Read/Write*

Keeps an object of the class EmbeddedFontsPreferences. See the chapter “Class: EmbeddedFontsPreferences”.

documentStatistics *DocumentStatistics* *Read/Write*

Keeps an object of the class DocumentStatistics. See the chapter “Class: DocumentStatistics” for more information.

searchPaths *SearchPaths* *Read/Write*

Keeps an object of the class SearchPaths. See the chapter “Class: SearchPaths” for more information about search paths.

filePath *String* *Read*

Contains the full file path including the file name of the document.

currentPage *DocumentPage* *Read*

Returns a DocumentPage object, which represents the page with the current cursor.

currentSpread *Spread* *Read*

Returns the current spread. See the chapter “Class: Spread” for more information about document spreads.

currentLayer *Layer* *Read/Write*

Returns the current document layer. See the chapter “Class: Layer” for more information about layers.

currentText *FormattedText* *Read*

Returns a FormattedText object which represents the text where the cursor is currently placed.

pageWidth *float* *Read*

Returns the page width of the document in pt (File > Document Setup). See ***Page.width*** for the size of a single page.

pageHeight *float* *Read*

Returns the page height of the document in pt (File > Document Setup).

documentSettings *DocumentSettings* *Read/Write*

Keeps an object of the class DocumentSettings with the settings of the document (File > Document Setup). A property changed there changes the document at once. See the chapter “Class: DocumentSettings”.

toggleColumn *int* *Read/Write*

Get/Set the column of the page arrangement in which the right-hand pages begin: pages in a column with a smaller number are left-hand pages. The default is 2. See ***Page.layoutPositionColumn***.

currentPictureObject *PictureObject* *Read*

Returns the current picture object (the picture object that is edited in picture mode) or ***null*** if there is none.

Methods:

objects(*WhichSpreads spreads = Document, WhichObjects objects = All, ObjectType objectType = All*) *Objects*

Returns an object of the class Objects. This object is a repository and an object provider of all document objects of the classes TextObject, PictureObject, TableObject, GraphicObject, LineObject, GroupObject. See examples of how to use the object provider in your JScript program.

spread: filter for searching for objects, the default is WhichSpreads:Document.

objects: filter for searching for objects, the default is WhichObjects:All.

objectType: filter for searching for object type. See the enumeration ObjectType

objectCount(*WhichSpreads spreads = Document, WhichObjects objects = TopLevel*) *int*

Returns the number of objects selected by the filters ***spreads*** and ***objects*** (see ***objects()***).

getObject(*int index, WhichSpreads spreads = Document, WhichObjects objects = TopLevel*) *Object*

Returns the object at the position ***index*** (0-based) in the list selected by the filters, or ***null***.

textChains(*WhichSpreads whichSpreads = Document, WhichObjects whichObjects = All*) *TextChains*

Returns an object of the class TextChains with the text chains of the objects selected by the filters. See the chapter “Class: TextChains”.

---

documentPageAt*(int index)* *DocumentPage*

Returns the DocumentPage object at the position which is specified by the parameter index. Returns the required DocumentPage or ***null*** if the page doesn’t exist.

index: Index of the required document page. The value must be in range 0 – (pageCount – 1).

---

addDocumentPage*(int count, PageSettings settings = null, AliasPage assignToAliasPage = null)* *DocumentPage*

The method adds new pages to the document. Returns the first created ***DocumentPage*** object or ***null*** if no page is created.

count: Number of added pages. The default value is 1.

settings: page settings. See PageSetting object.

assignToAliasPage: assigned AliasPage. See AliasPage description.

---

aliasPageAt *(int index)* *AliasPage*

Returns an AliasPage object at the position which is specified by the parameter index. Returns the required AliasPage or ***null*** if the page doesn’t exist.

index: Index of the required Alias page. The value must be in the range 0 – (aliasPageCount – 1).

---

aliasPageByName *(String name)* *AliasPage*

Returns an AliasPage object with the name which is specified by parameter name.

name: Name of the required Alias page.

Returns the required AliasPage or ***null*** if the page doesn’t exist.

---

addAliasPage *(String name, PageType type = SinglePage)* *AliasPage*

The method adds a new Alias page to the document. Returns the added Alias page.

name: Name of the new Alias page.

type: Page Type (See enumeration type PageType). The default is SinglePage.

---

gotoPage *(DocumentPage or AliasPage page)* *bool*

Makes the specified page current. Returns true in case of success.

page: Target page. This can be DocumentPage or AliasPage.

---

removeDocumentPages *(int fromPageNumber, int count)* *bool*

Removes several document pages from the document. Returns true in case of success.

fromPageNumber: Index of the first page removed The index is 0-based. The default value is 0.

count: Number of removed pages. The default value is 1.

---

documentSpreadAt *(int index)* *Spread*

Returns the Spread object at the position which is specified by parameter index.

index: Index of the required spread. The value must be in range 0 – (pageCount – 1).

---

aliasSpreadAt *(int index)* *AliasSpread*

Returns the Alias spread object at the position specified by parameter index.

index: Index of the required spread. The value must be in the range 0 – (pageCount – 1).

---

addLayer() *Layer*

Adds a new document layer. Returns the new layer.

---

layerAt*(int index)* *Layer*

Gets a layer which is specified by the term index.

index: Index of the required layer. The value must be in range 0 – (layerCount – 1).

---

moveLayer*(Layer layer, int beforeIndex)* *int*

Moves the specified layer to a position which is given by the index parameter. Returns the new index of the layer or -1 if the layer cannot be moved.

layer: The layer which has to be moved.

beforeIndex: Index of the layer before which the given layer has to be moved. The value, which is equal to the layerCount, moves the layer to the end of the list of layers.

---

closeAndSave() *void*

Closes the document. Before closing, the system asks a user if he wants to save the document before it will be closed.

---

close*()* *void*

Closes the document immediately. There is no request to save the document. If the document was changed and unsaved all changes are lost.

---

save*(String filePath)* *bool*

Save the document with the name 'filePath'.

---

exportToXML*(String filePath, Object parameters)* *bool*

Exports the document with the export filter that matches the file extension of *filePath*, e.g. .idml, .docx, .html, .epub, .vjson or .xml (VivaXML). See *application.documentExportFormats*. Returns true in case of success.

filePath: Path of the exported file.

parameters: optional. Options for the export filters that take them; the other filters ignore them:

IDML (.idml): *showDialog* (bool, default true). With *false* the document is exported without the options dialog, with the options last confirmed in it.

HTML (.html): *showDialog* as for IDML, and the options of the dialog: *allPages*, *startPage*, *endPage*, *createNavigation*, *centerPage*, *createPageBorders* and *alwaysCreateSinglePages*.

Word (.docx): *showDialog* (bool, default false), *"Mode"* (*"layout"* or *"text"*), *"Reading order"* (*"articles"* or *"hierarchy"*), *"Unmappable"* (*"image"* or *"omit"*), *"Image resolution"* (ppi; 0 keeps the original resolution), *"Embed fonts"* (bool), *"First page"* and *"Last page"* (page numbers).

Example:

```
doc.exportToXML("C:/Export/Book.idml", {showDialog: false});
```

exportToPDF*(PDFExportSettings settings)* *bool*

Exports the document as a PDF file into the path ***settings.outputPath***. Returns true in case of success. See the chapter “Class: PDFExportSettings”.

package*(String outputFolderPath, PackageSettings settings)* *bool*

Collects the document with its pictures and fonts in the folder ***outputFolderPath***, which must exist. Returns true in case of success. See the chapter “Class: PackageSettings”.

setUserProperty*(String ident, String data)* *void*

Set string data as user property with name ident. The data are ”base64” encoded into the document file to the part ”<uni:userInfos> <uni:base64Entry>”

userProperty*(String ident) String*

Get user property string with name ***ident***. The data are ‘base64’encoded into the document file.

Returns empty string if no item is found.

containsUserProperty*(String ident) bool*

Check if user property string with name ***ident*** exists

userXMLProperty*(String ownerId, String entryId)* *String*

Get user property string from the document file from the part ‘<uni:userInfos> <uni:xmlEntry>’

setUserXMLProperty*(String ownerId, String entryId, String data)* *void*

Set string data as user property to the document file to the part ‘<uni:userInfos> <uni:xmlEntry>’

updateView() *void*

Updates the view of the document.

copyObjectPreferences(*Document sourceDocument)* *void*

Copy the object preferences from the source document.

updateCustomPropertis*(Document sourceDocument)* *void*

Update the custom properties of the object preferences from the source document.

Example 1:

```
// Open and show an existing document
var doc = application.openDocument("C:/Documents/Example1.desd", true);
if (doc) {
  // Show file name from which the file was opened
  messageBox.information("Opened Document", doc.filePath);

  // Add 1 new empty page to the document
  var newPage = doc.addDocumentPage();
  // Manipulate the new page here:
  // Create a new text object and add it to the new page
  var objTxtFrm = doc.objects().create(ObjectType.Text);
  newPage.add(objTxtFrm);

  // Close the document with the 'Save' request
  doc.closeAndSave();
```

```
}
```

Example 2:

```
var doc = application.currentDocument;
if (doc) {
  for (var k = 1; k < 20; k++) {
    //get the count of selected objects
    var objs = doc.objects(WhichSpreads.All,WhichObjects.Selected);
    var arrSelLen = objs.length;

    //... do something

    repaint();
    pause(1000);
  }
}
```

Example 3:

```
doc.setUserProperty("IdentName", "IdentDATA");
if(doc.containsUserProperty("IdentName"))
  messageBox.information("Msg2:",doc.userProperty("IdentName"));
```

Class: Collection

An object of the class Collection represents an open collection (a .cold file): documents in a fixed order that are exported together as one PDF file. ***application.openCollection()*** opens a collection, ***application.currentCollection*** returns the collection in front.

Properties:

filePath	*String*	*Read*

Contains the full file path of the collection.

title	*String*	*Read/Write*

Get/Set the title of the file information of the collection. Title, description and keywords are written into the exported PDF file.

description	*String*	*Read/Write*

Get/Set the description of the file information.

keywords	*String*	*Read/Write*

Get/Set the keywords of the file information.

language	*Language*	*Read/Write*

Get/Set the main language of the collection. It is the language of the exported PDF file (tagged PDF only). ***Language.NoLang*** means no language of its own. See the enumeration type Language.

documentCount	*int*	*Read*

Returns the number of documents of the collection, hidden documents included.

Methods:

documentPathAt *(int index)*	*String*

Returns the path that the collection stores for the document at the position ***index***; the file may be missing. Returns an empty string if ***index*** is not in the range 0 to (documentCount - 1).

exportToPDF *(PDFExportSettings settings)* *bool*

Exports the visible documents of the collection as one PDF file into the path ***settings.outputPath***. Missing documents are left out, an existing file is replaced. Returns true only if the file was written. See the chapter “Class: PDFExportSettings”.

save *(String filePath = "")* *bool*

Saves the collection, under ***filePath*** if it is given. Returns false for a new collection without a file and for a ***filePath*** that does not end in .cold; nothing is written then.

Example:

```
var col = application.openCollection("C:/Work/Annual Report.cold", true);
if (col) {
  col.title = "Annual Report 2026";
  var pdf = new PDFExportSettings();
  pdf.outputPath = "C:/Work/Annual Report.pdf";
  if (col.exportToPDF(pdf))
    col.save();
}
```

Class: Page

The object of class ***Page*** represents one page.

Properties:

physicalNumber *int* *Read*

Contains the physical page number. Page numbers start from 1 in contrast to the page index, which starts from 0.

isAliasPage *bool* *Read*

Returns true if the page is an Alias page. Otherwise it returns false.

spread *Spread* *Read*

Returns a ***Spread*** object that belongs to a Page object.

offsetOnSpreadX *float* *Read*

Returns the X coordinate of the page offset on the spread.

offsetOnSpreadY *float* *Read*

Returns the Y coordinate of the page offset on the spread.

layoutPositionRow *int* *Read/Write*

Get/Set the row of the page in the page arrangement of the document.

layoutPositionColumn *int* *Read/Write*

Get/Set the column of the page in the page arrangement of the document. Pages in a column with a smaller number than ***Document.toggleColumn*** are left-hand pages.

guideCount *int* *Read*

Returns the number of guides of the page.

width *float* *Read*

Returns the width of the page in pt as shown on screen; for a landscape page it is the longer side. See *DocumentPage.setSize()* to change the size.

height *float* *Read*

Returns the height of the page in pt as shown on screen.

pageFormat *String* *Read*

Returns the name of the page format, e.g. *"DIN/A4"*; empty for a custom size.

landscape *bool* *Read*

Returns true if the page is in landscape orientation.

Methods:

add*(Object obj)* *void*

Adds an object to the page. The object can be of the class *TextObject*, *PictureObject*, *TableObject*, *GraphicObject*, *LineObject*, *GroupObject*.

obj: inserted object of the appropriate class

objects*(WhichObjects type = TopLevel, ObjectType objectType = All)* *Objects*

Returns an object of the class Objects. This object is a repository and an object provider of all page objects of the classes *TextObject*, *PictureObject*, *TableObject*, *GraphicObject*, *LineObject* and *GroupObject*.

type: Type of objects that are provided by the obtained “Objects” object. See the enumeration type *WhichObjects* at the end of this chapter.

objectType: See the enumeration ObjectType.

objectCount*(WhichObjects objects = TopLevel)* *int*

Returns the number of objects on the page selected by the filter *objects*.

getObject*(int index, WhichObjects objects = TopLevel)* *Object*

Returns the object at the position *index* (0-based) in the list selected by the filter, or *null*.

deleteObjects() *int*

Removes all objects from inside the page. Returns the number of removed objects.

deleteSelectedObjects() *int*

Removes selected objects from inside the page. Returns the number of removed objects.

setUserProperty*(String ident, String data)* *void*

Set string data as user property with name ident. The data are ‘base64’encoded into the document file to the part ‘<uni:userInfos> <uni:base64Entry>’

userProperty*(String ident)* *String*

Get user property string with name ident. The data are ‘base64’encoded into the document file to the part ‘<uni:userInfos> <uni:base64Entry>’

Returns empty string if no item is found.

---

containsUserProperty*(String ident)* *bool*

Check if user property string with name ident exists.

---

userXMLProperty*(String ownerId, String entryId)* *String*

Returns the custom part of the document description in the XML document.

---

setUserXMLProperty*(String ownerId, String entryId, String data)* *void*

Sets the custom part of the document description in the XML document.

---

addHorizontalGuide(String position, Layer layer) *int*

Adds a horizontal guide to the page. Returns the ID of the new guide, 0 if no guide was created, or -1 if *position* is not a valid measure.

position: Distance of the guide from the top edge of the page as UnitValueString, e.g. *"50mm"*.

layer: The layer of the guide. It must be given.

---

addVerticalGuide(String position, bool fitToPage, Layer layer) *int*

Adds a vertical guide to the page. Returns the ID of the new guide, 0 if no guide was created, or -1 if *position* is not a valid measure.

position: Distance of the guide from the left edge of the page as UnitValueString.

fitToPage: has no effect at present.

layer: The layer of the guide. It must be given.

---

guide *(int id)* *PageGuide*

Returns a PageGuide object for the guide with the ID *id*. Its property *isNull* is true if the page has no guide with this ID. See the chapter “Class: Guide”.

---

removeGuide(int id) *bool*

Removes the guide with the ID *id*. Returns true in case of success.

---

clearGuides() *void*

Removes all guides of the page.

---

preview *(String x, String y, String width, String height, PreviewInstructions pi, String fileName, int quality)* *bool*

Saves a picture of the page as an image file. Returns true if the file was written.

x, *y*: have no effect at present; use *"0"*.

width, *height*: Maximum size of the picture in pixels, e.g. *"800"*. A value with a unit is converted to pt; 1 pt gives 1 pixel.

pi: A PreviewInstructions object with scale, margins and background, or *null* for the defaults. See the chapter “Class: PreviewInstructions”.

fileName: Path of the image file. The file extension sets the format, e.g. .png or .jpg.

---

quality: Quality of the compression from 0 to 100; -1 for the default of the format.

---

preview *(String width, String height, String fileName, int quality) bool*

Saves a picture of the whole page that fits into *width* x *height* pixels, with the default PreviewInstructions.

Example:

```
var page = application.currentDocument.documentPageAt(0);
page.preview("600", "600", "C:/Temp/page1.png", -1);
// at 144 ppi, without the page border
var pi = new PreviewInstructions();
pi.scale = 200;
pi.withFrame = false;
page.preview("0", "0", "5000", "5000", pi, "C:/Temp/page1.jpg", 90);
```

Class: DocumentPage

The object of the class **DocumentPage** represents one particular page of the appropriate document. The class extends the root class Page and adds another properties, which are specific for this type of object. JScript can add one or more new pages and remove pages.

The page is a container of other objects which are instances of the classes *TextObject*, *PictureObject*, *TableObject*, *GraphicObject*, *LineObject* and *GroupObject*.

Properties:

---

logicalNumber *int* *Read*

Contains the logical page number. The page numbers start from 1 in contrast with the page index, which starts from 0.

---

label *String* *Read*

Contains textual label of the page. Normally it is same like logical number.

---

chapterInfo *ChapterInfo* *Read/Write*

Sets or gets the chapter info object. For more information about the chapter info object see the chapter “Class: ChapterInfo” in this documentation.

---

aliasPage *AliasPage* *Read*

Contains a reference to the Alias page. If the page has no Alias page, the value is *null*.

Methods:

---

setAliasPage*(AliasPage page, bool applyContent)* *bool*

Set the Alias page of the document page. Returns true in case of success.

page: AliasPage which is set as a template of the document page.

applyContent: If true the nested objects will be set to the document page from the Alias page

---

remove() *bool*

Removes a document page from the document. Returns true in case of success.

---

setSize *(String width, String height, String format = "")* *bool*

Gives the page its own size; the other pages keep their size. A width larger than the height makes the page landscape. Returns true in case of success.

width, *height*: Size of the page as UnitValueString, e.g. *"148mm"* and *"210mm"* for A5 portrait or *"297mm"* and *"210mm"* for A4 landscape.

format: optional. Name of the page format. Without it VivaDesigner looks up the format of the format list that matches the size (empty for a custom size). With the ID of a format of the format list (File > Document Setup), e.g. *"LinkedIn/Personal Cover/Header"*, the size must be the listed one or the turned one; then the page keeps its orientation when the Page Format dialog is closed with OK. Any other size leaves the page unchanged (errorStatus *IsNotProcessed*). A name that is not in the format list is stored as the format name of the page.

---

createChart *(Object settings)* *ChartObject*

Creates a chart object of the chart add-in on the page, as the Chart Object Tool draws it, in one step of Edit > Undo ("Insert Chart"). The document of the page must be the active document. The method never returns *null*: if the chart cannot be created, *isChart* of the returned object is false and its *lastResult* says why, e.g. *"not-licensed"* without a licence for charts. The property *errorStatus* of the page is set as for ChartObject. See the chapter "Class: ChartObject".

settings: an object with the following keys. Measures are numbers (pt) or UnitValueStrings such as *"20mm"*.

x, *y*: Position of the top left corner of the chart area on the page. Required.

width, *height*: Size of the chart area; the default is 128 mm x 84 mm. A smaller size than 48 mm x 36 mm is enlarged from the top left corner.

type: *"column"* (default), *"bar"*, *"line"*, *"area"*, *"pie"*, *"ring"* or *"combo"* (columns; the last series is a line on a second value axis).

title: The title of the chart; *""* for none.

legend: Show the legend (bool); the default is true.

categories: Array of the texts of the categories (category axis, pie slices).

series: Array of the series, each *{name, values}*; a value is a number or *null* (a gap). Without categories and series the chart shows sample values.

options: The look of the chart, as for *ChartObject.setOptions()*.

name: Name of the object as the Layers palette shows it; the default is "Chart".

An unknown key gives the errorCode *"unknown-parameter"*, a wrong type *"bad-params"*, a missing *x* or *y* *"missing-parameter"*; nothing is created then.

Examples:

```
var doc = application.currentDocument;
if (doc) {
  // Create a new object, which is then inserted to a page
  var obj = doc.objects().create(ObjectType.Text);
  // Configure the object
  with (obj) {
    name = "Text Frame 1";
    form = FrameForm.Ellipse;
```

```
      lineWidth = 12;
      pagex = "20mm";
      pagey = "100mm";
      height = "40mm";
      width = "170mm";
    }
    // create page setting object
      var pgs = new PageSettings();
    pgs.insertPosition = PageInsertPosition.AfterCurrentPage;
    pgs.afterPageNumber = 1;
    pgs.copyObjects = false;
    pgs.copyOnlySelectedObjects = false;
    pgs.copyAsAlias = false;
    pgs.linkToTextChain = false;
    // add a new page to the document
    var newPage = doc.addDocumentPage(1, pgs);
    // ... or a function call with default parameters can be used
    //var newPage = doc.addDocumentPage();
    if (newPage) {
      // Insert the new object to the new page
      newPage.add(obj);

      // Show logical number of the page
      messageBox.information("Page Number", newPage.logicalNumber);
    }
    // Get last page of the document
    var lastPage = doc.documentPageAt(doc.documentPageCount - 1);
    if (lastPage)
      // Remove all objects from the last page
      lastPage.deleteObjects();

    // Give the last page the size A4 landscape and add a pie chart
    if (lastPage && lastPage.setSize("297mm", "210mm")) {
      var chart = lastPage.createChart({x: "20mm", y: "40mm", type: "pie",
        categories: ["A", "B"], series: [{name: "2026", values: [30,
70]}]});
      if (!chart.isChart)
        messageBox.warning("Chart", chart.lastResult.error);
    }
  }
```

Class: AliasPage

The class ***AliasPage*** represents a kind of page template. The class extends the root class ***Page*** and adds another properties, which are specific for this type of object. The author of the document can create one or more Alias pages and assign them to any document page. Each document page may have 0 or 1 Alias page. The Alias page is useful for the creation of page elements which are repeated on several or all document pages. The main advantage of Alias pages is the possibility of editing these common page elements in one operation. All changes are of course automatically visible on all pages that use the appropriate Alias page.

Properties:

name *String* *Read/Write*

The property is used to read or set name of the Alias page.

pageType *PageType* *Read*

Returns the page type of the Alias page: ***PageType.SinglePage*** or ***PageType.FacingPages***.

Methods:

remove() *bool*

Removes the Alias page. Returns true in case of success.

Class: Spread

A spread is an area in which document pages are located. Each document has one or more spreads. Each page is located in one of these spreads. A spread may contain more than 1 page. Each object located on a page is also located on the appropriate spread. The object is considered as being on the page if at least 50% of its area lies on the page surface. Otherwise the object is not on the page but is still on the spread.

Properties:

pageCount *int* *Read*

Contains the number of pages located on the appropriate spread.

guideCount *int* *Read*

Returns the number of guides of the spread. Guides on a page belong to the page, see ***Page.guideCount***.

Methods:

add*(Object obj, Object afterObj = null) void*

Add an object to the spread.

obj: Object which is added to the spread.

afterObj: optional. The object is inserted after this object in the list of objects of the spread (as ***objects()*** returns it). Without ***afterObj*** it is inserted at the beginning of the list.

objects*(WhichObjects type = TopLevel, ObjectType objectType = All) Objects*

Returns an object of the class ***Objects***. This object is a repository and an object provider of all spread objects of the classes ***TextObject***, ***PictureObject***, ***TableObject***, ***GraphicObject***, ***LineObject*** and ***GroupObject***.

type: Type of objects which are provided by the “obtained Objects” object. See the enumeration type ***WhichObjects*** at the end of this chapter.

objectType: See the enumeration ObjectType.

objectCount*(WhichObjects objects = TopLevel) int*

Returns the number of objects on the spread selected by the filter ***objects***.

getObject*(int index, WhichObjects objects = TopLevel) Object*

Returns the object at the position ***index*** (0-based) in the list selected by the filter, or ***null***.

---

pageAt*(int index) Page*

Returns the page at the position which is identified by the index: a ***DocumentPage***, or an ***AliasPage*** in an Alias spread.

index: Index of the required page. Its value must be in the range 0 – (pageCount – 1);

Returns the required page or ***null*** if the required page doesn't exist.

---

objectPage*(Object object) Page*

Returns the page (***DocumentPage*** or ***AliasPage***) in which the specified object is located.

object: The object for which the container page searches.

Returns the container page or ***null*** if the object is not located in any page.

---

setUserProperty*(String ident, String data) void*

Set string data as user property with name ident. The data are 'base64'encoded into the document file to the part '<uni:userInfos> <uni:base64Entry>'

---

userProperty*(String ident) String*

Get user property string with name ident. The data are 'base64'encoded into the document file to the part '<uni:userInfos> <uni:base64Entry>'

Returns empty string if no item is found.

---

containsUserProperty*(String ident) bool*

Check if user property string with name ident exists.

---

userXMLProperty*(String ownerId, String entryId) String*

Get user property string from the document file from the part '<uni:userInfos> <uni:xmlEntry>

---

setUserXMLProperty *(String ownerId, String entryId, String data) void*

Set string data as user property to the document file to the part '<uni:userInfos> <uni:xmlEntry>

---

addHorizontalGuide *(String position, Layer layer) int*

Adds a horizontal guide across the spread. Returns the ID of the new guide, 0 if no guide was created, or -1 if ***position*** is not a valid measure.

position: Distance of the guide from the top edge of the spread as UnitValueString.

layer: The layer of the guide. It must be given.

---

addVerticalGuide *(String position, VerticalGuideZone zone, Layer layer) int*

Adds a vertical guide on the pasteboard beside the pages. Returns the ID of the new guide, 0 if no guide was created, or -1 if ***position*** is not a valid measure.

position: Distance of the guide from the pages as UnitValueString.

zone: ***VerticalGuideZone.BeforePages***: left of the first page, measured from its left edge; ***Vertical-GuideZone.AfterPages***: right of the last page, measured from its right edge.

layer: The layer of the guide. It must be given.

guide *(int id)* *SpreadGuide*

Returns a SpreadGuide object for the guide with the ID ***id***. Its property ***isNull*** is true if the spread has no guide with this ID. See the chapter "Class: Guide".

removeGuide *(int id)* *bool*

Removes the guide with the ID ***id***. Returns true in case of success.

clearGuides() *void*

Removes all guides of the spread.

Examples:

```
var doc = application.currentDocument;
if (doc) {
  // Show number of spreads in a current document
  messageBox.information("Spread Count", doc.documentSpreadCount);

  // Get 1st spread of the document
  var spr = doc.documentSpreadAt(0);

  // Create 2 various objects and add them to the spread
  var o1 = doc.objects().create(ObjectType.Table);
  o1.name = "Table";
  spr.add(o1);

  var o2 = doc.objects().create(ObjectType.Text);
  o2.name = "Text";
  spr.add(o2);

  // Enumerate objects in the spread
  var objs = spr.objects(WhichObjects.All);
  var objList = "";
  for (var idx = 0; idx < objs.length; idx++) {
    var obj = objs.at(idx);
    objList = objList + obj.name + " (" + obj.type + ")\n";
  }

  // Show list of objects in the spread
  messageBox.information("Spread Objects", objList);
}
```

Class: AliasSpread

The class ***AliasSpread*** represents a kind of spread template. The class extends the root class ***Spread*** and adds other properties which are specific for this type of object. It is an area in which an Alias page is located. Each Alias page is located in some Alias spread.

Properties:

name *String* *Read/Write*

Reads or sets the name of the Alias spread.

Methods:

remove() *bool*

Removes the Alias spread (including nested Alias pages) from the appropriate document. Returns true in case of success.

Examples:

```
var doc = application.currentDocument;
if (doc) {

  // Get number of Alias spreads
  messageBox.information("Alias Page Count", doc.aliasPageCount);

  // Add new Alias pages with single resp. facing Alias pages
  var ap1 = doc.addAliasPage("AP1", PageType.SinglePage);
  var ap2 = doc.addAliasPage("AP2", PageType.FacingPages);

  var aList = "";
  for (i = 0; i < doc.aliasPageCount; i++) {
    var apx = doc.aliasPageAt(i);
    if (apx) {
      var ax = apx.spread;
      ax.name = "A" + i;
      aList = aList + "Alias Spread [" + i + "] = " + ax.name + "\n";
    }
  }
  // Show Alias spread list
  messageBox.information("Alias Spreads", aList);
}
```

Class: Layer

Each document has one or possibly more layers. They work like transparent pieces of film. Each visible object on a page and/or on a spread is located in one of the document layers. Objects in a higher layer cover objects in a lower layer. Any layer can be hidden or displayed at any time. A layer can be locked to protect it against changes. Options for a layer can exclude it from printing, etc.

Properties:

name *String* *Read/Write*

Contains the name of the layer. By default the name is "Layer X", where X is an integer number.

hidden *bool* *Read/Write*

The value is true or false if the layer is visible or hidden.

locked *bool* *Read/Write*

The value is true or false if the layer is locked or unlocked.

printable *bool* *Read/Write*

The value is true or false if the layer is printable or not printable.

keepRunaround *bool* *Read/Write*

Get/Set whether the objects of the layer keep their runaround when the layer is hidden.

brush *ColorBrush* *Write*

Sets the color of the layer from a ColorBrush object. The layer color is used for the outlines of all objects (represented in the program interface via the option “Show Guides”). This allows you to distinguish objects on different layers. See also ***colorRgb***.

colorRgb *Rgb* *Read/Write*

Set/get the color of a layer.

Methods:

objects(ObjectType objectType = All) *Objects*

objectType: See the enumeration ObjectType

Returns a list of objects which are assigned to the layer.

objectCount*()* *int*

Returns the number of objects on the layer.

getObject*(int index)* *Object*

Returns the object at the position ***index*** (0-based) or ***null***.

remove() *bool*

Removes the layer from the appropriate document. Returns true in case of success.

orderedObjects() *Objects*

Returns the objects of the layer in their stacking order, as ***Page.objects()*** lists them. ***objects()*** returns them in no particular order.

Examples:

```
var doc = application.currentDocument;
if (doc) {
   // Check the number of layers in the new document
   messageBox.information(“Layer Count”, doc.layerCount);

   // Get the first layer and configure it.
   var lay1 = doc.layerAt(0);
   lay1.name = “Background”;
   lay1.colorRgb = new Rgb(0, 0, 255);
   lay1.printable = false; // Don’t print background layer.

   // Add the 2nd layer and configure it
   var lay2 = doc.addLayer();
   lay2.name = “Layer 1”;
   lay2.colorRgb = new Rgb(255, 0, 0);

   // Make the layer current. Then all objects which are added to current

   // page are placed on this layer.
   doc.currentLayer = lay2;
```

```
    // Work with current layer here
    …

    // Add another layer and move it between 2 existing layers.
    var lay3 = doc.addLayer();
    lay3.name = "Layer 2";
    lay3.colorRgb = new Rgb(0, 160, 0);

    doc.moveLayer(lay2, 2);
}
```

Class: Guide

The methods ***guide()*** of the classes Page and Spread return an object of the class PageGuide or SpreadGuide for a guide ID. Both classes extend the class Guide. The guides are created with ***addHorizontalGuide()*** and ***addVerticalGuide()***, which return the ID.

Properties:

isNull	*bool*	*Read*

Returns true if there is no guide with the ID.

Example:

```
var doc = application.currentDocument;
var page = doc.currentPage;
var id = page.addHorizontalGuide("50mm", doc.currentLayer);
if (id > 0 && !page.guide(id).isNull)
  page.removeGuide(id);
```

Class: PreviewInstructions

An object of the class PreviewInstructions holds the settings for a picture that ***Page.preview()*** saves: scale, alignment, additional margins, page border and background. The method ***preview()*** of the class Object takes it as well.

Constructor:

```
new PreviewInstructions()
new PreviewInstructions(float scale, HorizontalAlignment
horizontalAlignment, VerticalAlignment verticalAlignment, String cropsX,
String cropsY, String cropsWidth, String cropsHeight, bool withFrame, Color
color)
```

Without arguments the object has the default values. The second constructor sets all values at once; ***color*** is a value of the enumeration Color or a ColorBrush object and sets the background (see ***brush***). With ***Color.NoColor*** the other arguments are not taken either.

Properties:

scale	*float*	*Read/Write*

Get/Set the scale in percent: 100 gives 1 pixel per pt (72 ppi), 200 gives 2 pixels per pt. 0 (default) fits the page into the maximum size given to ***preview()***. The maximum size limits the picture in any case.

horizontalAlignment	*HorizontalAlignment*	*Read/Write*

Get/Set the horizontal alignment of the content in the picture. The default is Left.

verticalAlignment *VerticalAlignment* *Read/Write*

Get/Set the vertical alignment of the content in the picture. The default is Top.

cropsX *String* *Read/Write*

Get/Set additional space left and right of the page as UnitValueString. Negative values cut the page. The default is 0. When read, the value is a number in pt.

cropsY *String* *Read/Write*

Get/Set additional space above and below the page.

cropsWidth *String* *Read/Write*

Get/Set additional space on the right only, in addition to ***cropsX***.

cropsHeight *String* *Read/Write*

Get/Set additional space at the bottom only, in addition to ***cropsY***.

withFrame *bool* *Read/Write*

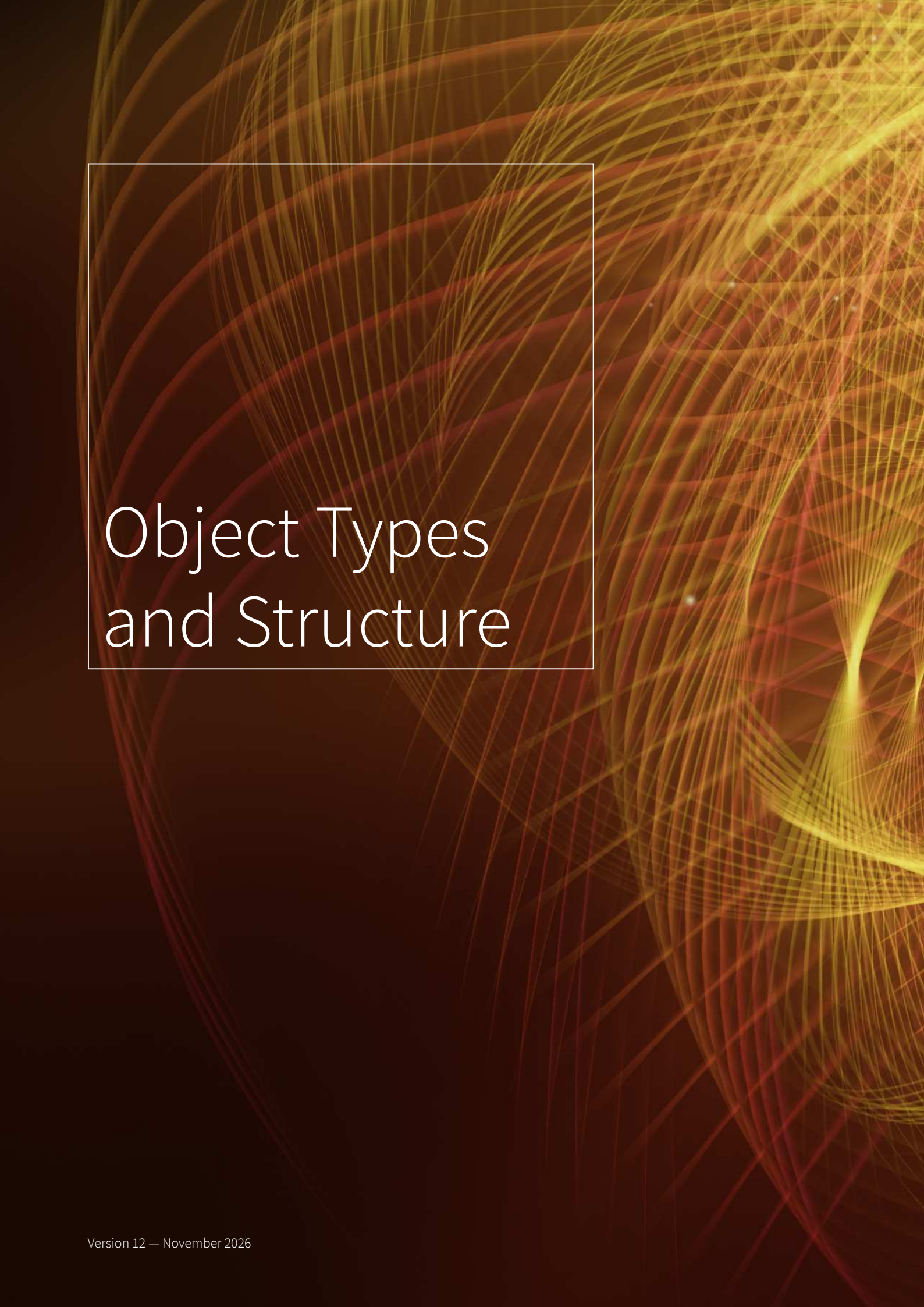
Get/Set whether the border of the page is drawn. The default is true.

brush *ColorBrush* *Write*

Sets a background for the picture instead of the default white. At present the color of the brush is not used: the background becomes black.

Example:

```
var pi = new PreviewInstructions();
pi.scale = 150;
pi.withFrame = false;
pi.cropsX = "5mm"; // 5 mm left and right
pi.cropsY = "5mm"; // 5 mm above and below
var page = application.currentDocument.currentPage;
page.preview("0", "0", "5000", "5000", pi, "C:/Temp/page.png", -1);
```



Object Types and Structure

Version 12 — November 2026

Object Types and Structure

This chapter describes the hierarchy of all graphic object classes that can be used on the page or on the spread in the document.

Document pages can contain objects of the classes *FrameObject*, *LineObject*, *TextObject*, *PictureObject*, *TableObject*, *GroupObject* and *ChartObject* as well as alias objects. The root of all these classes is the class *Object*. The root class cannot be instantiated but shares properties and methods with its subclasses. The complete tree of the page element classes is as follows:

```
Object
|--- GroupObject
   |--- ChartObject
|--- GraphicObject
   |--- LineObject
   |--- FrameObject
      |--- TextObject
      |--- PictureObject
      |--- TableObject
|--- ObjectAlias
   |--- TextObjectAlias
   |--- PictureObjectAlias
```

The classes *Object* and *GraphicObject* cannot be instantiated. They only share their properties and methods with subclasses.

The following chapters describe elements which can be inserted into *DocumentPages*, *Spreads*, *AliasPages* and *AliasSpreads*.

Class: Objects

The class *Objects* represents a collection of objects which are attached to another container object. For example some document pages may contain several text objects, picture objects or table objects. A method objects() of the class *DocumentPage* returns the collection of all objects inside the page. The object of the class *Objects* is also a kind of object factory, which allows you to create new objects (text, pictures etc.). See examples to understand better how to use objects of the class *Objects*.

The object list is current at the time when it is created. The object list must be refreshed when the object is created or deleted in the source object container or the property *isOnline* must be set to *TRUE*.

The collection is indexed from 0 to length-1.

Properties:

length	*int*	*Read*

Contains a number of objects which are members of this object collection.

isOnline	*bool*	*Read/Write*

If the property is set to TRUE, the object list is synchronized with the source object container every time. If the object is deleted or added, the change is reflected to the object list.

The default is *TRUE*.

Methods:

create *(ObjectType type)* *Object*

Creates a new object of the specified type.

type: Type of the created object. See enumeration type Object at the end of this chapter to see all available object types. The ***Alias*** objects cannot be created.

Returns an object of required type.

---

at *(int index)* *Object*

Get object at the specified position inside the object collection.

index: Position of the required object. The value must be in the range from 0 to (length – 1).

Returns the object at the specified position or ***null*** if the object is not found (e.g. index equals or is higher than the length).

---

refresh *()* *void*

The object is synchronized with the object in the source container.

Example 1:

```
var doc = application.currentDocument;
if (doc) {
var objs = doc.objects();
messageBox.information("Object Count", objs.length);
  // Create 1st object
  var obj1 = objs.create(ObjectType.Text);
  obj1.name = "Obj 1";
  doc.currentPage.add(obj1);
  messageBox.information("Object Count", objs.length);
  // Create 2nd object
  var obj2 = objs.create(ObjectType.Picture);
  obj2.name = "Obj 2";
  doc.currentPage.add(obj2);
  messageBox.information("Object Count", objs.length);

  // Get 1st object in the current page
  var objX = doc.currentPage.objects().at(0);
  if( objX != null )
    messageBox.information("1st Object", objX.name);
  else
    messageBox.information("1st Object", "NOT FOUND");

}
```

Example 2:

```
var doc = application.currentDocument;
if (doc) {
  var pg = doc.currentPage;
  var objs = pg.objects(WhichObjects.All);
  if (objs) {
    var textObject = null;
```

```
        var pictureObject = null;

        messageBox.information("Length", objs.length);

        for (var index = 0; index <= objs.length -1 ; index++) {
          var obj = objs.at(index);
          if (obj instanceof TextFrame) {
            obj.content.insertPlainText(index, 0);
            if (!textObject)
              textObject = obj;
          }
          else if (obj instanceof PictureFrame) {
            pictureObject = obj;
          }
        }
      }
    }
```

Class: Object

The class ***Object*** is a root class of page elements and provides the following properties and methods.

Properties:

name	*String*	*Read/Write*

Set or get the name of the object.

type	*ObjectType*	*Read*

Contains the current type of the object. See enumeration items at the end of this chapter.

pagex	*UnitValueString*	*Read/Write*

X-coordinate of the position of the top left corner of the object inside the container page. If the object is not on any page, the value is ***null*** and the setting of a new value is ignored. The value can be set at the top of the object only.

pagey	*UnitValueString*	*Read/Write*

Y-coordinate of position of the top left corner of the object inside the container page. If the object is not on any page, the value is ***null*** and the setting of a new value is ignored. The value can be set at the top of the object only.

spreadx	*UnitValueString*	*Read/Write*

X-coordinate of position of the top left corner of the object inside the container spread. The value can be set at the top of the object only.

spready	*UnitValueString*	*Read/Write*

Y-coordinate of position of the top left corner of the object inside the container spread. The value can be set at the top of the object only.

printable	*bool*	*Read/Write*

Set to true or false to make the object printable or not printable. If the object is not suitable for printing, simply set it to false.

protected *bool* *Read/Write*

If this property is set to true, the object content cannot be modified or deleted.

visible *bool* *Read/Write*

If this property is set to true, the object is visible. Set it to false to hide the object.

locked *bool* *Read/Write*

If this property is set to true, the object size and position cannot be changed.

rotation *float* *Read/Write <-360, 360> [degree]*

Rotation of the object in degrees. A positive value means rotation in an anticlockwise direction. A negative value means rotation in a clockwise direction.

skew *float* *Read/Write <-75, 75> [degree]*

Defines the skewing of the object in degrees. A positive value means skewing the object to the left. A negative value means skewing the object to the right.

mirrored *bool* *Read/Write*

If this property is set to true, the object is mirrored according to the horizontal axis.

selected *bool* *Read/Write*

If this property is set to true, the object is selected.

guide *bool* *Read/Write*

If this property is set to true, the object is set as a (magnetic) guide object.

topLevel *bool* *Read*

If this property is set to true, the object is a top level object.

matrix *Matrix* *Read/Write*

The matrix defines the transformation of the object. Transformation affects the properties: *Rotation*, *Skew*, *Scaling*, *Mirroring* and *Position/Offset (x/y)*. If the object is added to a page or a spread, the matrix parameters are applied to the spread.

note *String* *Read/Write*

Set/get a note for the object.

page *DocumentPage or AliasPage* *Read*

Returns the page object where the object is placed.

spread *Spread or AliasSpread* *Read*

Returns the spread object where the object is placed.

pageBoundingBox *BoundingBox* *Read*

Returns the object bounding box in the page coordinates or *null* if the page doesn't exist.

spreadBoundingBox *BoundingBox* *Read*

Returns the object bounding box in the spread coordinates.

groupObject *GroupObject* *Read*

Returns the group object where the object is placed or *null* if the group doesn't exists.

runaround *Runaround* *Read/Write*

Get/Set the runaround settings of the object. See class *Runaround*.

layer *Layer* *Read/Write*

Returns the layer object where the object is placed.

id *String* *Read*

Returns the identifier of the object in the document, e.g. *"Viva Technology/ObjectID/26c7ad42190"*. Two script objects that stand for the same object return the same identifier.

typeString *String* *Read*

Returns the internal type name of the object: *"Viva Technology/Object/Text"*, *"Viva Technology/Object/Picture"*, *"Viva Technology/Object/Table"*, *"Viva Technology/Object/Frame"* (graphic object), *"Viva Technology/Object/Line"*, *"Viva Technology/Object/Group"* (also for a chart object) or *"Viva Technology/Object/Alias"* (alias objects). Unlike *type*, it also names object kinds without a value in the enumeration type *ObjectType*.

taggedPdfOptions *TaggedPdfOptions* *Read/Write*

Get/Set how the object is tagged in an accessible (tagged) PDF file: automatically, as an artifact (decoration that screen readers skip) or as a figure with an alternative text. Reading gives a copy; change it and assign it back. See the chapter "Class: TaggedPdfOptions".

Methods:

setUserProperty *(String ident, String data)* *void*

Set string data as user property with name ident. The data are 'base64'encoded into the document file to the part '<uni:userInfos> <uni:base64Entry>'

ident: Name of the property.

data: Value set to the property.

userProperty *(String ident)* *String*

Get value from the named property. The value is set to the property by an earlier call of the method *setUserProperty(ident, data)*.

ident: Name of the property.

Returns textual value of the property. If the property doesn't exist, the returned value is an empty string.

copyUserProperties *(Object sourceObject)* *void*

The user property data are copied from the source object.

userXMLProperty *(String ownerId, String entryId)* *String*

Get user property string from the document file from the part '<uni:userInfos> <uni:xmlEntry>'

setUserXMLProperty *(String ownerId, String entryId, String data)* *void*

Set string data as user property to the document file to the part '<uni:userInfos> <uni:xmlEntry>'

containsUserProperty*(String ident)bool*

Checks if the object contains a specified user property.

ident: Name of the checked property.

Returns true if the user property exists, otherwise it returns false.

createAlias() *ObjectAlias*

Creates an alias object of the object and returns it. Its property ***owner*** returns this object. See class *ObjectAlias*.

fullCopy() *Object*

Copies the object and add it to the spread with right position on the page. Returns new object.

copy() *Object*

Copies the object. Returns new object only.

customProperty *(String ident)* *String*

Get value of the string type of the custom property. Name of the feature in the VivaDesigner is 'User Defined Properties'. Returns textual value of the property.

ident: Name of the property.

customPropertyBoolean *(String ident)* *bool*

Get value of the checkbox type of the custom property. The name of the feature in the VivaDesigner is 'User Defined Properties'. Returns boolean value (TRUE/FALSE) of the property.

ident: Name of the property.

setCustomProperty *(String ident, String data)* *void*

setCustomProperty *(String ident, bool data)* *void*

Set the value of the 'User Defined Properties' (String or boolean). Only properties that are defined in the document can be set; a property with an unknown name is not created.

ident: Name of the property.

data: New value of the property.

changeOrder *(ChangeOrderType type)* *bool*

Changes the order of object on the page.

moveOnPageTo *(Point p, MoveReferencePoint ref = BBoxTopLeft, Page pg = null)* *bool*

moveOnPageTo *(String x, String y, MoveReferencePoint ref = BBoxTopLeft, Page pg = null)* *bool*

Moves the object to the position ***p*** (or ***x***, ***y***) in the coordinates of the page. Returns true in case of success.

x, ***y***: Position as *UnitValueString*, e.g. *"20mm"*; a number without a unit is in points.

ref: The point of the object that is placed at the position: *MoveReferencePoint.BBoxTopLeft* (default, the top left corner of the bounding box) or *MoveReferencePoint.OriginTopLeft* (the top left corner of the object itself).

pg: The page whose coordinates are used; default the page of the object.

moveOnSpreadTo *(Point p, MoveReferencePoint ref = BBoxTopLeft)* *bool*

moveOnSpreadTo *(String x, String y, MoveReferencePoint ref = BBoxTopLeft)* *bool*

Moves the object to the position *p* (or *x*, *y*) in the coordinates of the spread. The parameters are the same as for *moveOnPageTo()*.

move *(Point point)* *bool*

Moves the top left corner of the bounding box of the object to the position *point* in the coordinates of the spread. Returns true in case of success.

getAttributes *()* *Attributes*

Returns all attributes of the object as an *Attributes* object. See class *Attributes*.

setAttributes *(Attributes attrs)* *void*

Applies the attributes *attrs* to the object.

copyAttributes *(Object fromObj)* *boolean*

Copies all attributes of the object *fromObj* to this object. Returns true in case of success.

preview *(String width, String height, String fileName, int quality) bool*

preview *(String x, String y, String width, String height, PreviewInstructions pi, String fileName, int quality) bool*

Saves a preview image of the object to the file *fileName*. The file extension sets the format, e.g. *.png* or *.jpg*. Returns true in case of success.

width, *height*: Maximum size of the image as *UnitValueString*; one point gives one pixel. By default the object is scaled proportionally to fit into this size.

x, *y*: Must be valid measures, e.g. *"0"*; only *width* and *height* limit the image.

pi: Scale, alignment, margins and background of the image, or *null* for the defaults. See the chapter “Class: PreviewInstructions”.

quality: 0 .. 100 for formats with compression (e.g. JPEG), -1 for the default.

Example:

```
  var doc = application.currentDocument;
  var obj = doc.currentPage.objects().at(0);
  // Preview image of at most 300 x 200 pixels
  obj.preview("300pt", "200pt", "C:/Temp/object.png", -1);
  // Tag the object as a figure for accessible PDF files
  obj.taggedPdfOptions = new TaggedPdfOptions(TaggedPdfType.Figure, "Logo
of the company");
  // Move the top left corner of its bounding box to 20 mm, 30 mm
  obj.moveOnPageTo("20mm", "30mm");
```

Class: GraphicObject

The class *GraphicObject* extends the root class *Object* and adds other properties that are specific for this type of object. This class also cannot be instantiated.

Properties:

dropShadow *Shadow* *Read/Write*

Defines the parameters of the object's drop shadow. See class ***Shadow*** in this documentation to see details of the shadow parameters.

innerShadow *Shadow* *Read/Write*

Defines the parameters of the object's inner shadow. See class ***Shadow*** in this documentation to see details of the shadow parameters.

innerGlow *InnerGlow* *Read/Write*

Defines the parameters of the object's inner glow. See class ***InnerGlow*** in this documentation to see details of the glow parameters.

outerGlow *OuterGlow* *Read/Write*

Defines the parameters of the object's outer glow. See class ***OuterGlow*** in this documentation to see details of the glow parameters.

colorOverlay *ColorOverlay* *Read/Write*

Defines the parameters of the object's color overlay. See class ***ColorOverlay***.

reflection *Reflection* *Read/Write*

Defines the parameters of the object's reflection. See class ***Reflection***.

transparency *Transparency* *Read/Write*

Defines the transparency of the object. See class ***Transparency***.

styleSheet *GraphicStyleSheet* *Read/Write*

styleSheetByName *String* *Read/Write*

Get/Set the style sheet which is applied to define the appearance of the graphic object. The style sheet can be defined by name or by an object ***GraphicStyleSheet***.

stroke *Object* *Read/Write*

Get/Set the layer effect Stroke (Object > Effects..., page Stroke): a line along the outline of the object. For a picture that is cut out (background mask, alpha channel or clipping path) the line runs along the cut-out, not along the outline of the picture object. The value is a plain object with the keys below. Reading gives a copy: change it and assign it back. Keys left out keep their value; numbers out of range are limited to the range. The keys:

active: bool, default false. Switches the effect on or off.

color: A color of the color list: ***"[Black]"*** (default), ***"[White]"*** or the name of a color of the document.

shade: Shade of the color in percent, 0 .. 100, default 100.

opacity: Opacity in percent, 0 .. 100, default 100.

width: Width in points, 0 .. 144, default 2.

position: ***"outside"*** (default, around the object), ***"center"*** (half outside, half inside) or ***"inside"*** (over the edge of the object).

mode: Blend mode, default *"normal"*. The blend modes of all layer effects are *"normal", "multiply", "screen", "overlay", "darken", "lighten", "colorDodge", "colorBurn", "hardLight", "softLight", "difference", "exclusion", "hue", "saturation", "color", "luminosity"*.

---

gradientOverlay *Object* *Read/Write*

Get/Set the layer effect Gradient Overlay (Object > Effects..., page Gradient Overlay): a color blend or a color of the color list laid over the object, like *colorOverlay* with a gradient. Reading and assigning work as for *stroke*. The keys:

active: bool, default false.

gradient: The name of a color blend or a color of the color list, default *"[Black]"*.

angle: Direction of the gradient in degrees, -360 .. 360, default 90.

opacity: Opacity in percent, 0 .. 100, default 100.

mode: Blend mode (see *stroke*), default *"normal"*. *"multiply"* darkens towards the gradient, *"screen"* lightens, *"color"* tints.

---

bevelEmboss *Object* *Read/Write*

Get/Set the layer effect Bevel and Emboss (Object > Effects..., page Bevel and Emboss): the object looks raised from its outline, lit from one side. Reading and assigning work as for *stroke*. The keys:

active: bool, default false.

style: *"innerBevel"* (default, the object rises from its edge inwards) or *"emboss"* (the edge rises inside and outside).

depth: Steepness of the slope in percent, 1 .. 1000, default 100.

size: Width of the slope in points, 0.5 .. 144, default 5.

soften: Smoothing of the slope in points, 0 .. 72, default 0.

angle: Direction the light comes from in degrees, counterclockwise from the right (90 = from above), -180 .. 180, default 120.

altitude: Height of the light above the surface in degrees, 0 .. 90, default 30.

highlightColor, *highlightOpacity*, *highlightMode*: Color, opacity (percent) and blend mode of the lit side, defaults *"[White]"*, 75 and *"screen"*.

shadowColor, *shadowOpacity*, *shadowMode*: Color, opacity (percent) and blend mode of the shaded side, defaults *"[Black]"*, 75 and *"multiply"*.

Example:

```
var obj = application.currentDocument.currentPage.objects().at(0);
// Read a copy, change it, assign it back
var s = obj.stroke;
s.active = true;
s.width = 3;
s.position = "center";
obj.stroke = s;
// Keys left out keep their value
obj.bevelEmboss = { active: true, size: 8, angle: 90 };
```

Class: LineObject

This class defines the features of a line which an author can place on any page. The ***LineObject*** is derived from the classes ***GraphicObject*** and ***Object***. It means that this kind of object has the same properties and methods as the superclasses and adds line specific properties and methods.

The line can be a simple line, or it can have line ends such as arrows or bullets. It can be a straight line or a curved line, etc. See following properties to see what it is possible to do with this line object.

Properties:

form	*LineForm*	*Read/Write*

This property defines the type of the line. See enumeration type ***LineForm*** at the bottom of this chapter.

lineLength	*UnitValueString*	*Read/Write*

Length of the line in points (pt) or millimeters (mm). This property is ignored if the form is ***LineForm PolyLine***. A polyline (multiple line) is then defined by the property ***geometryPoints***.

lineAngle	*float*	*Read/Write*

Angle of the line in degrees. For a constrained line, the value should be a multiple of 45. This property is ignored if the form is ***LineForm.PolyLine***. A polyline (multiple line) is then defined by the property *geometryPoints*.

lineColor	*ColorBrush*	*Read/Write*

The line color is defined by ***ColorBrush***. See the chapter “Class: ColorBrush” in this documentation.

lineShade	*float*	*Read/Write*

Defines the shade of the line color as a percentage. The maximum value is 100.0 (100%).The value 0 means that the line is white, but not transparent. The value -1 means that the line is fully transparent.

lineOpacity	*float*	*Read/Write*

Defines the line opacity. The maximum value 100.0 (i.e. 100%) means that the line has no translucence/ opacity. The value 0 means that the line is fully transparent.

lineWidth	*UnitValueString*	*Read/Write*

Defines line width.

lineStyle	*String*	*Read/Write*

Defines the style of the line. Available values are: “Solid” (solid line), “Pattern 1” (dashed), “Pattern 2” (dashed with shorter gaps), “Pattern 3” (dashdot line), “Pattern 4” (dotted, squared dots), “Pattern 5” (dashed with rounded ends of dashes), “Pattern 6” (dashed with shorter gaps and rounded ends of dashes), “Pattern 7” (dashdot line rounded ends of dashes), “Pattern 8” (dotted, circular dots), “Pattern 9” (dashed with short dashes), “Pattern 10” (dashdot line with short dashes), etc. ... up to “Pattern 19” as combination of longer or shorter dashes, circular or squared dots.

pageGeometryPoints	*PointList*	*Read/Write*

This property is applied if the form property is ***LineForm.PolyLine***. It is used to place the end points of polyline (multiple line) segments. See the chapter Point and PointList to see how to work with the points.

spreadGeometryPoints	*PointList*	*Read/Write*

This property is applied if the form property is ***LineForm.PolyLine***. It is used to place the end points of poly-line (multiple line) segments. See the chapter Point and PointList to see how to work with the points.

lineStart *LineCap* *Read/Write*

The start of the line may have a normal end or can be terminated by an arrowhead, an arrow end/tail or a bullet point. See the available values of the enumeration type ***LineCap***.

lineEnd *LineCap* *Read/Write*

The end of the line may have a normal end or can be terminated by an arrowhead, an arrow end/tail or a bullet point. See the available values of the enumeration type ***LineCap***.

Methods:

remove() *bool*

Removes the object from the appropriate page and destroys it. Returns true in case of success.

Examples:

```
var doc = application.currentDocument;
if (doc) {
  // Create simple line
  var line1 = doc.objects().create(ObjectType.Line);
  if (line1) {
    with(line1) {
      name = "Line 1";
      // Normal simple line
      form = LineForm.FreeLine;
      pagex = "20mm";
      pagey = "200.5pt";
      lineLength = "170pt";
      lineAngle = 30;
      lineWidth = "10pt";
      // Dashed line
      lineStyle = "Pattern 2";
      // Find (create if not found) a blue brush
      var brushBlue = doc.brushes.find("Blue");
      if( brushBlue == null ) {
        brushBlue = doc.brushes.addColor("Blue");
        brushBlue.rgb = new Rgb(0, 0, 255);
      }
      // Set the line color
      lineColor = brushBlue;
      // Set opacity of the line to be by 50% transparent
      lineOpacity = 50.0;
    }
    // Add the line to a current page
    doc.currentPage.add(line1);
  }
  // Create poly line
  var line2 = doc.objects().create(ObjectType.Line);
  if (line2) {
    with(line2) {
```

```
        name = "Line 2";

        // Poly-line
        form = LineForm.PolyLine;
        // Assign array of points
        // a) simple points => the poly-line consists of 3 straight
        // lines
        pageGeometryPoints = new PointList([[193.58, 360.119], [192.638,
247.249],
           [280.689, 350.379], [310.689, 99.379]]);

        // b) 2nd point is created by control and sizing points =>
        // the final poly-line consists of curved lines
        pageGeometryPoints = new PointList([[193.58, 360.119],
           [150.600, 200.00, "B"],[196.638, 255.249],
           [210.600, 400.00, "A"], [280.689, 350.379],
           [310.689, 99.379]]);

        // Dotted line, 3mm wide
        lineStyle = "Pattern 8";
        lineWidth = "3mm";
     }
     // Add the line to a current page
     doc.currentPage.add(line2);
   }
}
```

Class: FrameObject

The class ***FrameObject*** is another subclass of ***GraphicObject***. The basic feature of the frame object is that it can have a rectangular, oval or polygonal shape, can be framed and filled by a color or brush.

Optionally the frame object can be filled by a text or picture. If the frame object is rectangular, the content may also be a table.

The class ***FrameObject*** is a superclass for ***TextObject***, ***PictureObject*** and ***TableObject***.

Properties:

width	*UnitValueString*	*Read/Write*
Defines the object width.		
height	*UnitValueString*	*Read/Write*
Defines the object height.		
form	*FrameForm*	*Read/Write*
Defines the shape of the object. The value is one of the items of the enumeration type ***FrameForm***. A description of this can be found at the end of this chapter.		
pageGeometryPoints	*PointList*	*Read/Write*
The property is taken into consideration if the form property is ***FrameForm.Polygon***. It is used to place the end points of polygon segments.		

spreadGeometryPoints *PointList* *Read/Write*

The property is taken into consideration if the form property is ***FrameForm.Polygon***. It is used to place the end points of polygon segments.

frameWidth *UnitValueString* *Read/Write*

Defines the frame width (stroke).

frameStyle *String* *Read/Write*

Defines the line style of the frame. Available values are: “Solid” (solid line), “Pattern 1” (dashed), “Pattern 2” _ (dashed with shorter gaps), “Pattern 3” (dashdot line), “Pattern 4” (dotted, squared dots), “Pattern 5” (dashed with rounded ends of dashes), “Pattern 6” (dashed with shorter gaps and rounded ends of dashes), “Pattern 7” (dashdot line rounded ends of dashes), “Pattern 8” (dotted, circular dots), “Pattern 9” (dashed with short dashes), “Pattern 10” (dashdot line with short dashes), etc. ... up to “Pattern 19” as combination of longer or shorter dashes, circular or squared dots.

frameColor *ColorBrush* *Read/Write*

The line/frame color is defined by the term ***ColorBrush***. See the chapter “Class: ColorBrush” in this documentation.

frameShade *float* *Read/Write*

Defines the line/frame color shade as a percentage. The maximum value is 100.0 (100%).The value 0 means that the line is white, but not transparent. The value -1 means that the line/frame is fully transparent.

frameOpacity *float* *Read/Write*

Defines the opacity of the line/frame. The value 100.0 (i.e. 100%) means that the frame has no translucence/opacity. The value 0 means that the frame is fully transparent.

frameRadius *UnitValueString* *Read/Write*

Defines the radius of this rounding. A rectangular frame object can have rounded corners.

fillColor *Brush* *Read/Write*

The frame object is filled by using the term ***Brush***. Examples are shown below.

fillShade *float* *Read/Write*

Defines the shade of the fill color. The maximum value is 100.0 (i.e. 100%).The value 0 means that the fill is white, but not transparent. The value -1 means that the fill is fully transparent.

fillOpacity *float* *Read/Write*

Defines the opacity of the fill color. The value 100.0 (i.e. 100%) means that the fill has no translucence/opacity. The value 0 means that the fill is fully transparent.

fillBlendAngle *float* *Read/Write*

Defines the angle of the color blend, if the frame is filled by a color blend instead of a solid color.

objectOpacity *float* *Read/Write*

Defines the overall opacity of the frame object. The value 100.0 (i.e. 100%) means that the object (fill and frame) has no translucence/opacity. The value 0 means that the object (fill and frame) is fully transparent.

Methods:

remove() *bool*

Removes the object from the appropriate page and destroys it. Returns true in case of success.

Examples:

```
var doc = application.currentDocument;
if (doc) {
  // Create simple frame object
  var frmo = doc.objects().create(ObjectType.Graphic);
  if (frmo) {
    with(frmo) {
      name = "Frame Object 1";
      // Rectangular frame object
      form = FrameForm.Rectangle;
      // position and size of the frame object
      pagex = "50.15pt"; pagey = "201.65pt"; width = "200pt"; height =
"300pt";

      // Rounded corners (2.0 mm)
      frameRadius = "2mm";

      // Find (create if not found) a blue brush for border
      var brushBlue = doc.brushes.find("Blue");
      if( brushBlue == null ) {
        brushBlue = doc.brushes.addColor("Blue");
        brushBlue.rgb = new Rgb(0, 0, 255);
      }
      // Set style of the border: color, line width, pattern
      frameColor = brushBlue;
      frameWidth = "3pt";
      frameStyle = "Pattern 4";

      // Find (create if not found) a red brush for interior
      var brushRed = doc.brushes.find("Red");
      if( brushRed == null ) {
        brushRed = doc.brushes.addColor("Red");
        brushRed.rgb = new Rgb(255, 0, 0);
      }

      // Set color of the frame interior
      fillColor = brushRed;
      fillShade = 100.0;
      // Set overall opacity of the frame object to 50%
      objectOpacity = 50.0;
    }
    // Add the frame object to a current page
    doc.currentPage.add(frmo);
  }
}
```

Class: TextObject

The TextObject represents an area that is filled by formatted text content. The formatted text can contain not only text, but also pictures and tables. The content of the text object is accessible via its property content. The position, size, shape, border, background and other features of the text object are defined by the values of properties which are derived from the class FrameObject. See the description of the FrameObject in the chapter “Class: FrameObject”. The position is set by the properties pagex and pagey (class Object), the size by width and height (class FrameObject).

Properties:

`content` *TextContent* *Read*

Defines the entry point of the content of the text object. See examples in this chapter to see how to use and manipulate the object content.

`individualAliasContent` *bool* *Read/Write*

Get/Set whether each alias object of this text object can have its own content. The option takes effect on all alias objects of this original object.

`contentOpacity` *float* *Read/Write*

Set/Get the opacity of the text content in percent.

`textChain` *TextChain* *Read*

Returns the text chain the text object belongs to. See class ***TextChain***.

Methods:

`connectTo` *(TextObject object)* *bool*

Links the text object with ***object***: the text flows on from this text object into ***object*** (text chain). Returns true in case of success.

`remove()` *bool*

Removes the object from the appropriate page and destroys it. Returns true in case of success.

Examples:

```
var doc = application.currentDocument;
if (doc) {
   // Create new text object
   var objT = doc.objects().create(ObjectType.Text);
   if (objT) {
      with (objT) {
         // Configure the shape to be an ellipse
         form = FrameForm.Ellipse;

         // Set the position (x,y) and the width and height
         pagex = “25mm”;
         pagey = “45.5pt”;
         width = “150mm”;
         height = “120mm”;

   // Create red border, yellow interior and set opacity
         frameWidth = "1.5mm";
```

```
      frameColor = doc.brushes.find("[Red]");
      fillColor = doc.brushes.find("[Yellow]");
      fillShade = 50;
      fillOpacity = 70;
    }
  }
  // Insert the text object on a current page
  doc.currentPage.add(objT);

  // Access the content of the text object
  with (objT.content) {
    // The text content will be arranged in 2 columns
    columnCount = 2;
    columnOffset = "10mm";
    // Set content position inside the text object
    leftIndent = "5mm";
    topIndent = "25mm";
    rightIndent = "5mm";
    bottomIndent = "35mm";

    // Add plain text to the content object
    insertPlainText("AABBBCCC", 10);

    // Get the object which processes text formatting
    // inside the text object content.
    var ft = formattedText;
    // Insert new text at the 1st position before
    // previously inserted plain text.
    ft.insertPlainText("This is a sample formatted text! ", 0);
    ft.setFontSize(new FontSize(20), 2, 5);

    // Create a picture object
    var objPicture = doc.objects().create(ObjectType.Picture);
    // Configure the size and content
    objPicture.width = "20mm";
    objPicture.height = "40mm";
    objPicture.content.importPicture("anypicture.jpg", 1);
    // Insert the picture into the formatted text
      if(objPicture)
        ft.setObject(objPicture, 3);

    // Create a table object and insert it also to the text
    var objTable = doc.objects().create(ObjectType.Table);
    if(objTable)
      ft.setObject(objTable, 10);
  }
}
```

Class: TextContent

The class ***TextContent*** describes the content of a text object (see class ***TextObject***). The text object consists of a frame (closed) object (see class ***FrameObject***) and text content. The frame object implements the area for the content and can be a rectangle/square, ellipse/circle or polygon/bézier object, can be filled with any fill color or blend and can be surrounded by any border/ frame. The text content of the text object is implemented by the class ***TextContent***.

Properties:

columnCount	*int*	*Read/Write*

Defines the number of columns in a text object. The default value is 1.

columnOffset	*UnitValueString*	*Read/Write*

Defines the gap or gutter between columns.

leftIndent	*UnitValueString*	*Read/Write*
topIndent	*UnitValueString*	*Read/Write*
rightIndent	*UnitValueString*	*Read/Write*
bottomIndent	*UnitValueString*	*Read/Write*

Define left, top, right and bottom distance between the text and a “virtual” rectangle surrounding whole frame of the text object.

columnLines	*ColumnLines*	*Read/Write*

Get/Set the vertical lines between the columns. See class ***ColumnLines***. Reading gives a copy; change it and assign it back.

textRange	*TextRange*	*Read*

Use this method to get the parameters of the text inside the appropriate text object. See class ***TextRange*** to see all available parameters for the text.

formattedText	*FormattedText*	*Read*

Get an object which allows you to read and manipulate the formatted text inside the text object. See class ***FormattedText*** to see all text properties and methods.

baselinePosition	*BaselinePosition*	*Read/Write*

Set/get the ***BaselinePosition*** object. See Class: BaselinePosition.

baselineDescenders	*bool*	*Read/Write*

Descenders are the part of a character that hang down below the baseline, such as “p” or “y”. When this function is activated the last line of the text may now fit in the object when it did not before.

linkableWithOriginal	*bool*	*Read/Write*

Get/Set the option “Linkable with Original”: the text object is linked automatically with the text chain of the previous page.

baselineGrid	*BaselineGrid*	*Read/Write*

Get/Set the baseline grid of the text object. See class ***BaselineGrid***.

header	*HeaderFooter*	*Read/Write*
footer	*HeaderFooter*	*Read/Write*

Get/Set the header and the footer area of the text object: whether it is active and its height. See class ***HeaderFooter***. Reading gives a copy; change it and assign it back.

footnotes	*GraphicFootnotes*	*Read/Write*

Get/Set the position and the columns of the footnotes in the text object. See class ***GraphicFootnotes***.

headerFormattedText	*FormattedText*	*Read*
footerFormattedText	*FormattedText*	*Read*

Return the text of the header and of the footer area as ***FormattedText***, or ***null*** if the area is not active.

root	*bool*	*Read*

Returns true if the text object is the first object of its text chain or is not linked.

Methods:

plainText *(TextRange textRange)* *String*

Get simple plain text from the text object. The text doesn't contain any formatting parameters or any object that was inserted in the text.

textRange: The parameter is used to define the beginning and end of the requested text.

Returns the required text as a String.

plainText *(int begin, int end)* *String*

Get simple plain text from the text object. The text doesn't contain any formatting parameters or any object that was inserted inside the text.

Begin: Beginning of the requested text. Value is 0-based, i.e. first letter has index 0.

end: Position behind the last character of requested text.

Returns the requested text as a String.

insertPlainText *(String text, int index = 0)* *void*

Insert plain text to the requested position in the existing text.

text: Inserted text.

index: Index of the character before which the inserted text is placed.

getAttributes *()*	*Attributes*
setAttributes *(Attributes attrs)*	*void*
copyAttributes *(Object fromObj)*	*boolean*

Read, set or copy all attributes of the text content as an ***Attributes*** object. See class ***Attributes***.

Examples:

```
var doc = application.currentDocument;
if (doc) {
  // Create new text object
  var objT = doc.objects().create(ObjectType.Text);
```

```
  if (objT) {
    with (objT) {
      // Configure its shape as a rectangle
      form = FrameForm.Rectangle;
      pagex = "15mm"; pagey = "35.5pt"; width = "180mm"; height = "120mm";
      // Create red border, yellow interior and set opacity
      frameWidth = "1mm";
      frameColor = doc.brushes.find("[Red]");
    }
  }
  // Insert the text object on a current page
  doc.currentPage.add(objT);

  // Access the content of the text object
  with (objT.content) {
    // Set content position inside the text object
    leftIndent = "5mm";
    topIndent = "25mm";
    rightIndent = "5mm";
    bottomIndent = "15mm";

    // The text content will be arranged as 3 columns with a gutter of 8
mm
    columnCount = 3;
    columnOffset = "8mm";
    // Configure the vertical line separators between the columns
    var cl = new ColumnLines();
    cl.lineColor = doc.brushes.find("[Green]");
    cl.lineShade = 50;
    cl.lineWidth = "1.5mm";
    cl.topIndent = "4mm";
    cl.bottomIndent = "4mm";
    cl.topAnchor = TextColumnLineAnchor.Baseline;
    cl.bottomAnchor = TextColumnLineAnchor.ObjectFrame;
    cl.lineStyle = "Pattern 8";
    cl.active = true;
    columnLines = cl;

    // Add plain text to the content object
    insertPlainText("This is plain text.", 10);

    // Get object which processes text formatting
    // inside the text object content.
    var ft = formattedText;
    // Insert a new text at the 1st position before
    // previously inserted plain text.
    ft.insertPlainText("This is formatted text.\n\n", 0);
    ft.setFontSize(new FontSize(20), 8, 22);
    // Get plain text from the text object
    var txt = plainText(8, 22);
    messageBox.information("Text", txt);
  }
```

```
}
```

Class: PictureObject

The ***PictureObject*** represents an object which is filled by a picture. The picture itself is accessible via its property content. The position, size, shape, border, background and other features of the picture object are defined by values of properties that are derived from the class ***FrameObject***. See the description of the FrameObject in the chapter “Class: FrameObject”. The position is set by the properties pagex and pagey (class Object), the size by width and height (class FrameObject).

Properties:

content	*PictureContent*	*Read*

This property is the entry point for image properties. See examples in this chapter to see how to use and manipulate the image. Also see the chapter “Class: PictureContent” for detailed information about all picture properties.

individualAliasContent	*bool*	*Read/Write*

Get/Set whether each alias object of this picture object can have its own content. The option takes effect on all alias objects of this original object.

contentOpacity	*float*	*Read/Write*

Set/Get the opacity of the imported picture in percent

Methods:

remove()	*bool*

Removes the object from the appropriate page and destroys it. Returns true in case of success.

Examples:

```
var doc = application.currentDocument;
if (doc) {
  var objPic = doc.objects().create(ObjectType.Picture);
  if (objPic) {
    with (objPic) {
      // Set rectangular shape of the picture object
      form = FrameForm.Rectangle;
      // Set the position (x,y) and the width and height
      pagex = "35mm"; pagey = "220.2pt"; width = "130mm"; height = "85mm";
      // Rotate the object by 15 degrees
      rotation = 15.0;
      contentOpacity = 80;

      // Import a picture to the picture object
      content.importPicture("c:\\picture.pdf");
      // Read or manipulate the picture using property "content"
      with (content) {
        rotation = 10;
        skew = 0;
        verticalScale = 100;
        horizontalScale = 150;
        verticalMirrored = false;
```

```
          horizontalMirrored = false;
          // set the picture coordinates
          horizontalOffset = "10mm";
          verticalOffset = "25.5pt";
          // following properties are for reading only
          filePath;
          resolutionX;
          resolutionY;
          width;
          height;
        }
      }
      doc.currentPage.add(objPic);
    }
}
```

Class: PictureContent

The class ***PictureContent*** describes the content of the picture object (see the class ***PictureObject***).

Properties:

filePath	*String*	*Read*
Returns the full path of the imported image to the object or ***null*** if picture doesn't exist.		
horizontalOffset	*UnitValueString*	*Read/Write*
Set/get the X position of the top-left corner of the imported image.		
verticalOffset	*UnitValueString*	*Read/Write*
Set/get the Y position of the top-left corner of the imported image.		
width	*float*	*Read*
Get the width of the imported image in points [pt].		
height	*float*	*Read*
Get the height of the imported image in points [pt].		
resolutionX	*float*	*Read*
Get the X resolution of the imported image in ppi (pixels per inch).		
resolutionY	*float*	*Read*
Get the Y resolution of the imported image in ppi (pixels per inch).		
verticalScale	*float*	*Read/Write*
Set/get the vertical scale of the imported image in percent. [%]		
horizontalScale	*float*	*Read/Write*
Set/get the horizontal scale of the imported image in percent. [%]		
rotation	*float*	*Read/Write*
Set/get the rotation of the imported image in degree at interval (-360,360).		

skew *float* *Read/Write*

Set/get the skew of the imported image in degree at interval (-75,75).

verticalMirrored *bool* *Read/Write*

Set/get a flag if the imported image is mirrored vertically.

horizontalMirrored *bool* *Read/Write*

Set/get a flag if the imported image is mirrored horizontally.

clippingEnabled *bool* *Read/Write*

Set/get a flag if the imported image can be clipped if clipping for the imported picture is activated.

clippingTight *bool* *Read*

Returns the option *tight* of the clipping. See *setClipping()*.

clippingTolerance *int* *Read*

Returns the clipping tolerance of the imported image, 0 .. 255. See *setClipping()*.

colorMode *ColorMode* *Read/Write* *value limits: enumeration type*

Set/get the color mode.

colorSpace *ColorSpace* *Read*

Get the color space.

halftoneAngle *float* *Read/Write* *unit: degree* *value limits: <-360, 360>*

Set/get the halftone angle.

halftoneScreen *float* *Read/Write* *unit: lpi* *value limits: <0,20000>*

Set/get the halftone screen.

styleSheet *PictureStyleSheet* *Read/Write*

styleSheetByName *String* *Read/Write*

Set/get the style sheet which is applied to define the appearance of the picture object. The style sheet can be defined by name or by an object *PictureStyleSheet*.

isEmpty *bool* *Read*

Returns true if the picture object holds no picture.

isEmbedded *bool* *Read*

Returns true if the picture is embedded in the document instead of linked to a file.

layerOverrides *bool* *Read/Write*

Get/Set the option “Keep Layer Selection When Updating” of Picture > Image Layers: the layers of the picture shown or hidden in VivaDesigner are kept when the picture is updated.

adjustments *Object* *Read/Write*

Get/Set the color and tone corrections of the picture (Picture > Image Adjustments...): warmer or cooler, brighter or darker, contrast, saturation, hue, levels, tone curves, grayscale or sepia, sharpen, blur and the automatic corrections. They are applied when the picture is drawn (on the screen and in the output); the image file is not changed. The value is a plain object. Reading gives a copy: change it and assign it back. When assigning, a key left out takes its default (no correction), so `{}` removes every correction. Values out of range are limited to the range.

Raster images, PDF, AI, EPS and SVG pictures can carry corrections; vector pictures are rasterized for the output while a correction is set. Assigning a correction to an empty picture object throws an error; see the key `supported`. The keys:

`autoTone`: bool, default false. Automatic tone: black and white point per channel, removes color casts.

`autoContrast`: bool, default false. Automatic contrast: one black and white point for all channels, keeps the colors.

`autoColor`: bool, default false. Automatic color: the darkest and lightest colors become black and white, the midtones neutral.

`autoWhiteBalance`: bool, default false. Automatic white balance: the average color becomes gray.

`exposure`: Exposure in EV, -3 (darker) .. 3 (brighter), default 0.

`brightness`: Brightness of the midtones, -100 (darker) .. 100 (brighter), default 0.

`contrast`: -100 (flat) .. 100 (more contrast), default 0.

`saturation`: -100 (gray) .. 100 (more vivid colors), default 0.

`temperature`: Color temperature, -100 (cooler, blue) .. 100 (warmer, yellow), default 0.

`hue`: Hue rotation in degrees, -180 .. 180, default 0.

`levelsInBlack`, `levelsInWhite`: Levels for all channels: input black and white point, 0 .. 1, defaults 0 and 1. Tones at or below the black point become the output black, tones at or above the white point the output white. The levels are applied after brightness and contrast, before the tone curve.

`levelsGamma`: Levels: midtone gamma, 0.1 .. 10, default 1. Values above 1 brighten the midtones, values below 1 darken them.

`levelsOutBlack`, `levelsOutWhite`: Levels: output black and white point, 0 .. 1, defaults 0 and 1. A higher black point gives lighter, flatter shadows, a lower white point darker, flatter highlights.

`curve`: Tone curve for all channels: an array of control points `[x, y]` that map the input `x` to the output `y`, both 0 .. 1, sorted by `x`, at most 32 points. A smooth curve runs through them. Empty (default) is a straight line; malformed points are ignored. An S curve for more contrast: `[[0, 0], [0.25, 0.2], [0.75, 0.8], [1, 1]]`.

`curveRed`, `curveGreen`, `curveBlue`: Tone curve of one color channel, applied after `curve`, in the same form. Raising the red curve makes the picture redder, lowering it more cyan (green: magenta, blue: yellow). Grayscale images ignore them.

`maskInvert`: bool, default false. Inverts the mask of the picture (its cut-out: the background mask, else the alpha channel of the image file, together with its clipping path): what was cut away is shown, the rest is hidden.

`maskExpand`: Grows (positive) or shrinks (negative) the cut-out, in points of the picture, -200 .. 200, default 0. Shrinking by 1 or 2 removes a fringe of background color.

maskFeather: Soft edge of the cut-out in points of the picture, 0 (hard, default) .. 200.

maskDensity: How much the mask hides, 0 .. 1, default 1; 0.5 lets the hidden part show through at half strength.

retouchLayer: bool. An AI retouch of the image editing (elements removed or replaced, text removed) lies with its own mask over the unchanged picture. Reading gives true if the picture has such a retouch layer (without one the key is missing). Assigning false removes it, so that the original shows again; true changes nothing.

retouchFeather: Soft edge of the retouched area in points of the picture, 0 (default) .. 200.

retouchStrength: Strength of the retouch layer, 0 .. 1, default 1; 0.5 lets the original show through at half strength, 0 hides the retouch.

colorMode: *"color"* (default), *"grayscale"* or *"sepia"*. Not the same as the property *colorMode* of this class.

sharpenAmount: Sharpening (unsharp mask) in percent, 0 (off, default) .. 500.

sharpenRadius: Radius of the sharpening in points of the picture (its size at 100%), 0 .. 100, default 1.

sharpenThreshold: 0 .. 255, default 0; smaller differences are not sharpened.

blur: *"none"* (default), *"gaussian"* (soft blur in every direction) or *"motion"* (blur along *blurAngle*).

blurRadius: Radius of the blur (distance of the motion blur) in points of the picture, 0 .. 500, default 0.

blurAngle: Direction of the motion blur in degrees, -180 .. 180, default 0.

blurTransparency: bool, default false. Blurs the transparency (alpha channel) too, which gives a cut-out picture a soft edge. Only together with *blur*.

gain, *black*, *white*, *gamma*: The values measured by the automatic corrections, each an array [red, green, blue]: white balance factor (default [1, 1, 1]), black point and white point per channel (0 .. 1, defaults [0, 0, 0] and [1, 1, 1]) and midtone exponent (default [1, 1, 1]). Keep them as read: switching an automatic correction on or off measures them again.

active: Read only: true if the corrections change the picture.

supported: Read only: false for picture objects that cannot carry corrections (empty objects and picture types other than raster images, PDF, AI, EPS and SVG).

---

pathClipping *Object* *Read/Write*

Get/Set which path clips the picture and which contour text runs around (runaround shape “Clipping Path”), as in the group Clipping of the dialog Picture Settings. The transparency of the picture always applies; the clipping chooses the path that clips in addition. The value is a plain object; reading gives a copy. When assigning, a key left out keeps its value. The keys:

clip: *"clippingPath"* (default, the clipping path saved in the file), *"alphaChannel"* (no path) or *"path"* (the path named in *clipPath*; without it the clipping path of the file).

clipPath: Name of the path for *clip* = *"path"*.

topmostPath: bool, default false. If the file has no clipping path, the topmost path clips.

contour: The runaround contour: *"sameAsClipping"* (default, the path that clips, without one the alpha channel), *"clippingPath"* (the clipping path of the file, without one the alpha channel), *"alphaChannel"*, *"alphaChannelAndPath"* (inside both) or *"path"* (the path named in *contourPath*).

contourPath: Name of the path for *contour* = *"path"*.

paths: Read only: the names of the paths of the file, topmost first.

fileClippingPath: Read only: the name of the clipping path saved in the file, empty without one.

---

hasBackgroundMask *bool* *Read*

Returns true if the picture has a background mask (from Remove Background of the image editing, or set with *setBackgroundMask()*). Soft edge, expansion, inversion and density of the mask are keys of *adjustments* (*maskFeather*, *maskExpand*, *maskInvert*, *maskDensity*).

---

backgroundMaskVisible *bool* *Read/Write*

Get/Set whether the background mask cuts the picture out. False shows the whole picture and keeps the mask. Without a background mask the value is false and cannot be set.

---

pdfPageNo *int* *Read*

Returns the placed page of a PDF file or document, counted from 1 (1 also for a picture without pages); 0 for an empty picture object.

---

isPlacedDocument *bool* *Read*

Returns true if the picture is a page of a placed VivaDesigner document (.desd or .nwpd file). For any other picture *pageBox* and *sourcePath* are empty and *sourcePageCount* is 0.

---

pageBox *String* *Read*

Returns the box of the placed document page: *"page"* (the page format) or *"bleed"* (the page with its bleed).

---

sourcePath *String* *Read*

Returns the path of the placed document.

---

sourcePageCount *int* *Read*

Returns the number of pages of the placed document.

---

image *Image* *Read/Write*

Get/Set the pixels of the picture as an object of the class *Image*. Reading returns the image at the full resolution of the file, or *null* for an empty picture object. Assigning places the image as an embedded picture.

---

preview *Image* *Read*

Returns the preview image of the picture as an object of the class *Image*.

Methods:

---

importPicture *(String filePath, int index = 0)* *bool*

Import the image to the object from the file *filePath*. The index parameter is the page number (zero based) in the imported PDF document or VivaDesigner document. A page of a VivaDesigner document (.desd or .nwpd) is placed as a placed document (see *isPlacedDocument*); append *"<=props=>box=bleed"* to the file path to place it with its bleed.

Returns TRUE if the image is imported successfully.

setClipping *(int tolerance, bool tight)* *void*

Set the parameters ***tolerance*** and ***tight*** to clip the imported image.

The parameter ***tolerance*** is in intervals of <0,255>

movePicture *(PictureHorizontalPosition hor = Ignore, PictureVerticalPosition ver = Ignore)* *void*

Moves the picture inside the object to the left, center or right and to the top, center or bottom. See the enumeration types ***PictureHorizontalPosition*** and ***PictureVerticalPosition***; ***Ignore*** keeps the position in that direction.

scalePicture*(ScalePictureType type = Proportional, bool center = true)* *void*

Scales the picture, see the enumeration type ***ScalePictureType***. ***FitToPicture*** adapts the object to the size of the picture, ***OriginalSize*** sets the scale to 100%. ***center***: the picture is centered in the object.

optimizePicture*(OptimizePictureSettings settings)* *bool*

Optimize the image. Returns ***TRUE*** if success.

changeColorSpace*(ChangePictureColorSpaceSettings settings)* *bool*

Change the picture color space.

getAttributes *()* *Attributes*

setAttributes *(Attributes attrs)* *void*

copyAttributes *(Object fromObj)* *boolean*

Read, set or copy all attributes of the picture content as an ***Attributes*** object. See class ***Attributes***.

setBackgroundMask *(String filePath)* *bool*

Sets the background mask of the picture from a grayscale image file: white shows, black hides; of an image with an alpha channel its alpha is used. An empty ***filePath*** removes the mask. Returns false if the file is not an image; the picture keeps its mask then.

removeBackgroundMask() *void*

Removes the background mask: the whole picture shows again.

copyContentFrom *(PictureContent fromObj)* *bool*

Places the picture of ***fromObj*** in this picture object as well. Returns true in case of success.

saveToFile *(String fullPath, String format = null, int quality = -1)* *void*

Saves the pixels of the picture (see ***image***) to an image file. ***format***: e.g. ***"PNG"*** or ***"JPG"***; without it the file extension sets the format. ***quality***: 0 .. 100 for formats with compression, -1 for the default. If the file cannot be written, ***errorStatus*** is ***IsNotValid***.

imageData *(String format, int quality = -1)* *String*

Returns the pixels of the picture as an image file in the format ***format*** (e.g. ***"PNG"*** or ***"JPG"***), Base64 encoded; an empty string on failure.

setImageData *(String data, String format = null)* *void*

Sets the picture from Base64 encoded image data, e.g. from ***imageData()***. Without ***format*** the format is detected from the data. The image becomes an embedded picture.

Example:

```
  var doc = application.currentDocument;
  var pc = doc.currentPage.objects().at(0).content;   // content of a
picture object
  // Warmer and with more contrast: read a copy, change it, assign it back
  var a = pc.adjustments;
  if (a.supported) {
    a.temperature = 30;
    a.curve = [[0, 0], [0.25, 0.2], [0.75, 0.8], [1, 1]];
    pc.adjustments = a;
  }
  // Text runs around the alpha channel
  pc.pathClipping = { contour: "alphaChannel" };
  // A soft edge for the background mask
  if (pc.hasBackgroundMask) {
    a = pc.adjustments;
    a.maskFeather = 4;
    pc.adjustments = a;
  }
```

Class: TextChains

The class holds list of TextChain objects.

Properties:

length *int* *Read*

Returns the number of text chains in the list.

Methods:

at *(int index)* *TextChain*

Returns the text chain at the position ***index*** (0 .. length-1).

Example:

```
var chns = doc.textChains(WhichSpreads.All, WhichObjects.All);
if (chns)
{
 doc.textChains(WhichSpreads.All,
WhichObjects.All).at(0).text.appendPlainText("Text");
 var text = doc.textChains(WhichSpreads.All,
WhichObjects.All).at(0).text.plainText();

 // or ...

 for (var i=0; i<chns.length; i++)
 {
   var chn = chns.at(i);
   var formatText = chn.text;
```

```
   var plainText =  formatText.plainText();

   // ...
 }
}
```

Class: TextChain

The class "TextChain" holds a list of text objects that are linked and contain one text that continues to the next and following text objects.

Properties:

areaCount	*int*	*Read*

Returns the number of chained text objects.

text	*FormattedText*	*Read*

Returns the object FormattedText with the whole text that is placed in linked objects.

Methods:

areaAt *(int index) TextContent*

Returns the text content (class ***TextContent***) of the text object at the index position in the text chain.

areaIndex *(TextContent area)* *int*

Returns the index of the text content ***area*** in the text chain.

equals *(TextChain obj)* *bool*

Returns ***TRUE*** if the text chains are identical. They have same ID.

Class: TableObject

The class ***TableObject*** implements a table which is drawn into the frame object. The position, size, shape, border, background and other features of the table object are defined by the values of properties derived from the class FrameObject. See description in the chapter "Class: FrameObject". The position is set by the properties pagex and pagey (class Object), the size by width and height (class FrameObject) or by tableWidth and tableHeight.

Properties:

tableHeight	*UnitValueString*	*Read/Write*

Defines table object height.

tableWidth	*UnitValueString*	*Read/Write*

Defines table object width.

tableWidthType	*WidthType*	*Read/Write*

Defines how the width of the table object is calculated. Permitted values are specified by the enumeration type ***WidthType***. See the description at the end of this chapter.

rowCount	*int*	*Read/Write*

Set/get the number of table rows.

columnCount *int* *Read/Write*

Set/get the number of table columns.

horizontalSeparatorAbove *bool* *Read/Write*

Sets the horizontal separator lines over the vertical separator lines and is the default setting for all new table objects. It only makes sense to use it when the color of the horizontal separator lines is different from the color of the vertical separator lines. The vertical separator is set over the horizontal separator when this property is set to false.

styleSheet *TableStyleSheet* *Read/Write*

styleSheetByName *String* *Read/Write*

Set/get the style sheet that is applied to define the appearance of the table object. The style sheet can be defined by name or by an object *TableStyleSheet*.

Methods:

column *(int index)* *TableColumn*

Returns an object that represents the whole table column.

index: Index of the required column. The value must be in the range from 0 to (columnCount – 1).

Returns a TableColumn object or *null* if the column doesn't exist.

row *(int index)* *TableRow*

Returns an object that represents the whole table row.

index: Index of the required row. The value must be in the range from 0 to (rowCount – 1).

Returns a TableRow object or *null* if the row doesn't exist.

cell *(int row, int column)* *TableCell*

Returns an object that represents one particular table cell.

row: Index of a row in which the required cell is located. The value must be in the range from 0 to (rowCount – 1).

column: Index of a column in which the required cell is located. The value must be in the range from 0 to (columnCount – 1).

Returns a TableCell object or *null* if the cell doesn't exist.

separator *(int row, int column, TableSeparatorPosition position)* *TableSeparator*

Returns selected table separator. i.e. parameters of a line of the appropriate cell frame. See the specification of table separator in chapter "Class: TableSeparator".

row: Row index of the required table cell.

column: Column index of the required table cell.

position: Parameter defines which part of the cell border the function must return.

Returns the required TableSeparator object or *null* if the separator is not found.

setFillStyle *(Brush brush, float shade, float opacity, float blendAngle) void*

Defines the fill parameters of the table.

brush: Color of the table object fill.

shade: The parameter defines the color shade of the table fill color. The maximum value is 100.0 (100%). The value 0 means that the table fill is white, but not transparent. The value -1 means that the table fill is fully transparent.

opacity: Defines the opacity of the table fill color. The value 100.0 (i.e. 100%) means that the fill has no translucence/opacity. The value 0 means that the fill is fully transparent.

blendAngle: If the table object is filled by a color blend instead of a solid color, this parameter defines the angle of the color blend.

---

remove() *boolean*

Removes table object.

---

importRange *(String filePath, Object options = {})* *Object*

Writes cells of an Excel workbook (.xlsx), a Numbers document (.numbers) or a CSV file into this table, as Table > Import... does, but without any dialog. By default only the values change and the table keeps its look; a range that does not fit into the table changes nothing (errorCode *"does-not-fit"*). The import is one step of Edit > Undo.

filePath: Absolute path of the file.

options: A plain object with the following keys, all optional. An unknown key gives errorCode *"unknown-parameter"*, a wrong type or value *"bad-params"*; nothing is read then.

cell: First cell of this table that is written, e.g. *"B2"*; default *"A1"*.

format: *"xlsx"*, *"numbers"* or *"csv"*; default: from the file name.

sheet: Name of the sheet of a workbook or Numbers document; default: the first sheet with cells.

table: Name of the table on the sheet of a Numbers document; default: the first one.

range: Cells of the file: *"B2:F40"*, whole columns *"C:C"*, whole rows *"2:5"* or one cell *"B2"*; default: the used range.

withFormatting: bool, default false. Fonts, colors, borders and alignment of the file are taken over; with false the table keeps its look and only the values change. CSV files have no formatting.

columnWidths: bool, default false. The column widths of the file are taken over.

fit: Cells of the file beyond the edge of the table: *"cut"* leaves them out, *"expand"* adds rows and columns. Without *fit* nothing is imported if the range does not fit.

mergedCells: Merged cells and cells without text in the table, as the question “Import Tab Text” of Table > Import...: *"merge"* (default) gives a merged cell the values of all cells it covers, *"suppress"* only the first one, *"keep"* splits it so that every value gets a cell of its own and turns a cell without text into a text cell.

delimiter: CSV only: *"auto"* (default, detected), *"semicolon"*, *"comma"* or *"tab"*.

encoding: CSV only: *"auto"* (default, detected), *"utf-8"*, *"windows-1252"* or *"mac-roman"*.

quote: CSV only: bool, default true. Fields may be enclosed in double quotes.

Returns a result object (see below) with these keys in addition on success: *table*, *name*, *page* (identifier, name and page number of the table), *sheet*, *sourceTable*, *range* (what was read: sheet, Numbers table and range of the file), *cells*, *rows*, *columns* (the cells written in this table, e.g. *"B2:D4"*, and their number of rows and columns), *expanded* (true if rows or columns were added), *cut* (*{rows, columns}* left out), *csv* (for a CSV file the *{delimiter, encoding, byteOrderMark}* found), *undo* (the text of the entry in Edit > Undo) and *report*. With errorCode *"does-not-fit"* the keys *tableRows*, *tableColumns*, *needRows* and *needColumns* give the size of the table and the size the import needs.

report: What Table > Import... reports, as an object of lists of cell names; keys without findings are left out: *numberFormat* (number formats VivaDesigner cannot show; the cells show the plain value), *formulaWithoutValue* (formulas without a stored result; the cells stay empty), *borderStyle* (dash-dot borders, drawn dashed), *diagonalBorder* (not taken over), *hiddenRows*, *hiddenColumns* (imported visible), *groupRows* (group and summary rows of tables with categories or a pivot layout, not taken over), *pictures* and *damaged* (numbers of pictures and charts on the sheet, not taken over, and of damaged parts of the file), *missingFonts* (fonts that are not installed) and, for cells of this table, *outerBorderCells* (borders on the outer edge; the border of the table stays as it is), *spanCells* (merged cells that could not be merged) and *overflowCells* (text that does not fit into its cell).

---

exportRange *(String filePath, Object options = {})* *Object*

Writes cells of this table into an Excel workbook (.xlsx) or a CSV file, as Table > Export... does, but without any dialog. An existing file is replaced.

filePath: Absolute path of the file.

options: A plain object with the following keys, all optional; unknown keys and wrong values as for *importRange()*.

cells: Cells of this table, e.g. *"A1:C3"*; default: the whole table.

format: *"xlsx"* or *"csv"*; default: from the file name.

sheet: XLSX only: the name of the sheet; default as Table > Export...

withFormatting: XLSX only: bool, default true. Fonts, colors, borders and number formats are written into the file.

delimiter: CSV only: *"auto"* (default, a semicolon where numbers are written with a decimal comma, else a comma), *"semicolon"*, *"comma"* or *"tab"*.

byteOrderMark: CSV only: bool, default true. The file is written as UTF-8 with a byte order mark, which Excel needs to read accented characters.

Returns a result object with these keys in addition on success: *file* (the file written), *format*, *cells* (the cells of this table in the file, e.g. *"A1:C3"*), *rows* and *columns*.

---

fileInfo *(String filePath, Object options = {})* *Object*

Returns the sheets of a workbook, the sheets and tables of a Numbers document or the size of a CSV file, each with its used range, to choose *sheet*, *table* and *range* for *importRange()*. The method only reads the file; it needs neither this table nor the active document.

options: *format* (*"xlsx"*, *"numbers"* or *"csv"*; default: from the file name) and *previewRows* (0 .. 10: the first rows of each sheet are returned as text; default none).

Returns a result object with the keys *format* and *sheets* in addition on success. *sheets* is an array with one object per sheet (one for a CSV file): *name*, *usedRange* (the cells that hold something, e.g. *"A1:C8"*), *hidden*, *tables* (Numbers: the tables on the sheet, each *{name, usedRange}*), *preview* (with *previewRows*: the first rows as arrays of texts) and, for a CSV file, *delimiter*, *encoding*, *rows* and *columns*.

The result object of the three methods always has the keys *success* (bool), *errorCode* and *error* (*""* on success, else a code and an English text) and *messages* (texts of message boxes the program caught on the way; none was shown). The methods throw no exception; *errorStatus* is set: *Ok* on success, *IsNotAccessToFileSystem* for *"no-such-file"* and *"write-failed"*, *IsNotUsable* for *"not-active"*, *"not-supported"*, *"read-only"*, *"formatting-only"*, *"object-locked"*, *"no-document"*, *"no-reader"* and *"no-writer"*, *IsNotValid* for *"not-found"* and *"not-a-table"*, *IsNotProcessed* for all other codes.

Error codes before anything is read: *"not-found"* (the table object is not valid), *"missing-parameter"* (no file path), *"bad-params"* (file path not absolute, an option of a wrong type or value), *"unknown-parameter"*, *"not-active"* (the document of the table is not *application.currentDocument*; a document opened without showing it cannot be used) and *"not-supported"*. Error codes from the program: *"no-document"*, *"read-only"*, *"formatting-only"*, *"object-locked"*, *"bad-cell"*, *"bad-range"*, *"no-such-file"*, *"unsupported-format"*, *"no-reader"*, *"no-such-sheet"*, *"no-such-table"*, *"read-failed"*, *"too-large"*, *"empty"*, *"does-not-fit"*, *"not-imported"*, *"write-failed"* and *"no-writer"*.

Examples:

```
var doc = application.currentDocument;
if (doc) {
  // Create and name new table object
  var oTbl = doc.objects().create(ObjectType.Table);
  oTbl.name = "Table 1";
  // Set number of rows and columns
  oTbl.rowCount = 5;
  oTbl.columnCount = 3;
  // Set position and size of the table
  oTbl.tableWidthType = WidthType.Manual;
  oTbl.pagex = "10mm";
  oTbl.pagey = "10mm";
  oTbl.width = "120mm";
  oTbl.height = "150mm";
  // Colorize interior of the table
  oTbl.setFillStyle(doc.brushes.find("[Green]"), 20, 100, 0);

  // Add the table to a current page
  doc.currentPage.add(oTbl);

  // Get a table cell and change its color
  var tblCell = oTbl.cell(1, 1);
  if( tblCell ) {
    tblCell.brush = doc.brushes.find("[Red]");
    tblCell.shade = 70;
  }

  // Retrieve content type of the cell
  if (tblCell.contentType == ContentType.Custom)
    messageBox.information("Content Type", "Custom");
  if (tblCell.contentType == ContentType.Text)
```

```
      messageBox.information("Content Type", "Text");
    if (tblCell.contentType == ContentType.Picture)
      messageBox.information("Content Type", "Picture");
    if (tblCell.contentType == ContentType.Table)
      messageBox.information("Content Type", "Table");
  }
```

Example: import and export of cells

```
var doc = application.currentDocument;
var oTbl = doc.currentPage.objects().at(0);   // a table object
var info = oTbl.fileInfo("C:/Data/sales.xlsx", { previewRows: 2 });
if (info.success) {
  var sheet = info.sheets[0];
  var r = oTbl.importRange("C:/Data/sales.xlsx",
    { sheet: sheet.name, range: sheet.usedRange, fit: "expand" });
  if (!r.success)
    messageBox.information("Import", r.error);
}
oTbl.exportRange("C:/Data/table.csv", { delimiter: "semicolon" });
```

Class: TableRow

The class ***TableRow*** represents the settings for one table row. The row settings take effect on all the table cells in the table row, unless an individual cell or cells is/are assigned attributes that take priority over the row settings, such as fill color or shade.

Properties:

height	*UnitValueString*	*Read/Write*

Defines the table row height. See the property heightType to see how this property value is interpreted.

heightType	*HeightType*	*Read/Write*

Defines how to interpret value in property height. See the enumeration type ***HeightType*** at the end of this chapter to see available values.

blendAngle	*float*	*Read/Write*

Defines the angle of the color blend if the table row is filled by a color blend instead of a solid color.

shade	*float*	*Read/Write*

Defines the shade of the table row fill color. The maximum value is 100.0 ((i.e. 100%).The value 0 means that the table row is white, but not transparent. The value -1 means that the table row is fully transparent. If the row fill color is a blend instead of a solid color, then only the values 100% or Transparent (-1) are relevant.

brush	*Brush*	*Read/Write*

The table row color is defined by the term ***Brush***. See the chapter “Class: Brush” in this documentation.

styleSheetByName	*String*	*Read/Write*
styleSheet	*TableRowStyleSheet*	*Read/Write*

Set/get the style sheet applied to define the appearance of the table row object. The style sheet can be defined by name or by an object `TableRowStyleSheet`.

Methods:

setFillStyle *(Brush brush, float density, float opacity, float blendAngle) void*

Defines the filling parameters of the table row.

brush: Color of the table row.

shade: The parameter defines the shade of the table row fill color. The maximum value is 100.0 (100%). The value 0 means that the table row is white, but not transparent. The value -1 means that the table row is fully transparent. If the row fill color is a blend instead of a solid color, then only the values 100% or Transparent (-1) are relevant.

opacity: The parameter defines the opacity of the table row. The value 100.0 (i.e. 100%) means that the row has no translucence/opacity. The value 0 means that the table row is fully transparent.

blendAngle: Defines the angle of the color blend if the table row is filled by a color blend instead of a solid color.

getAttributes *()*	*Attributes*
setAttributes *(Attributes attrs)*	*void*

copyAttributes *(Object fromObj) boolean*

Read, set or copy all attributes of the table row as an *Attributes* object. See class *Attributes*.

Examples:

See examples in the chapters “Class: TableObject” and “Class: TableCell”.

Class: TableColumn

The class *TableColumn* represents the settings for one table column. The column settings take effect on all the table cells in one table column unless an individual cell or cells is/are assigned attributes that take priority over the column settings, such as fill color or shade..

Properties:

width *UnitValueString* *Read/Write*

Defines the table column width. See the property *widthType* to see how this property value is interpreted.

widthType *WidthType* *Read/Write*

Defines the interpretation of the value in the property width. See the enumeration type *WidthType* at the end of this chapter to see available values for this property.

blendAngle *float* *Read/Write*

Defines the angle of the color blend if the table column is filled by a color blend instead of a solid color.

shade *float* *Read/Write*

Defines the shade of the table column fill color. The maximum value is 100.0 (100%). The value 0 means that the table column is white, but not transparent. The value -1 means that the table column is fully transparent. If the column fill color is a blend instead of a solid color, then only the values 100% or Transparent (-1) are relevant.

brush	*Brush*	*Read/Write*

The table column color is defined by the term ***Brush***. See the chapter “Class: Brush” in this documentation.

styleSheetByName	*String*	*Read/Write*

styleSheet	*TableColumnStyleSheet*	*Read/Write*

Set/get style sheet that is applied to define the appearance of the table column object. The style sheet can be defined by name or by an object ***TableColumnStyleSheet***.

Methods:

setFillStyle *(Brush brush, float density, float opacity, float blendAngle) void*

Sets filling parameters of the table column.

brush: Color of the table column.

shade: The parameter defines the color shade of the table column fill color. The maximum value is 100.0 (100%). The value 0 means that the table column is white, but not transparent. The value -1 means that the table column is fully transparent.

opacity: The parameter defines the opacity of the table column. The value 100.0 (i.e. 100%) means that the column has no translucence/opacity. The value 0 means that the table column is fully transparent. If the column fill color is a blend instead of a solid color, then only the values 100% or Transparent (-1) are relevant.

blendAngle: Defines the angle of the color blend if the table column is filled by a color blend instead of solid color.

getAttributes *()*	*Attributes*
setAttributes *(Attributes attrs)*	*void*
copyAttributes *(Object fromObj)*	*boolean*

Read, set or copy all attributes of the table column as an ***Attributes*** object. See class ***Attributes***.

Examples:

See examples in chapters “Class: TableObject” and “Class: TableCell”.

Class: TableCell

The class ***TableCell*** represents one particular table cell. The cell is accessed in a table via row and column index using method cell(row, column). Each cell can contain another table or text or picture, or may have the content setting “None”. Cell properties such as fill color take preference over the fill color properties of columns and rows.

Properties:

blendAngle	*float*	*Read/Write*

Defines the angle of the color blend if the table cell is filled by a color blend instead of solid color.

shade	*float*	*Read/Write*

Defines the shade of the table cell fill color. The maximum value is 100.0 ((i.e. 100%). The value 0 means that the table cell is white, but not transparent. The value -1 means that the table cell is fully transparent. If the cell fill color is a blend instead of a solid color, then only the values 100% or Transparent (-1) are relevant.

brush	*Brush*	*Read/Write*

The table cell color is defined by the term ***Brush***. See the chapter “Class: Brush” in this documentation.

leftIndent	*UnitValueString*	*Read/Write*
topIndent	*UnitValueString*	*Read/Write*
rightIndent	*UnitValueString*	*Read/Write*
bottomIndent	*UnitValueString*	*Read/Write*

Define the indent of the cell content from the left, top, right and bottom boundaries of the table cell.

horizontalAlignment	*HorizontalAlignment*	*Read/Write*

Horizontal alignment of an object inside the table cell when the cell content is “None” and an object is imported into the cell. See enumeration type HorizontalAlignment at the end of this chapter to see all allowed values for this property.

verticalAlignment	*VerticalAlignment*	*Read/Write*

Vertical alignment of an object inside the table cell when the cell content is “None” and an object is imported into the cell. See enumeration type VerticalAlignment at the end of this chapter to see all allowed values for this property.

contentRotation	*ContentRotation*	*Read/Write*

The cell content can be rotated by 0°, 90°, 180° or 270°. This property defines the rotation angle and applies to all cell content including imported objects when the cell content is “None” and an object is imported into the cell. See enumeration type ContentRotation to see available values for this property.

rowSpan	*int*	*Read/Write*

Any table cell can be merged with one or more adjacent cells in one table column. The property ***rowSpan*** defines the number of merged cells, i.e. number of rows from which the cells are merged to one cell. A single cell has rowSpan = 1. In the VivaDesigner Table Dialog, the option is “Extend Cell”.

columnSpan	*int*	*Read/Write*

Any table cell can be merged with one or more adjacent cells in one table row. The property columnSpan specifies number of merged cells, i.e. number of columns, from which the cells are merged to one cell. A single cell has columnSpan = 1. In the VivaDesigner Table Dialog, the option is “Extend Cell”.

contentType	*ContentType*	*Read*

Defines the cell content type. See enumeration type ***ContentType*** at the end of this chapter to see allowed content types.

contentObject	*Object*	*Read/Write*

Set/get the content object of the table cell. It can be a picture object, text object or another table object. The cell content type is not assigned to the inserted object.

contentExpandableObject	*Object*	*Write*

Sets the content object of the table cell. It can be a picture object, text object or another table object. The cell content type is assigned to the inserted object and the object parameters are affected. See the property ***contentType*** to see details about expected content type.

styleSheetByName	*String*	*Read/Write*

styleSheet	*TableCellStyleSheet*	*Read/Write*

Set/get the style sheet that is applied to define the appearance of the table cell object. The style sheet can be defined by name or by an object `TableCellStyleSheet`.

row	*int*	*Read*

Returns the row index of the cell.

column	*int*	*Read*

Returns the column index of the cell.

valid	*bool*	*Read*

Returns true if the object stands for an existing cell of the table.

Methods:

removeContent()	*void*

Resets the table cell content.

setFillStyle*(Brush brush*, *float shade*, *float opacity*, *float blendAngle)* *void*

Sets the cell filling parameters.

brush: Table cell fill color.

shade: Defines the shade of the table cell fill color. The maximum value is 100.0 ((i.e. 100%). The value 0 means that the table cell is white, but not transparent. The value -1 means that the table cell is fully transparent. If the cell fill color is a blend instead of a solid color, then only the values 100% or Transparent (-1) are relevant.

opacity: Defines the table cell opacity. The value 100.0 (i.e. 100%) means that the cell has no translucence/opacity. The value 0 means that the cell is fully transparent.

blendAngle: Defines the angle of the color blend if the table cell is filled by a color blend instead of solid color.

setIndents *(UnitValueString left*, *UnitValueString top*, *UnitValueString right*, *UnitValueString bottom)* *void*

Sets the indent of the content from left, top, right and bottom boundaries of the table cell.

getAttributes *()*	*Attributes*

setAttributes *(Attributes attrs)*	*void*

copyAttributes *(Object fromObj)*	*boolean*

Read, set or copy all attributes of the table cell as an ***Attributes*** object. See class ***Attributes***.

Examples:

```
var doc = application.currentDocument;
if (doc) {
```

```
// Create example table
var oTbl = doc.objects().create(ObjectType.Table);
oTbl.name = "Examples: Tables";

// Define number of rows and columns
oTbl.rowCount = 4;
oTbl.columnCount = 3;

// Size of the table. The width is fixed.
oTbl.pagex = "10mm";
oTbl.pagey = "10mm";
oTbl.tableWidthType = WidthType.Manual;
oTbl.width = "120mm";
oTbl.height = "150mm";

// Insert the table to a current page
doc.currentPage.add(oTbl);

// Get table row
var oTblRow = oTbl.row(2);
if( oTblRow ) {
  // Minimum height of the row is 20mm
  oTblRow.heightType = HeightType.Minimum;
  oTblRow.height = "20mm";
  // Create a blend brush for the table row.
  // First check if the brush exists. If a brush with existing
  // name is created, an error occurs.
  var brBl = doc.brushes.find("TblRowBrush");
  if( brBl == null ) {
    brBl = doc.brushes.addBlend("TblRowBrush");
    brBl.type = BlendType.Linear;
    brBl.setStopAt(0, new Stop(Color.Red, 70, 40, 0, 85));
    brBl.setStopAt(1, new Stop(Color.White, 100, 55, 100, 23));
    var stop3 = new Stop(Color.Green, 50, 55, 60, 25);
    brBl.appendStop(stop3)
  }
  // Set the brush to the table row.
  oTblRow.brush = brBl;
  // Don't forget to make it visible. By default
  // the background of the row is transparent.
  oTblRow.shade = 100;
}
// Prepare a graphic object for one of the cells
var oFrm = doc.objects().create(ObjectType.Graphic);
with (oFrm) {
  // Ellipse with blue border and yellow interior
  form = FrameForm.Ellipse;
  frameWidth = "8pt";
  pagex = "0mm";
  pagey = "0mm";
  width = "30mm";
  height = "20mm";
```

```
    frameColor = doc.brushes.find("[Blue]");
    fillColor = doc.brushes.find("[Yellow]");
    frameShade = 80;
    fillShade = 70;
  }
  // Get one of the cells and insert the ellipse there
  var oTblCell = oTbl.cell(1,0);
  if( oTblCell ) {
    oTblCell.contentObject = oFrm;
  }
  // Prepare the text object for another table cell
  var oTxt = doc.objects().create(ObjectType.Text);
  with(oTxt) {
    pagex = "25mm";
    pagey = "45.5pt";
    width = "30mm";
    height = "20mm";
    content.insertPlainText("Text in a cell", 0);
  }
  // Get a table cell
  oTblCell = oTbl.cell(1,2);
  if( oTblCell ) {
    // Define indents inside the cell
    oTblCell.leftIndent = 10;
    oTblCell.topIndent = 5;
    oTblCell.rightIndent = 10;
    oTblCell.bottomIndent = 5;
    // Rotate the text by 90 degrees
    oTblCell.contentRotation = ContentRotation.Angle90;
    // Insert the text in the cell
    oTblCell.contentObject = oTxt;
  }
}
```

Class: TableSeparator

The class ***TableSeparator*** describes the parameters of one particular separator line of a table cell. See the method ***separator()*** of the class ***TableObject***, which returns an object representing the required separator.

Properties:

visible	*bool*	*Read/Write*

Set this property to true to make the separator visible or to false to hide the separator.

shade	*float*	*Read/Write*

Defines the color shade of the separator color. The maximum value is 100.0 (100%). The value 0 means that the separator is white, but not transparent. The value -1 means that the separator is fully transparent.

brush	*ColorBrush*	*Read/Write*

The color of the separator is defined by the term ***ColorBrush***. See chapter “Class: ColorBrush” in this documentation.

Methods:

setFrameStyle *(ColorBrush brush, float shade, float opacity, String lineStyle, UnitValueString lineWidth)*	*void*

This method implements the formatting of the cell separator in a complex way.

brush: Defines the separator color. Blends are not permitted.

shade: Defines the shade of the separator color. The maximum value is 100.0 (100%). The value 0 means that the separator is white, but not transparent. The value -1 means that the separator is fully transparent.

opacity: The parameter defines the opacity of the separator. The value 100.0 (i.e. 100%) means that the separator has no translucence/opacity. The value 0 means that the separator is fully transparent.

lineStyle: The parameter defines the style of the line used to draw the separator. Available values are: “Solid” (solid line), “Pattern 1” (dashed), “Pattern 2” (dashed with shorter gaps), “Pattern 3” (dashdot line), “Pattern 4” (dotted, squared dots), “Pattern 5” (dashed with rounded ends of dashes), “Pattern 6” (dashed with shorter gaps and rounded ends of dashes), “Pattern 7” (dashdot line rounded ends of dashes), “Pattern 8” (dotted, circular dots), “Pattern 9” (dashed with short dashes), “Pattern 10” (dashdot line with short dashes), etc. up to “Pattern 19” as combination of longer or shorter dashes, circular or squared dots.

lineWidth: Defines the line width of the separator, e.g. ***"0.5mm"***; a number without a unit is in points (pt).

getAttributes *()*	*Attributes*
setAttributes *(Attributes attrs)*	*void*
copyAttributes *(Object fromObj)*	*boolean*

Read, set or copy all attributes of the table separator as an ***Attributes*** object. See class ***Attributes***.

Examples:

```
var doc = application.currentDocument;
if (doc) {
   // Create example table
   var oTbl = doc.objects().create(ObjectType.Table);
oTbl.name = “Examples: Tables”;
   // Define number of rows and columns
   oTbl.rowCount = 2;
   oTbl.columnCount = 3;

   // Size of the table. The width is fixed.
   oTbl.pagex = “10mm”;
   oTbl.pagey = “10mm”;
   oTbl.tableWidthType = WidthType.Manual;
   oTbl.width = “100mm”;
   oTbl.height = “80mm”;

   // Insert the table to a current page
```

```
  doc.currentPage.add(oTbl);

  // Make the left cell separator red
  var oSep = oTbl.separator(1, 1, TableSeparatorPosition.Left);
  oSep.visible = true;
  oSep.brush = doc.brushes.find("[Red]");
  oSep.shade = 100;
  // Make the right separator blue, thick and using non-solid line
  oSep = oTbl.separator(1, 1, TableSeparatorPosition.Right);
  oSep.visible = true;
  oSep.setFrameStyle(doc.brushes.find("[Blue]"), 80, 50, "Pattern 1",
"5pt");
}
```

Class: GroupObject

The group object is an invisible object that only groups several page or spread objects and allows some manipulation on all grouped objects, particularly moving, resizing and rotation.

Properties:

`length` *`int`* *`Read`*

The property contains a number of grouped objects.

Methods:

`objects`*`(ObjectType objectType = All)`* *`Objects`*

Get the set of objects which are grouped in this group object. See chapter “Class: Objects” for all information about the objects collection class.

`objectType`: See the enumeration ObjectType.

`objectCount()` *`int`*

Returns the number of grouped objects, like ***`length`***.

`getObject`(int index) *`Object`*

Returns the grouped object at the position ***`index`*** (0 .. length-1), or ***`null`***.

`ungroup()` *`void`*

Release all objects that are grouped in this group object. A chart object or another object built by an add-in is not dissolved; ***`errorStatus`*** is ***`IsNotProcessed`*** then (see ***`ChartObject.toGroup()`***).

`insertAfter`*`(Object obj, Object afterObj = null)`* *`void`*

Insert an object in the group object, directly after the object ***`afterObj`*** in the order of the group.

`obj`: Newly inserted object.

`afterObj`: The object after which the new object is inserted. Without it, the new object gets the position 0, as with ***`add()`***.

`insertAt`*`(Object obj, int pos)`* *`void`*

Insert an object in the group object after the object at the position ***`pos`***.

obj: Newly inserted object.

pos: Position of the object after which the new object is inserted, 0 .. length-1; the new object gets the position pos + 1. With a position outside this range, the new object gets the position 0, as with *add()*.

---

add*(Object obj)* *void*

Add an object to the group object. The new object gets the position 0 (see *getObject()*); the objects grouped before move up by one.

obj: Newly inserted object.

An object built by an add-in, such as a chart object, takes further objects only if the add-in allows it; otherwise *add()*, *insertAt()* and *insertAfter()* change nothing.

---

remove() *bool*

Removes group object.

---

asChart() *ChartObject*

Returns this group as a *ChartObject* if it is a chart object of the chart add-in, otherwise *null*. See class *ChartObject*.

Examples:

```
var doc = application.currentDocument;
if (doc) {
  var objs = doc.objects();

  // Create several objects
  var obj1 = objs.create(ObjectType.Text);
  obj1.name = "Obj1";
  obj1.pagex = "20pt"; obj1.pagey = "20pt"; obj1.width = "100pt";
obj1.height = "100pt";

  var obj2 = objs.create(ObjectType.Graphic);
  obj2.name = "Obj2";
  obj2.pagex = "60pt"; obj2.pagey = "60pt"; obj2.width = "150pt";
obj2.height = "150pt";

  var obj3 = objs.create(ObjectType.Text);
  obj3.name = "Obj3";
  obj3.pagex = "210pt"; obj3.pagey = "210pt"; obj3.width = "50pt";
obj3.height = "50pt";

  // Add the objects to the current page
  var page = doc.currentPage;
  page.add(obj1);
  page.add(obj2);
  page.add(obj3);
  messageBox.information("Objects in Page", "Count = " +
page.objects().length
        + "\n(no group exist)");
  // Create group object
  var group = objs.create(ObjectType.Group);
  group.name = "First Group";
```

```
  group.pagex = "50mm";
  group.pagey = "150mm";
  page.add(group);

  messageBox.information("Objects in Page", "Count = " +
page.objects().length
        + "\n(including new group)");
  // Add 2 objects to the group
  group.add(obj2); // add to the end
  group.insertAt(obj3, 0); // insert to 1st position
  messageBox.information("Objects in Group", "Count = " + group.length);
}
```

Class: ChartObject

The class ***ChartObject*** extends the class ***GroupObject***. It stands for a chart object of the chart add-in, as the Chart Object Tool draws it and the window Chart Data edits it. For everything else (position, name, layer, ***remove()*** ...) it is a group object. Its parts are built by the add-in: they cannot be added, removed or ungrouped one by one.

A chart object is created with the method ***createChart()*** of the class ***DocumentPage*** and found with ***GroupObject.asChart()***. Every change is one step of Edit > Undo. Creating and changing a chart needs a licence that includes charts in layout documents; without it the methods change nothing (errorCode ***"not-licensed"***), reading still works. The document of the chart must be the active document (***application.currentDocument***).

Properties:

isChart *bool* *Read*

Returns true if the object is a chart object; false if ***createChart()*** failed (see ***lastResult***) or after ***toGroup()***.

editable *bool* *Read*

Returns true if the licence allows changing the chart.

model *Object* *Read*

Returns the data and options of the chart: ***type***, ***grouping***, ***title***, ***subtitle***, ***legend***, ***legendPosition***, ***categories***, ***series*** (each ***{name, values}***) and ***options*** in the form ***setOptions()*** takes them; ***{}*** if the object is no chart.

lastResult *Object* *Read*

Returns the result object of the last method call of this object (see below).

Methods:

info() *Object*

Returns information on the chart. The result object also has the keys ***chart*** (true for a chart object), ***kind*** (***"chart"***), ***editable***, ***locked***, ***layout*** and ***databasePublishing*** (the licence includes charts in layout documents, in database publishing), ***dataVersion***, ***builtWidth*** and ***builtHeight*** (the size in points the chart was built for), ***frame*** (***{x, y, width, height}***: the chart area on the page in points), ***model*** and ***warnings***.

setData *(Array categories, Array series = null)* *Object*

Sets new categories and/or series. A series keeps the look of the series at its position. One undo step "Edit Chart".

categories: Array of texts: the categories (labels of the category axis, slices of a pie). *null* or *undefined* keeps them.

series: Array of objects *{name, values}*; *values* is an array of numbers, *null* for a gap. *null* or *undefined* keeps them.

---

setOptions *(Object options)* *Object*

Changes the type and the options of the chart; only the options given change. One undo step "Edit Chart". The chart add-in checks every value; a wrong one gives errorCode *"bad-params"* with the name of the value, and nothing changes. The options:

type: *"column"*, *"bar"*, *"line"*, *"area"*, *"pie"*, *"ring"* or *"combo"* (columns, the last series as a line on a second value axis).

grouping: *"clustered"*, *"stacked"* or *"percentStacked"* (not for pie, ring and combo).

title, *subtitle*: Texts; *""* for none. *legend*: bool. *legendPosition*: *"r"*, *"l"*, *"t"*, *"b"* or *"tr"* (right, left, top, bottom, top right).

locale: Language of the numbers, e.g. *"de-DE"* or *"en-US"*; *""* for the language of the program. *groupSeparator*: bool, numbers with a thousands separator. *blanksAs*: empty values of a line chart: *"gap"*, *"span"* (joined across) or *"zero"*.

skipZeros, *sortDescending*, *counterClockwise*, *relativeToFirst*, *negativeLabelsLeft*: Switches of the chart (true or false; *negativeLabelsLeft* also *null*).

pieLayout: *"office"*, *"classic"*, *"modern"* or *"balanced"*. *startAngle*: start angle of a pie or ring. *holeSize*: hole of a ring, 10 .. 90. *gapWidth*: gap between the categories of a column or bar chart in percent of the width of a column, 0 .. 500.

secondaryAxis: bool (combo: the line on the second value axis). *legendTabs*: array of *{position, align}* (position 0 .. 100, align *"left"*, *"center"*, *"right"* or *"decimal"*).

separators: *{colour, width}*, the lines between the slices of a pie. *areaFill*, *areaLine*, *plotFill*, *legendLine*: colors of the chart area, its outline, the plot area and the outline of the legend, or *null*.

texts: Text formats per element (*all*, *title*, *subtitle*, *legend*, *categoryAxis*, *valueAxis*, *secondaryAxis*, *dataLabels*, *negativeLabels*), each *{family, fontStyle, bold, italic, size, colour, style, rotation}*.

series: One object per series by position (*null*: unchanged) with *fill*, *line*, *lineWidth*, *lineStyle* (*"solid"*, *"dashed"*, *"dotted"*), *marker* (*"none"*, *"square"*, *"circle"*, *"diamond"*, *"triangle"*), *negativeFill*, *labelTemplate*, *labelStyle*, *innerLabelStyle*, *showValues*, *labelPosition* (*"inside"*, *"outside"*, *"center"*), *pointFills* (array of colors), *showPercent* (pie, ring: the share) and *labelFormat* (number format of the labels, e.g. *"0.0"*; *""* for the format of the values).

valueAxis, *secondaryValueAxis*, *categoryAxis*: *{min, max, preferredMin, preferredMax, suffix, format, grid, gridColour, gridStyle, gridInFront, line, labels}*; *categoryAxis* also *dateLabels* (*""* for a text axis; *"all"*, *"firstlast"* or *"allwithquarters"* for a date axis) and *headerTitle* (the header cell of the data as the title of the axis).

Colors are given as *"RRGGBB"*, as the name of a color of the document, as *{name, tint, hex, definition, spot}*, as *""* (the color of the layout) or as *null* (none). A color name the document does not have needs *{name, hex}*; the color is created (see *warnings*).

---

set *(Object changes)* *Object*

Changes data and options together. *changes* is *{categories, series, options}*, each key optional, as for *setData()* and *setOptions()*. The data are set first, then the options, all in one undo step "Edit Chart". Another key gives errorCode *"unknown-parameter"*.

---

toGroup() *GroupObject*

Converts the chart object into a plain group with copies of its parts at the same place, in one undo step. Afterwards this object is no chart object (*isChart* is false). Returns the new group, or *null* if the conversion was refused (see *lastResult*).

Each method returns a result object and keeps it in *lastResult*: *success* (bool), *errorCode* and *error* (*""* on success, else a code and an English text) and further keys: *model* (the data and options after the change), *undo* (the text of the undo step), *warnings* (colors created, styles the document lacks), *group* (*toGroup()*). The methods throw no exception; *errorStatus* is set: *Ok* on success, *IsNotUsable* for *"not-licensed"*, *"not-active"*, *"layer-locked"*, *"chart-not-loaded"*, *"no-document"* and *"not-supported"*, *IsNotValid* for *"not-found"*, *"not-a-chart"* and *"no-page"*, *IsNotProcessed* for the other codes (*"bad-params"*, *"unknown-parameter"*, *"missing-parameter"*, *"failed"*).

Example:

```
  var doc = application.currentDocument;
  var chart = doc.currentPage.createChart({ x: "20mm", y: "30mm", type:
"column",
    categories: ["Q1", "Q2", "Q3"],
    series: [{ name: "2026", values: [12, 15, 9] }] });
  if (chart.isChart) {
    chart.setOptions({ title: "Sales", legend: false });
    chart.setData(null, [{ name: "2026", values: [12, 17, 11] }]);
  }
  else
    messageBox.information("Chart", chart.lastResult.error);
  // All charts of the page as bar charts
  var objs = doc.currentPage.objects();
  for (var i = 0; i < objs.length; i++) {
    var c = objs.at(i).type == ObjectType.Group ? objs.at(i).asChart() :
null;
    if (c)
      c.setOptions({ type: "bar" });
  }
```

Class: ObjectAlias

The class *ObjectAlias* extends the root class *Object* and adds other properties that are specific for this type of object.

Properties:

---

owner *Object* *Read*

Returns the original object of the alias object.

Methods:

`remove()` *bool*

Removes the alias object from the page and destroys it. Returns true in case of success.

Class: TextObjectAlias

The class ***TextObjectAlias*** extends the root class ***ObjectAlias*** and adds other properties that are specific for this type of object.

Properties:

`content` *TextContent* *Read*

Returns the text content of the alias object. See class ***TextContent***.

`textChain` *TextChain* *Read*

Returns the text chain the alias object belongs to. See class ***TextChain***.

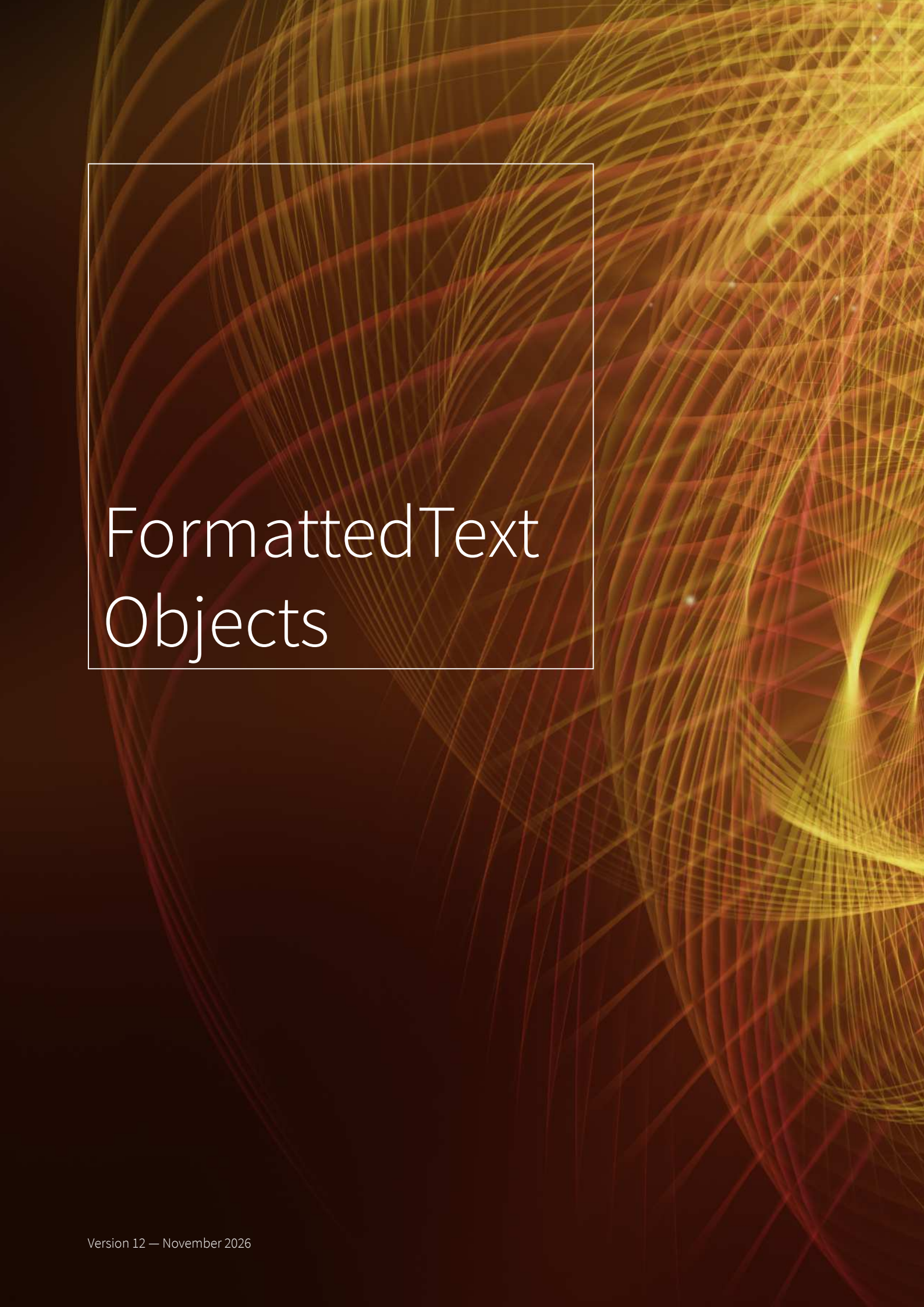
Class: PictureObjectAlias

The class ***PictureObjectAlias*** extends the class ***ObjectAlias*** and adds other properties that are specific for this type of object.

Properties:

`content` *PictureContent* *Read*

Returns the picture content of the alias object. See class ***PictureContent***.



Version 12 — November 2026

FormattedText Objects

Class: FormattedText

Text with colored letters, with various fonts, with chapters, indents, lists etc. is called “formatted text”. The object of the class ***FormattedText*** is a repository of formatted text and a tool for manipulating it. The class FormattedText is nested in the class ***TextObject***. Use the property ***TextObject.content*** and its property ***formattedText*** to access the text content of the text object.

Properties:

length	*int*	*Read*

The property contains a number of characters, including blank spaces and invisible characters (e.g. line breaks).

notesVisible	*bool*	*Read/Write*

Use this property to show or hide notes in the text.

Methods: (text creation)

plainText*(TextRange range)*	*String*
plainText*(int begin=0, int end=length)*	*String*

Get the plain text or its fragment without any formatting.

begin: Index of first character. The default value is 0.

end: Index of the position behind the last requested character. The default value is the length of the text.

range: The parameter (its properties ***begin*** and ***end***) defines the fragment.

Returns the required text or its fragment between positions `begin` (including) and `end` (excluding).

insertPlainText*(String text, int pos = 0, TypographicQuotesMode mode = TypographicQuotesOff)*	*TextRange*

Insert new plain text in the formatted text object.

text: Inserted text.

pos: The position where the text has to be inserted. If not specified the default value 0 is taken.

mode: Typographic quotes mode. ***TypographicQuotesOff*** (default) inserts the quotation marks as they are, ***TypographicQuotesOn*** replaces straight quotation marks by the quotation marks of the text language, ***TypographicQuotesPref*** does so if ***TextPreferences.typographicQuotationMarks*** is on. See the enumeration type ***TypographicQuotesMode***.

Returns a TextRange object which is related to the newly inserted text.

appendPlainText*(String text, TypographicQuotesMode mode = TypographicQuotesOff)*	*TextRange*

Append new plain text to the end of the existing formatted text.

text: Added text.

mode: Typographic quotes mode, see ***insertPlainText()***.

Returns a TextRange object which is related to the newly inserted text.

insertHtml(*String text, int pos=0*) *TextRange*

Insert HTML text in the formatted text object. The HTML is converted to formatted text as in the HTML import, e.g. ** becomes bold and *<i>* italic.

text: HTML text, e.g. *"Some bold text"*.

pos: The position where the text has to be inserted. The default value is 0.

Returns a TextRange object which is related to the newly inserted text, or *null* if nothing was inserted.

charAsString(*int pos=0*) *String*

charAsCode *(int pos=0) String*

Get the character located at the specified position in the formatted text.

pos: Position of the requested character in the formatted text.

Returns the required character as a string of one character. If the parameter pos is not a position in the text, charAsString() returns an empty string and charAsCode() the character with the code 0.

insertCharacter(*Char character*, *int pos=0*, *TypographicQuotesMode mode = TypographicQuotesOff*) *void*

Insert the character in the text.

character: Inserted character. The character can be set as a character ('a') or as a code (0x0061).

pos: The position where the text has to be inserted. The default value is 0.

object(*int pos=0*) *Object*

Get an object that is inserted to the formatted text at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns an object at the specified position or *null* if there is no object at this position. For a graphic object, the method returns *null* as well.

setObject(*Object obj, int pos=0*) *void*

Insert an object at the specified position in the formatted text.

obj: Object that is inserted at the specified position.

pos: Target position in the formatted text. The default value is 0.

After the call, the variable *obj* can no longer be used; get the inserted object with *object()*.

formattedText(*int begin=0, int end=length*) *FormattedText*

formattedText(*TextRange range*) *FormattedText*

Get fragment of formatted text.

begin: Index of the first letter of the requested text fragment. The default value is 0.

end: Index of the position behind the last requested character. The default value is 0.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

Returns the specified fragment of the formatted text.

insertFormattedText*(FormattedText text, int pos=0)* *TextRange*

Insert a new fragment of formatted text to this formatted text.

text: Inserted formatted text.

pos: Target position for the inserted formatted text.

Returns a text range object which represents the inserted text fragment inside the target formatted text.

annotation*(int pos=0)* *Annotation*

Get an annotation from a specified position in the formatted text.

pos: Position in the formatted text. The default value is 0.

Returns an object of the class Annotation if a annotation is found at the specified position. Otherwise it returns *null*. See the chapter "Class: Annotation" for detailed information about annotation object.

insertAnnotation*(String text, int begin=0, int end=len, RubyPosition pos)* *void*

insertAnnotation*(String text, TextRange range, RubyPosition pos)* *void*

Inserts a new annotation in the formatted text and assign it to the specified text fragment.

text: Text of the new annotation.

begin: Index of the first letter of the annotated text fragment. The default value is 0.

end: Index of the position behind the last character of the annotated text fragment. The default value is 0.

pos: Position of the inserted annotation. See the enumeration type *RubyPosition* for information about possible annotation positions.

range: The parameter (its properties begin and end) defines the fragment of annotated text.

pos: Position of the inserted annotation. See enumeration type *RubyPosition* for information about possible annotation positions.

comment*(int pos=0)* *Comment*

Get a comment from specified position of the formatted text.

pos: Position in the formatted text. The default value is 0.

Returns an object of the class Comment, if any comment is found at the specified position. Otherwise it returns *null*. See the chapter "Class: Comment" for detailed information about comment object.

insertComment*(Comment comment, int pos=0)* *TextRange*

Insert a new comment in the specified position in the formatted text.

comment: New comment that is inserted in the specified position in the formatted text. See the chapter "Class: Comment" for all information for creating a new comment object.

pos: Target position for the new inserted comment.

Returns a text range object that describes the new inserted comment inside the target formatted text.

note*(int pos=0)* *Note*

Get a note from specified position of the formatted text.

pos: Position in the formatted text. Default value is 0.

Returns an object of the class *Note* if a note is found at the specified position. Otherwise it returns *null*. See chapter “Class: Note” to get detailed information about note object.

insertNote*(NoteType noteType, int pos=0)* *Note*

Insert a new note to the specified position in the formatted text.

noteType: New note inserted to the specified position in the formatted text. See chapter “Class: Note” to get all information about the creation a new note object.

pos: Target position for new inserted note.

positionCorrection*(int pos=0)* *CharacterPositionCorrection*

Get the current value of the character position correction.

pos: Position of the inspected character in the formatted text.

Returns an object of the class *CharacterPositionCorrection* that contains the position correction. If no correction is applied, the returned object‘s horizontal and vertical correction is set to 0. See the chapter “Class: CharacterPositionCorrection” for information about correction values.

setPositionCorrection*(CharacterPositionCorrection cor, int pos=0)* *void*

Set the small horizontal and vertical shifts of particular characters to correct their position in the text.

cor: The parameters define the position correction. See the chapter “Class: CharacterPositionCorrection” for information about setting correction values.

pos: Position of the corrected character inside the formatted text.

characterFunction*(int pos=0)* *CharacterFunction*

Get a code that provides information about the function of the character located at the specified position in the formatted text.

pos: Position of the inspected character in the formatted text.

Returns one of the values of the enumeration type CharacterFunction. See the end of this chapter for information about all available function codes.

setCharacterFunction*(CharacterFunction function, int pos=0)* *TextRange*

Insert a special function character at a specified position in the formatted text.

function: Special character code. See the enumeration type *CharacterFunction* at the end of this chapter for complete information about the available function characters.

The value *CharacterFunction.NormalCharacter* doesn’t insert any character. The value is useful only for the function *characterFunction()*.

pos: Target position in the formatted text.

Returns a text range object that represents the inserted special character in the target formatted text.

variable*(int pos=0)* *Variable*

insertVariable*(Variable variable, int pos=0)* *TextRange*

variable() returns the variable at the position *pos* as a *Variable* object. If no variable stands at this position, the *type* of the returned object is an empty string.

insertVariable() inserts a variable at the position *pos* and returns a text range of the inserted variable. Create the variable with *new Variable(VariableType type)* or *new Variable(String typeName)*; see "Class: Variable". A *Variable* object without a variable (e.g. from *variable()* at a position without a variable) causes a script error.

```
var tr = ft.insertVariable(new Variable(VariableType.Date), 0);
```

removeRange*(int begin=0, int end=length)* *void*

removeRange*(TextRange range)* *void*

Remove text or its fragment from the formatted text.

begin: Index of first letter of removed text fragment. The default value is 0.

end: Index of the position behind the last removed character. The default value is the length of the text.

range: The parameter (its properties begin and end) defines the fragment of removed text.

clear() *void*

Remove the whole text from the text object.

copy() *FormattedText*

The command creates a new instance of the formatted text.

find*(int from, String pattern, bool ignoreCase=true, bool wholeWord=false)* *TextRange*

Find the first occurrence of the text *pattern* in the formatted text from the position *from*. Returns a TextRange object or *null* if the text is not found.

findRegEx*(int from, String pattern, bool ignoreCase=true)* *TextRange*

Find the first occurrence of the regular expression *pattern* in the formatted text from the position *from*. Returns a TextRange object or *null* if the text is not found.

findAndReplace*(int startPosition, String pattern, String replacement, bool ignoreCase = true, bool wholeWord = false)* *TextRange*

findAndReplaceRegEx*(int startPosition, String pattern, String replacement, bool ignoreCase = true)* *TextRange*

findAndReplaceAll*(int startPosition, String pattern, String replacement, bool ignoreCase = true, bool wholeWord = false)* *bool*

findAndReplaceAllRegEx*(int startPosition, String pattern, String replacement, bool ignoreCase = true)* *bool*

findAndReplace() and *findAndReplaceRegEx()* replace the first occurrence of *pattern* (a text or a regular expression) from the position *startPosition* by *replacement* and return the TextRange of the replaced text, or *null*. *findAndReplaceAll()* and *findAndReplaceAllRegEx()* replace all occurrences and return *true* unless the parameters are not valid.

removeStyleSheets*(int begin=0, int end=length, boolean withAttributes=false)* *void*

removeStyleSheets*(TextRange range, bool withAttributes=false)* *void*

The method removes style sheet setting from the text and converts the style sheets to a normal attribute setting. If the parameter 'withAttribute' set to true, the styles are removed completely without the conversion.

getAttributes(*AttributeGroup group, int pos=0*)	*Attributes*

Get all attributes of one group at the specified position, e.g. to copy the formatting of a text fragment to another one with ***setAttributes()***.

group: ***AttributeGroup.Character***, ***AttributeGroup.Paragraph*** or ***AttributeGroup.Layout***. See the enumeration type ***AttributeGroup***.

pos: Position in the formatted text. The default value is 0.

Returns an object of the class ***Attributes***.

setAttributes(*Attributes attrs, TextRange range*)	*void*

Apply the attributes of an ***Attributes*** object, e.g. from ***getAttributes()***, to the specified text fragment.

attrs: The attributes.

range: The parameter (its properties ***begin*** and ***end***) defines the fragment of affected text. Without it, nothing is changed.

Example:

```
var attrs = ft.getAttributes(AttributeGroup.Character, 0);
ft.setAttributes(attrs, new TextRange(10, 20));
```

Methods (character setting)

Most attributes have a getter and a setter. A getter without an argument reads the whole text, with ***pos*** the character at this position, with ***begin*** and ***end*** or a ***TextRange*** a fragment. If the value is not the same in the whole text or fragment, the getter returns ***null*** or a default value and sets ***errorStatus*** to ***IsNotValid***. A setter without a range changes the whole text. Paragraph and layout attributes apply to all paragraphs and layouts that belong (at least partially) to the fragment. This applies to all methods of the following sections.

font()	*FontName*
font(*int pos*)	*FontName*
font(*int begin, int end*)	*FontName*
font(*TextRange ran*)	*FontName*

Get the name of the font which is applied to the text at the specified position.

pos: Position inside the text.

Returns the name of the applied font or the value `null` when the error is occurred.

setFont(*FontName fontName, int begin=0, int end=length*)	*void*
setFont(*FontName fontName, TextRange range*)*void*	

Apply a font to specified text fragment.

fontName: Name of the font. E.g. "Arial-Regular", "Constantia-Bold".

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

setFont*(String fontFamilyName, String styleName, TextRange tr)*	*void*
setFont*(String fontFamilyName, String styleName, int begin=0, int end=len)*	*void*

Apply a font to a specified text fragment.

fontFamilyName: Name of the font family. e.g. “Arial”, “Constantia”.

styleName: Name of the font style. e.g. “Regular”, “Bold”, “Italic”, etc.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

fontSize*()*	*FontSize*
fontSize*(int pos)*	*FontSize*
fontSize*(int begin, int end)*	*FontSize*
fontSize*(TextRange ran)*	*FontSize*

Get the font size object as applied to the text at the specified position.

pos: Index of the character in the text. The default value is 0.

Returns the object FontSize of the font.

setFontSize*(FontSize size, TextRange range)*	*void*
setFontSize*(FontSize size, int begin=0, int end=length)*	*void*

Set the font size to be applied to the specified text range.

size: Required font object.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

languageByName*()*	*String*
languageByName*(int pos)*	*String*
languageByName*(int begin, int end)* *String*	
languageByName*(TextRange ran)*	*String*

Get the name of the language that is set for the text at a specified position.

pos: Position in the text. The default value is 0.

Returns the key of the language, e.g. *"german_DE_ref"*.

setLanguageByName*(String langName, int begin=0, int end=length)*	*void*
setLanguageByName*(String langName, TextRange range)*	*void*

Set name of language for the specified fragment of the text.

langName: The key of the language as returned by *languageByName()* (e.g. *"german_DE_ref"*; upper and lower case do not matter) or a BCP 47 tag (e.g. *"de-DE"*).

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties begin and end) defines the fragment of affected text.

language*()*	*Language*
language*(int pos)*	*Language*
language*(int begin, int end)*	*Language*
language*(TextRange range)*	*Language*

Get the id of the language that is set for the text at a specified position.

pos: Position in the text. The default value is 0.

Returns the language as a value of the enumeration type *Language*.

setLanguage*(Language lang, int begin=0, int end=length)*	*void*
setLanguage*(Language lang, TextRange range)*	*void*

Set id of language for the specified fragment of the text.

lang: ID of selected language.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties begin and end) defines the fragment of affected text.

brush*()*	*ColorBrush*
brush*(int pos)*	*ColorBrush*
brush*(int begin, int end)*	*ColorBrush*
brush*(TextRange range)*	*ColorBrush*

Get the brush (color) used for the text at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns the brush (color) used at the specified position.

setBrush*(ColorBrush brush, int begin=0, int end=length)*	*void*
setBrush*(ColorBrush brush, TextRange range)*	*void*
setBrush*(Color color, int begin=0, int end=length)*	*void*
setBrush*(Color color, TextRange range)*	*void*

Apply a new brush (color) to a specified text fragment.

brush: ColorBrush applied to the text fragment.

color: A value of the enumeration type *Color*, e.g. *Color.Blue*.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties begin and end) defines the fragment of affected text.

shade*()*	*float*
shade*(int pos)*	*float*
shade*(int begin, int end)*	*float*
shade*(TextRange range)*	*float*

Get value of shade level at the specified text position.

pos: Position in the formatted text. The default value is 0.

Returns level of shade. The value range is 0 – 100.

setShade*(float shade , int begin=0, int end=length)*	*void*
setShade*(float shade, TextRange range)*	*void*

Set level of shade to the specified text fragment.

shade: New value of shade level.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of position behind the last affected character. The default value is length of the text.

range: The parameter (its properties begin and end) defines the fragment of affected text.

opacity*()*	*float*
opacity*(int pos)*	*float*
opacity*(int begin, int end)*	*float*
opacity*(TextRange range)*	*float*

Get the opacity level at the specified text position.

pos: Position in the formatted text. The default value is 0.

Returns level of opacity. The value range is 0 – 100.

setOpacity*(float opacity, int begin=0, int end=length)*	*void*
setOpacity*(float opacity, TextRange range)*	*void*

Set the opacity level for the specified text fragment.

shade: New value of opacity level.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is length of the text.

range: The parameter (its properties begin and end) defines the fragment of the affected text.

outlineOptions*()*	*OutlineOptions*

outlineOptions*(int pos)*	*OutlineOptions*
outlineOptions*(int begin, int end)*	*OutlineOptions*
outlineOptions*(TextRange range)*	*OutlineOptions*

Get parameters for drawing the character outline. See the chapter “Class: OutlineOptions” to see all details about outline parameters.

pos: Position in the formatted text. The default value is 0.

Returns an object of the class OutlineOptions with outline parameters.

setOutlineOptions*(OutlineOptions options, int begin=0, int end=length)*	*void*
setOutlineOptions*(OutlineOptions options, TextRange range)*	*void*

Switch the outlining of characters on or off and set the parameters.

options: Object containing parameters for outlining. See chapter “Class: OutlineOptions” to see all the available parameters.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is length of the text.

range: The parameter (its properties begin and end) defines the fragment of affected text.

underline*()*	*UnderlineOptions*
underline*(int pos)*	*UnderlineOptions*
underline*(int begin, int end)*	*UnderlineOptions*
underline*(TextRange range)*	*UnderlineOptions*

Get parameters of the underline see is applied on the text at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns an object of class UnderlineOptions. See the chapter “Class: UnderlineOptions” for detailed information about underline parameters.

setUnderline*(UnderlineOptions options, int begin=0, int end=length)*	*void*
setUnderline*(UnderlineOptions options, TextRange range)*	*void*

Set the underline.

options: Detailed parameters of the underline. See the chapter “Class: UnderlineOptions” for detailed information about underline parameters.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default is the text length.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

strikethrough*()*	*StrikethroughOptions*
strikethrough*(int pos)*	*StrikethroughOptions*
strikethrough*(int begin, int end)*	*StrikethroughOptions*

strikethrough*(TextRange range)*	*StrikethroughOptions*

Get the strikethrough options.

pos: Position in the formatted text. The default value is 0.

setStrikethrough*(StrikethroughOptions options,* *int begin=0, int end=length)*	*void*
setStrikethrough*(StrikethroughOptions options, TextRange range)*	*void*

Set the strikethrough data.

options: Detailed parameters of the strikethrough. See the chapter “Class: StrikethroughOptions” for detailed information about strikethrough parameters.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default is the text length.

range: The parameter (its properties begin and end) defines the fragment of affected text.

characterPosition*()*	*CharacterPosition*
characterPosition(int pos)	*CharacterPosition*
characterPosition(int begin, int end) *CharacterPosition*	
characterPosition*(TextRange range)*	*CharacterPosition*

Get the current character position, which is used for text formatting at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns character position at the specified position. See enumeration type `CharacterPosition` at the end of this chapter to see all available values.

setCharacterPosition*(CharacterPosition type,* *int begin=0, int end=length)*	*void*
setCharacterPosition*(CharacterPosition type, TextRange range)*	*void*

Set new character position of the specified fragment of the formatted text.

type: New character position. See enumeration type CharacterPosition at the end of this chapter to see all available values.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default is the text length.

range: The parameter (its properties begin and end) defines the fragment of affected text.

characterUpperLowerCase*()*	*CharacterUpperLowerCase*
characterUpperLowerCase*(int pos)*	*CharacterUpperLowerCase*
characterUpperLowerCase*(int begin, int end)*	*CharacterUpperLowerCase*
characterUpperLowerCase*(TextRange range)* *CharacterUpperLowerCase*	

Get the current character upper/lower case that is used for text formatting at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns the character upper/lower case at the specified position. See the enumeration type CharacterUpperLowerCase at the end of this chapter to see all the available values.

setCharacterUpperLowerCase*(CharacterUpperLowerCase type, TextRange range)*	*void*
setCharacterUpperLowerCase*(CharacterUpperLowerCase type, int begin=0, int end=length)*	*void*

Set a new character upper/lower case of the specified fragment of the formatted text.

type: New character upper/lower case. See the enumeration type `CharacterUpperLowerCase` at the end of this chapter to see all the available values.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default is the text length.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

characterHeightScale*()*	*float*
characterHeightScale*(int pos)*	*float*
characterHeightScale*(int begin, int end)*	*float*
characterHeightScale*(TextRange range)*	*float*

Get the character height scale as a percentage of the original height at the specified position of the formatted text.

pos: Position in the formatted text. The default value is 0.

Returns percentage value of height scale.

setCharacterHeightScale*(float scale, TextRange range)*	*void*
setCharacterHeightScale*(float scale, int begin=0, int end=length)*	*void*

Set the character height scale of specified text fragment as a percentage of the original height.

scale: New height scale in percent.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

characterWidthScale*()*	*float*
characterWidthScale*(int pos)*	*float*
characterWidthScale*(int begin, int end)* *float*	
characterWidthScale*(TextRange range)* *float*	

Get character width scale as a percentage of the original width at the specified position of the formatted text. The default value is 0.

setCharacterWidthScale*(float scale, int begin=0, int end=length)*	*void*
setCharacterWidthScale*(float scale, TextRange range)*	*void*

Set character width scale of specified text fragment as a percentage of the original width.

scale: New height scale in percent.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

characterSpacing*()*	*float*
characterSpacing*(int pos)*	*float*
characterSpacing*(int begin, int end)*	*float*
characterSpacing*(TextRange range)*	*float*

Get character spacing extension as a percentage of the original spacing. The value 0 (i.e. 0%) means normal spacing. Positive values make character spacing bigger. Negative values reduce the spaces between characters.

pos: Position in the formatted text. The default value is 0.

Returns percentage value of current spacing.

setCharacterSpacing*(float space, int begin=0, int end=length)*	*void*
setCharacterSpacing*(float space, TextRange range)*	*void*

Set new character spacing that is applied to specified text fragment.

space: Character spacing in percent. The value 0 (i.e. 0%) means normal spacing. Positive values make character spacing bigger. Negative values reduce the spaces between characters.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

characterKerning*()*	*CharacterKerning*
characterKerning*(int pos)*	*CharacterKerning*
characterKerning*(int begin, int end)*	*CharacterKerning*
characterKerning*(TextRange range)*	*CharacterKerning*

Get character kerning object. The value 0 (i.e. 0%) means normal kerning. Positive values make character kerning bigger. Negative values reduce the spaces between characters.

pos: Position in the formatted text. The default value is 0.

Returns the kerning object.

setCharacterKerning*(CharacterKerning value, TextRange range)*	*void*
setCharacterKerning*(CharacterKerning value, int begin=0, int end=length)*	*void*

Set object of the character kerning for the specified text fragment.

value: New kerning object.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties begin and end) defines the fragment of affected text.

characterBaselineShift*()*	*float*
characterBaselineShift*(int pos)*	*float*
characterBaselineShift*(int begin, int end)*	*float*
characterBaselineShift*(TextRange range)*	*float*

Get current shift of character baseline at specified text position. Positive value shifts the characters upwards. Negative value means a shift downwards.

pos: Position in the formatted text. The default value is 0.

Returns current shift of character baseline in points (pt).

setCharacterBaselineShift*(float shift, int begin=0, int end=length)*	*void*
setCharacterBaselineShift *(float shift, TextRange range)*	*void*

Shift character baseline of specified text range upwards or downwards.

shift: Baseline shift in points (pt).

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of position behind the last affected character. The default value is length of the text. `range`: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

characterBaselineOrientation*()*	*CharacterBaselineOrientation*
characterBaselineOrientation*(int pos)*	*CharacterBaselineOrientation*
characterBaselineOrientation*(int begin, int end)*	*CharacterBaselineOrientation*
characterBaselineOrientation*(TextRange range)*	*CharacterBaselineOrientation*

Get current orientation of the character baseline at the specified text position.

pos: Position in the formatted text. The default value is 0.

Returns base line orientation. See enumeration type *CharacterBaselineOrientation* at the bottom of this chapter to see all available values.

setCharacterBaselineOrientation*(CharacterBaselineOrientation value, TextRange range)*	*void*
setCharacterBaselineOrientation*(CharacterBaselineOrientation value,int begin=0, int end=length)*	*void*

Set character baseline orientation of the specified text fragment.

value: Required baseline orientation. See the enumeration type *CharacterBaselineOrientation* at the bottom of this chapter to see all available values.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

characterRotation*()*	*CharacterRotation*
characterRotation*(int pos)*	*CharacterRotation*
characterRotation*(int begin, int end)*	*CharacterRotation*
characterRotation*(TextRange range)*	*CharacterRotation*

Get current character rotation at the specified text position.

pos: Position in the formatted text. The default value is 0.

Returns character rotation. See enumeration type ***CharacterRotation*** at the bottom of this chapter to see all available values.

setCharacterRotation*(CharacterRotation value, int begin=0, int end=length)*	*void*
setCharacterRotation*(CharacterRotation value, TextRange range)*	*void*

Set character rotation of the specified text fragment. Notice that baseline orientation remains unchanged. Only particular characters are rotated.

value: New character rotation. See enumeration type CharacterRotation at the bottom of this chapter to see all available values.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of position behind the last affected character. The default value is length of the text.

range: The parameter (its properties ***begin*** and ***end***) defines the fragment of affected text.

characterLigatures*()*	*bool*
characterLigatures*(int pos)*	*bool*
characterLigatures*(int begin, int end)*	*bool*
characterLigatures*(TextRange range)*	*bool*

Get information whether the ligatures are applied to the characters at the specified position in the text.

pos: Position in the formatted text. The default value is 0.

Returns ***true*** if the ligatures are used in the text. Otherwise it returns ***false***.

setCharacterLigatures*(bool value, int begin=0, int end=length)*	*void*
setCharacterLigatures*(bool value, TextRange range)*	*void*

Set or reset usage of ligatures to the specified text range.

value: Use value ***true*** or ***false*** to set respectively reset usage of ligatures in specified text range.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of position behind the last affected character. The default value is length of the text. *range*: The parameter (its properties ***begin*** and ***end***) defines the fragment of affected text.

characterNoKashida*()*	*bool*
characterNoKashida*(int pos)*	*bool*
characterNoKashida*(int begin, int end)*	*bool*

characterNoKashida*(TextRange range)* *bool*

Get information whether the characters at the specified position are kept out of kashida justification (option No Kashida). See ***kashidaLevel()***.

pos: Position in the formatted text.

Returns ***true*** if the characters are kept out of kashida justification. Otherwise it returns ***false***.

setCharacterNoKashida*(bool value, int begin=0, int end=length)* *void*

setCharacterNoKashida*(bool value, TextRange range)* *void*

Keep the characters of the specified text fragment out of kashida justification or allow kashidas again.

value: ***true*** keeps the characters out of kashida justification, ***false*** allows kashidas again.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties ***begin*** and ***end***) defines the fragment of affected text.

characterDigits*()* *Digits*

characterDigits*(int pos)* *Digits*

characterDigits*(int begin, int end)* *Digits*

characterDigits*(TextRange range)* *Digits*

Get the digits used at the specified position (option Digits): the digits of the numbers typed and of the list, note and page numbers.

pos: Position in the formatted text.

Returns a value of the enumeration type ***Digits***. ***Digits.Default*** uses the digits of the text language. A value that this version does not know comes back as its number.

setCharacterDigits*(Digits value, int begin=0, int end=length)* *void*

setCharacterDigits*(Digits value, TextRange range)* *void*

Set the digits for the specified text fragment. The characters of the text do not change, only the digits shown.

value: A value of the enumeration type ***Digits***, e.g. ***Digits.Western*** (0 1 2 3), ***Digits.ArabicIndic*** or ***Digits.Persian***. ***Digits.Default*** uses the digits of the text language, ***Digits.Native*** the native digits of the language.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties ***begin*** and ***end***) defines the fragment of affected text.

tablesIndexEntry*()* *TablesIndex*

tablesIndexEntry*(int pos)* *TablesIndex*

tablesIndexEntry*(int begin, int end)* *TablesIndex*

tablesIndexEntry*(TextRange range)* *TablesIndex*

setTablesIndexEntry*(TablesIndex value, int begin=0, int end=length)* *void*

setTablesIndexEntry*(TablesIndex value, TextRange range)*	*void*
tablesContentsEntry*()*	*TablesContents*
tablesContentsEntry*(int pos)*	*TablesContents*
tablesContentsEntry*(int begin, int end)*	*TablesContents*
tablesContentsEntry*(TextRange range)* *TablesContents*	
setTablesContentsEntry*(TablesContents val, int begin=0, int end=length)*	*void*
setTablesContentsEntry*(TablesContents value, TextRange range)*	*void*
tablesBibliographyEntry*()*	*TablesBibliography*
tablesBibliographyEntry*(int pos)*	*TablesBibliography*
tablesBibliographyEntry*(int begin, int end)*	*TablesBibliography*
tablesBibliographyEntry*(TextRange range)*	*TablesBibliography*
setTablesBibliographyEntry*(TablesBibliography value, int begin=0, int end=length)* *void*	
setTablesBibliographyEntry*(TablesBibliography value, TextRange range)*	*void*
tablesPictureEntry*()*	*TablesPicture*
tablesPictureEntry*(int pos)*	*TablesPicture*
tablesPictureEntry*(int begin, int end)*	*TablesPicture*
tablesPictureEntry*(TextRange range)*	*TablesPicture*
setTablesPictureEntry*(TablesPicture value, int begin=0, int end=length)*	*void*
setTablesPictureEntry*(TablesPicture value, TextRange range)*	*void*
tablesAbbreviationEntry*()*	*TablesAbbreviation*
tablesAbbreviationEntry*(int pos)*	*TablesAbbreviation*
tablesAbbreviationEntry*(int begin, int end)*	*TablesAbbreviation*
tablesAbbreviationEntry*(TextRange range)*	*TablesAbbreviation*
setTablesAbbreviationEntry*(TablesAbbreviation value, int begin=0, int end=length)* *void*	
setTablesAbbreviationEntry*(TablesAbbreviation value, TextRange range)*	*void*
tablesRunningTitleEntry*()*	*TablesRunningTitle*
tablesRunningTitleEntry*(int pos)*	*TablesRunningTitle*
tablesRunningTitleEntry*(int begin, int end)*	*TablesRunningTitle*
tablesRunningTitleEntry*(TextRange range)*	*TablesRunningTitle*
setTablesRunningTitleEntry*(TablesRunningTitle value, int begin=0, int end=length)* *void*	
setTablesRunningTitleEntry*(TablesRunningTitle value, TextRange range)*	*void*
openType*()*	*OpenType*
openType*(int pos)*	*OpenType*

openType*(int begin, int end)*	*OpenType*
openType*(TextRange range)*	*OpenType*
setOpenType*(OpenType value, int begin=0, int end=length)*	*void*
setOpenType*(OpenType value, TextRange range)*	*void*
link*()*	*Link*
link*(int pos)*	*Link*
link*(int begin, int end)*	*Link*
link*(TextRange range)*	*Link*
setLink*(Link link, int begin=0, int end=length)*	*void*
setLink*(Link link, TextRange range)*	*void*
characterBackground*()*	*CharacterBackground*
characterBackground*(int pos)*	*CharacterBackground*
characterBackground*(int begin, int end)*	*CharacterBackground*
characterBackground*(TextRange range)*	*CharacterBackground*

Get parameters that are applied to draw the character background at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns an object that keeps the parameters of the background. See the chapter "Class: CharacterBackground" to see all the parameters that can be used to draw the background.

setCharacterBackground*(CharacterBackground chb, int begin=0, int end=length)*	*void*
setCharacterBackground*(CharacterBackground chb, TextRange range)*	*void*

Show or hide the character background of the specified text fragment and set its parameters.

chb: Parameters of the character background. See the chapter "Class: CharacterBackground" to see all the parameters that can be used to draw the background.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of position behind the last affected character. The default value is length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

characterFrame*()*	*CharacterFrame*
characterFrame*(int pos)*	*CharacterFrame*
characterFrame*(int begin, int end)*	*CharacterFrame*
characterFrame*(TextRange range)*	*CharacterFrame*

Get the parameters that are applied to draw the frame around the text at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns an object that keeps the parameters of the text frame. See the chapter "Class: CharacterFrame" to see all the parameters that can be used to draw the text frame.

setCharacterFrame*(CharacterFrame chb, TextRange range)*	*void*
setCharacterFrame*(CharacterFrame chb, int begin=0, int end=length)*	*void*

Show or hide the text frame of the specified text fragment and set its parameters.

chb: Parameters of the text frame. See the chapter "Class: CharacterFrame" to see all the parameters that can be used to draw the text frame.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of position behind the last affected character. The default value is length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

characterRule*()*	*CharacterRule*
characterRule*(int pos)*	*CharacterRule*
characterRule*(int begin, int end)*	*CharacterRule*
characterRule*(TextRange range)*	*CharacterRule*

Get parameters that are applied to draw the character rule at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns an object that keeps the parameters of the character rule. See the chapter "Class: CharacterRule" to see all the parameters that can be used to draw the rule.

setCharacterRule*(CharacterRule chr, TextRange range)*	*void*
setCharacterRule*(CharacterRule chr, int begin=0, int end=length)*	*void*

Show or hide the character rule in the specified text fragment and set its parameters.

chr: Parameters of the character rule. See the chapter "Class: CharacterRule" to see all particular parameters that can be used to draw the rule.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of position behind the last affected character. The default value is length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

characterStyleSheet*()*	*CharacterStyleSheet*
characterStyleSheet*(int pos)*	*CharacterStyleSheet*
characterStyleSheet*(int begin, int end)*	*CharacterStyleSheet*

Get an object of the character style sheet that is applied to the formatted text at the specified position. No parameter means the whole text.

Returns an object of the character style sheet applied. If there no style sheet is applied, the method returns *null*.

setCharacterStyleSheet*(CharacterStyleSheet style, TextRange range)*	*void*
setCharacterStyleSheet*(CharacterStyleSheet style, int begin=0, int end=length)*	*void*

Apply a character style sheet to the specified text fragment.

style: Name of the style sheet to be applied.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text..

characterStyleSheetByName*()*	*String*
characterStyleSheetByName*(int pos)*	*String*
characterStyleSheetByName*(int begin, int end)*	*String*

Get the name of the character style sheet that is applied to the formatted text at the specified position. . No parameter means the whole text.

Returns the object name of the character style sheet applied. If there is no style sheet applied, the method returns *null*.

setCharacterStyleSheetByName*(String style, int begin=0, int end=length)*	*void*
setCharacterStyleSheetByName*(String style, TextRange range)*	*void*

Apply character style sheet to the specified text fragment.

style: Name of the style sheet to be applied.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

Methods (paragraph setting)

textAlignment*()*	*ParagraphAlignment*
textAlignment*(int pos)*	*ParagraphAlignment*
textAlignment*(int begin, int end)*	*ParagraphAlignment*
textAlignment*(TextRange range)*	*ParagraphAlignment*

Get paragraph alignment at the specified position in the formatted text.

pos: Position in the formatted text. The default value is 0.

Returns the paragraph alignment code. See enumeration type *ParagraphAlignment* at the end of this chapter to see all available alignments.

setTextAlignment*(ParagraphAlignment align, TextRange range)*	*void*
setTextAlignment*(ParagraphAlignment align, int begin=0, int end=length)*	*void*

Set the paragraph alignment of the specified text fragment.

align: New paragraph alignment. See enumeration type *ParagraphAlignment* at the end of this chapter to see all available alignments.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is length of the text.

range: The parameter (its properties ***begin*** and ***end***) defines the fragment of affected text.

baselineGridType*()*	*BaselineGridType*
baselineGridType*(int pos)*	*BaselineGridType*
baselineGridType*(int begin, int end)*	*BaselineGridType*
baselineGridType*(TextRange range)*	*BaselineGridType*
setBaselineGridType*(BaselineGridType type, int begin=0, int end=len)*	*void*
setBaselineGridType*(BaselineGridType type, TextRange range)*	*void*
paragraphLineSpacing*()*	*String*
paragraphLineSpacing*(int pos)*	*String*
paragraphLineSpacing*(int begin, int end)*	*String*
paragraphLineSpacing*(TextRange range)*	*String*
setParagraphLineSpacing*(String value, int begin=0, int end=len)*	*void*
setParagraphLineSpacing*(String value, TextRange range)*	*void*
spaceToPreviousParagraph*()*	*String*
spaceToPreviousParagraph*(int pos)*	*String*
spaceToPreviousParagraph*(int begin, int end)*	*String*
spaceToPreviousParagraph*(TextRange range)*	*String*
setSpaceToPreviousParagraph*(String value, int begin=0, int end=len)*	*void*
setSpaceToPreviousParagraph*(String value, TextRange range)*	*void*
spaceToNextParagraph*()* *String*	
spaceToNextParagraph*(int pos)*	*String*
spaceToNextParagraph*(int begin, int end)* *String*	
spaceToNextParagraph*(TextRange range)* *String*	
setSpaceToNextParagraph*(String value, int begin=0, int end=len)*	*void*
setSpaceToNextParagraph*(String value, TextRange range)*	*void*
automaticWordSpacing*()* *AutomaticWordSpacing*	
automaticWordSpacing*(int pos)*	*AutomaticWordSpacing*
automaticWordSpacing*(int begin, int end)* *AutomaticWordSpacing*	
automaticWordSpacing*(TextRange range)* *AutomaticWordSpacing*	
setAutomaticWordSpacing*(AutomaticWordSpacing value, int begin=0, int end=len)* *void*	
setAutomaticWordSpacing*(AutomaticWordSpacing value, TextRange range)* *void*	
automaticCharacterSpacing*()*	*AutomaticCharacterSpacing*

automaticCharacterSpacing*(int pos)*	*AutomaticCharacterSpacing*
automaticCharacterSpacing*(int begin, int end)*	*AutomaticCharacterSpacing*
automaticCharacterSpacing*(TextRange range)*	*AutomaticCharacterSpacing*
setAutomaticCharacterSpacing*(AutomaticCharacterSpacing value, int begin=0, int end=len)*	*void*
setAutomaticCharacterSpacing*(AutomaticCharacterSpacing value, TextRange range)*	*void*
automaticCharacterWidth*()*	*AutomaticCharacterWidth*
automaticCharacterWidth*(int pos)*	*AutomaticCharacterWidth*
automaticCharacterWidth*(int begin, int end)*	*AutomaticCharacterWidth*
automaticCharacterWidth*(TextRange range)*	*AutomaticCharacterWidth*
setAutomaticCharacterWidth*(AutomaticCharacterWidth value, int begin=0, int end=len)*	*void*
setAutomaticCharacterWidth*(AutomaticCharacterWidth value, TextRange range)*	*void*
paragraphLeftIndent*()*	*String*
paragraphLeftIndent*(int pos)*	*String*
paragraphLeftIndent*(int begin, int end)* *String*	
paragraphLeftIndent*(TextRange range)* *String*	
setParagraphLeftIndent*(String value, int begin=0, int end=len)*	*void*
setParagraphLeftIndent*(String value, TextRange range)*	*void*
paragraphRightIndent*()*	*String*
paragraphRightIndent*(int pos)*	*String*
paragraphRightIndent*(int begin, int end)* *String*	
paragraphRightIndent*(TextRange range)* *String*	
setParagraphRightIndent*(String value, int begin=0, int end=len)*	*void*
setParagraphRightIndent*(String value, TextRange range)*	*void*
paragraphIndent*()*	*String*
paragraphIndent*(int pos)*	*String*
paragraphIndent*(int begin, int end)* *String*	
paragraphIndent*(TextRange range)*	*String*
setParagraphIndent *(String value, int begin=0, int end=len)*	*void*
setParagraphIndent *(String value, TextRange range)*	*void*
paragraphTabs*()*	*Tabulators*
paragraphTabs*(int pos)*	*Tabulators*

paragraphTabs*(int begin, int end)* *Tabulators*	
paragraphTabs*(TextRange range)*	*Tabulators*
setParagraphTabs *(Tabulators tabs, int begin=0, int end=len)*	*void*
setParagraphTabs *(Tabulators tabs, TextRange range)*	*void*
dropCaps*()*	*DropCaps*
dropCaps*(int pos)*	*DropCaps*
dropCaps*(int begin, int end)*	*DropCaps*
dropCaps*(TextRange range)*	*DropCaps*
setDropCaps*(DropCaps data, int begin=0, int end=len)*	*void*
setDropCaps*(DropCaps data, TextRange range)*	*void*
hyphenation*()*	*Hyphenation*
hyphenation*(int pos)*	*Hyphenation*
hyphenation*(int begin, int end)*	*Hyphenation*
hyphenation*(TextRange range)*	*Hyphenation*
setHyphenation*(Hyphenation hyp, int begin=0, int end=len)*	*void*
setHyphenation*(Hyphenation hyp, TextRange range)*	*void*
enumerations*()*	*Enumerations*
enumerations*(int pos)*	*Enumerations*
enumerations*(int begin, int end)*	*Enumerations*
enumerations*(TextRange range)*	*Enumerations*
setEnumerations*(Enumerations enum, int begin=0, int end=len)*	*void*
setEnumerations*(Enumerations enum, TextRange range)*	*void*
writingDirection*()*	*WritingDirection*
writingDirection*(int pos)*	*WritingDirection*
writingDirection*(int begin, int end)* *WritingDirection*	
writingDirection*(TextRange range)* *WritingDirection*	
setWritingDirection*(WritingDirection direction, TextRange range)*	*void*
setWritingDirection*(WritingDirection direction, int begin=0, int end=len)*	*void*
opticalAlignment*()*	*OpticalAlignment*
opticalAlignment*(int pos)*	*OpticalAlignment*
opticalAlignment*(int begin, int end)* *OpticalAlignment*	
opticalAlignment*(TextRange range)* *OpticalAlignment*	

setOpticalAlignment(*OpticalAlignment alignment, TextRange range*)	*void*
setOpticalAlignment(*OpticalAlignment alignment, int begin=0, int end=len*)	*void*
kashidaLevel()	*KashidaLevel*
kashidaLevel(*int pos*)	*KashidaLevel*
kashidaLevel(*int begin, int end*)	*KashidaLevel*
kashidaLevel(*TextRange range*)	*KashidaLevel*

Get the kashida level of the paragraph at the specified position (option Insert Kashida). Kashida justification stretches Arabic words with tatweels instead of widening the spaces only.

pos: Position in the formatted text.

Returns a value of the enumeration type *KashidaLevel*: *None*, *Short*, *Medium*, *Long* or *Stylistic*. A level that this version does not know comes back as its number.

setKashidaLevel(*KashidaLevel level=KashidaLevel.None, int begin=0, int end=length*)	*void*
setKashidaLevel(*KashidaLevel level, TextRange range*)	*void*

Set the kashida level of all paragraphs that belong (at least partially) to the specified text fragment. The kashida rules of each paragraph stay as they are.

level: New kashida level. See the enumeration type *KashidaLevel*.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

kashidaRules()	*String*
kashidaRules(*int pos*)	*String*
kashidaRules(*int begin, int end*)	*String*
kashidaRules(*TextRange range*)	*String*

Get the kashida rules of the paragraph at the specified position (option Kashida Rules).

pos: Position in the formatted text.

Returns *"naskh"*, *"simple"* or an empty string for automatic (the rules follow the font).

setKashidaRules(*String rules="", int begin=0, int end=length*)	*void*
setKashidaRules(*String rules, TextRange range*)	*void*

Set the kashida rules of all paragraphs that belong (at least partially) to the specified text fragment. The kashida level of each paragraph stays as it is.

rules: *"naskh"*, *"simple"* or *""* for automatic. Other names are kept as they are.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties ***begin*** and ***end***) defines the fragment of affected text.

Example:

```
// ft: the formatted text of a text object with Arabic text
ft.setTextAlignment(ParagraphAlignment.Justified);
ft.setKashidaLevel(KashidaLevel.Medium);
ft.setKashidaRules("naskh");
ft.setCharacterNoKashida(true, 0, 3);    // no kashidas in the first word
ft.setCharacterDigits(Digits.ArabicIndic);
var level = ft.kashidaLevel(0);    // KashidaLevel.Medium
```

supressLineNumbering*()*	*bool*
supressLineNumbering*(int pos)*	*bool*
supressLineNumbering*(int begin, int end)*	*bool*
supressLineNumbering*(TextRange range)*	*bool*
setSupressLineNumbering*(bool on, TextRange range)*	*void*
setSupressLineNumbering*(bool on=true, int begin=0, int end=len)*	*void*

Get/Set whether the lines of the paragraphs are left out of the line numbering.

on: ***true*** leaves the lines out of the line numbering, ***false*** numbers them.

paragraphFrame*()*	*ParagraphFrame*
paragraphFrame*(int pos)*	*ParagraphFrame*
paragraphFrame*(int begin, int end)*	*ParagraphFrame*
paragraphFrame*(TextRange range)*	*ParagraphFrame*
setParagraphFrame*(ParagraphFrame pf, TextRange range)*	*void*
setParagraphFrame*(ParagraphFrame pf, int begin=0, int end=len)*	*void*
paragraphBackground*()*	*ParagraphBackground*
paragraphBackground*(int pos)*	*ParagraphBackground*
paragraphBackground*(int begin, int end)*	*ParagraphBackground*
paragraphBackground*(TextRange range)*	*ParagraphBackground*
setParagraphBackground*(ParagraphBackground pb, TextRange range)*	*void*
setParagraphBackground*(ParagraphBackground pb, int begin=0, int end=len)*	*void*
paragraphRuleAbove*()*	*ParagraphRule*
paragraphRuleAbove*(int pos)*	*ParagraphRule*
paragraphRuleAbove*(int begin, int end)*	*ParagraphRule*
paragraphRuleAbove*(TextRange range)*	*ParagraphRule*
setParagraphRuleAbove*(ParagraphRule pr, TextRange range)*	*void*
setParagraphRuleAbove*(ParagraphRule pr, int begin=0, int end=len)*	*void*

paragraphRuleBelow`()`	*ParagraphRule*
paragraphRuleBelow`(int pos)`	*ParagraphRule*
paragraphRuleBelow`(int begin, int end)` *ParagraphRule*	
paragraphRuleBelow`(TextRange range)` *ParagraphRule*	
setParagraphRuleBelow`(ParagraphRule pr, TextRange range)`	*void*
setParagraphRuleBelow`(ParagraphRule pr, int begin=0, int end=len)`	*void*
widowsOrphans`()`	*WidowsOrphans*
widowsOrphans`(int pos)`	*WidowsOrphans*
widowsOrphans`(int begin, int end)` *WidowsOrphans*	
widowsOrphans`(TextRange range)`	*WidowsOrphans*
setWidowsOrphans`(WidowsOrphans value, TextRange range)`	*void*
setWidowsOrphans`(WidowsOrphans value, int begin=0, int end=length)`	*void*
keepParagraphTogether`()`	*bool*
keepParagraphTogether`(int pos)`	*bool*
keepParagraphTogether`(int begin, int end)` *bool*	
keepParagraphTogether`(TextRange range)` *bool*	
setKeepParagraphTogether`(bool value, TextRange range)`	*void*
setKeepParagraphTogether`(bool value, int begin=0, int end=length)`	*void*
paragraphStyleSheet`()`	*ParagraphStyleSheet*
paragraphStyleSheet`(int pos)`	*ParagraphStyleSheet*
paragraphStyleSheet`(int begin, int end)` *ParagraphStyleSheet*	

Get an object of the paragraph style sheet that is applied to the formatted text at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns an object of the paragraph style sheet applied. If there is no style sheet applied, the method returns *null*.

setParagraphStyleSheet`(ParagraphStyleSheet style, int begin=0, int end=length)` *void*	
setParagraphStyleSheet`(ParagraphStyleSheet style, TextRange range)`	*void*

Apply a paragraph style sheet to all paragraphs that belong (at least partially) to the specified text fragment.

style: Name of the style sheet to be applied.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

paragraphStyleSheetByName`()`	*String*
paragraphStyleSheetByName`(int pos)`	*String*
paragraphStyleSheetByName`(int begin, int end)`	*String*

Get the name of the paragraph style sheet that is applied to the formatted text at the specified position.

pos: Position in the formatted text. The default value is 0.

Returns the name of applied paragraph style sheet. If no style sheet is applied, the method returns *null*.

setParagraphStyleSheetByName`(String style, int begin=0, int end=length)`	*void*
setParagraphStyleSheetByName`(String style, TextRange range)`	*void*

Apply a paragraph style sheet to all paragraphs that belong (at least partially) to the specified text fragment.

style: Name of the style sheet to be applied.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties *begin* and *end*) defines the fragment of affected text.

Methods (layout settings)

layoutLineOrientation`()`	*LayoutLineDirection*
layoutLineOrientation`(int pos)`	*LayoutLineDirection*
layoutLineOrientation`(int begin, int end)`	*LayoutLineDirection*
layoutLineOrientation`(TextRange range)`	*LayoutLineDirection*
setLayoutLineOrientation`(LayoutLineDirection direction, TextRange range)`	*void*
setLayoutLineOrientation`(LayoutLineDirection direction, int begin=0, int end=len)`	*void*
layoutLineJustification`()`	*LayoutLineJustification*
layoutLineJustification`(int pos)`	*LayoutLineJustification*
layoutLineJustification`(int begin, int end)`	*LayoutLineJustification*
layoutLineJustification`(TextRange range)`	*LayoutLineJustification*
setLayoutLineJustification`(LayoutLineJustification justif, TextRange range)`	*void*
setLayoutLineJustification`(LayoutLineJustification justif, int begin=0, int end=len)`	*void*
justifiedSpaceBetweenLinesMaximum`()`	*String*
justifiedSpaceBetweenLinesMaximum`(int pos)`	*String*
justifiedSpaceBetweenLinesMaximum`(int begin, int end)`	*String*
justifiedSpaceBetweenLinesMaximum`(TextRange range)`	*String*
setJustifiedSpaceBetweenLinesMaximum`(String value, TextRange range)`	*void*

setJustifiedSpaceBetweenLinesMaximum*(String value, int begin=0, int end=len)*	*void*
paragraphSpaceMaximum*()*	*String*
paragraphSpaceMaximum*(int pos)* *String*	
paragraphSpaceMaximum*(int begin, int end)* *String*	
paragraphSpaceMaximum*(TextRange range)* *String*	
setParagraphSpaceMaximum*(String value, TextRange range)*	*void*
setParagraphSpaceMaximum*(String value, int begin=0, int end=len)*	*void*
spaceToPreviousLayout*()* *String*	
spaceToPreviousLayout*(int pos)*	*String*
spaceToPreviousLayout*(int begin, int end)* *String*	
spaceToPreviousLayout*(TextRange range)* *String*	
setSpaceToPreviousLayout*(String value, TextRange range)*	*void*
setSpaceToPreviousLayout*(String value, int begin=0, int end=len)*	*void*
spaceToNextLayout*()*	*String*
spaceToNextLayout*(int pos)*	*String*
spaceToNextLayout*(int begin, int end)* *String*	
spaceToNextLayout*(TextRange range)* *String*	
setSpaceToNextLayout*(String value, TextRange range)*	*void*
setSpaceToNextLayout*(String value, int begin=0, int end=len)*	*void*
layoutColumns*()*	*LayoutColumnsInfo*
layoutColumns*(int pos)*	*LayoutColumnsInfo*
layoutColumns*(int begin, int end)* *LayoutColumnsInfo*	
layoutColumns*(TextRange range)*	*LayoutColumnsInfo*
setLayoutColumns*(LayoutColumnsInfo columns, TextRange range)*	*void*
setLayoutColumns*(LayoutColumnsInfo columns, int begin=0, int end=len)*	*void*
layoutFootnotes*()*	*LayoutFootnotes*
layoutFootnotes*(int pos)*	*LayoutFootnotes*
layoutFootnotes*(int begin, int end)* *LayoutFootnotes*	
layoutFootnotes*(TextRange range)*	*LayoutFootnotes*
setLayoutFootnotes*(LayoutFootnotes fn, TextRange range)*	*void*
setLayoutFootnotes*(LayoutFootnotes fn, int begin=0, int end=len)*	*void*
layoutStyleSheet*()*	*LayoutStyleSheet*
layoutStyleSheet*(int pos)*	*LayoutStyleSheet*

layoutStyleSheet*(int begin, int end)* *LayoutStyleSheet*

Get an object of a layout style sheet that is applied to the formatted text at the specified position. No parameter means the whole text.

Returns an object of the layout style sheet applied. If no style sheet is applied, the method returns ***null***.

setLayoutStyleSheet*(LayoutStyleSheet style, int begin=0, int end=length)* *void*

setLayoutStyleSheet*(LayoutStyleSheet style, TextRange range)* *void*

Apply a layout style sheet to all layouts that belong (at least partially) to the specified text fragment.

style: Name of the style sheet to be applied.

begin: Index of the first letter of the affected text fragment. The default value is 0.

end: Index of the position behind the last affected character. The default value is the length of the text.

range: The parameter (its properties ***begin*** and ***end***) defines the fragment of affected text.

layoutStyleSheetByName*()* *String*

layoutStyleSheetByName*(int pos)* *String*

layoutStyleSheetByName*(int begin, int end)* *String*

Get the name of the layout style sheet that is applied to the formatted text at the specified position. No parameter means the whole text.

Returns the name of the layout style sheet applied. If no style sheet is applied, the method returns ***null***.

setLayoutStyleSheetByName*(String style, TextRange range)* *void*

setLayoutStyleSheetByName*(String style, int begin=0, int end=length)* *void*

Apply layout style sheet to all layouts that belong (at least partially) to the specified text fragment.

style: Name of the style sheet to be applied.

begin: Index of the first letter of the affected text fragment. Default value is 0.

end: Index of the position behind the last affected character. The default value is length of the text.

range: The parameter (its properties ***begin*** and ***end***) defines the fragment of affected text.

Methods (chapter setting)

chapterName*()* *String*

chapterName*(int pos)* *String*

chapterName*(int begin=0, int end=length)* *String*

chapterName*(TextRange range)* *String*

setChapterName*(String name, TextRange range)* *void*

setChapterName*(String name, int begin=0, int end=length)* *void*

Methods (Story setting)

storyEndnotes*()* *TextEndnotes*

storyEndnotes*(int pos)* *TextEndnotes*

storyEndnotes*(int begin=0, int end=length) TextEndnotes*	
storyEndnotes*(TextRange range)*	*TextEndnotes*
setStoryEndnotes*(TextEndnotes value, TextRange range)*	*void*
setStoryEndnotes*(TextEndnotes value, int begin=0, int end=length)*	*void*
storyFootnotes*()*	*TextFootnotes*
storyFootnotes*(int pos)*	*TextFootnotes*
storyFootnotes*(int begin=0, int end=length)*	*TextFootnotes*
storyFootnotes*(TextRange range)*	*TextFootnotes*
setStoryFootnotes*(TextFootnotes value, TextRange range)*	*void*
setStoryFootnotes*(TextFootnotes value, int begin=0, int end=length)void*	
storyLineCounterData*()*	*LineNumbers*
storyLineCounterData*(int pos)*	*LineNumbers*
storyLineCounterData*(int begin=0, int end=length)*	*LineNumbers*
storyLineCounterData*(TextRange range) LineNumbers*	
setStoryLineCounterData*(LineNumbers data, TextRange range)*	*void*
setStoryLineCounterData*(LineNumbers data, int begin=0, int end=length)*	*void*

Methods (lists)

paragraphAtCharacterPosition*(int pos=0) TextRange*

Get a paragraph, part of the formatted text, which contains a specified text position.

pos: Position in the formatted text. The default value is 0.

Returns a text range which represents the appropriate paragraph.

chapters*(TextRange range)*	*TextRanges*

chapters*(int begin=0, int end=length) TextRanges*

Returns a collection of the object TextRanges. Each TextRange object defines one chapter that is contained in the text in the range ***range***. If no range input parameter is entered all text is used.

layouts*(TextRange range)*	*TextRanges*

layouts*(int begin=0, int end=length) TextRanges*

Returns a collection of the object TextRanges. Each TextRange object defines one layout that is contained in the text in the range ***range***. If no range input parameter is entered all text is used.

paragraphs*(TextRange range)*	*TextRanges*

paragraphs*(int begin=0, int end=length) TextRanges*

Returns a collection of the object TextRanges. Each TextRange object defines one paragraph that is contained in the text in the range ***range.*** If no ***range*** input parameter is entered all text is used.

spans*(TextRange range)*	*TextRanges*

spans*(int begin=0, int end=length)* *TextRanges*

Returns a collection of the object TextRanges. Each TextRange object defines one span that is contained in the text in the range ***range.*** If no ***range*** input parameter is entered all text is used.

endnotesRanges*(TextRange range)* *TextRanges*

endnotesRanges*(int begin=0, int end=length)* *TextRanges*

Returns a collection of the object TextRanges. Each TextRange object defines one endnotes ranges that is contained in the text in the range ***range.*** If no ***range*** input parameter is entered all text is used.

story*()* *TextRange*

Returns a text range of the whole text of the story.

Examples:

```
var doc = application.currentDocument;
if (doc) {
    // Create new text object
    var objT = doc.objects().create(ObjectType.Text);
    if (objT) {
        with (objT) {
            // Configure its shape to be a rectangle
            form = FrameForm.Rectangle;
            pageGeometryBounds = ["15mm", "35.5pt", "180mm", "750pt"];
        }
        // Insert the text object in the current page
        doc.currentPage.add(objT);
    }
    // Access the content of the text object
    with (objT.content.formattedText) {
        // Add some text
        var trLine1 = appendPlainText("This is an example text, which is
appended to a formatted text of TextObject.\n");
        var trLine2 = appendPlainText("After a new line here is another
line of this text.\n");
        appendPlainText("\n");
        // Apply another font, its size and color
        setFont("Courier New-Bold", trLine1);
        setFontSize(new FontSize(20), trLine1.begin, trLine1.end - 20);
        setBrush(Color.Blue, trLine1.begin + 20, trLine1.end - 20);
        // Automatic underline: no style parameters are necessary
        setUnderline(new UnderlineOptions(FontStyleLineType.Automatic),
trLine2.begin, trLine2.begin + 11);
        // Manual strikethrough: prepare parameters and apply them to a
text
        var optSt = new StrikethroughOptions(FontStyleLineType.Manual);
        with (optSt) {
            brush = new ColorBrush(Color.Red);
            shade = 100;
            opacity = 60;
            lineWidth = "1.5pt";
            lineStyle = "dashed-3-2";
```

```
            offset = "-3pt";
            rangeType = FontStyleLineRangeType.Words;
        }
        setStrikethrough(optSt, trLine2.begin + 12, trLine2.begin + 32);
        // Glyphs
        var trGl = appendPlainText("1 x 1012 ... No Glyph Superscript
Subscript\n");
        appendPlainText("\n");
        setFontSize(new FontSize(20), trGl);
        setCharacterPosition(CharacterPosition.Superscript, trGl.begin + 6,
            trGl.begin + 8);
        setCharacterPosition(CharacterPosition.Superscript, trGl.begin +
22,
            trGl.begin + 33);
        setCharacterPosition(CharacterPosition.Subscript, trGl.begin + 34,
            trGl.begin + 43);
        setCharacterUpperLowerCase(CharacterUpperLowerCase.Caps, trGl.begin
+ 34,              trGl.begin + 43);
        // Outlines
        appendPlainText("Outlined Text:\n");
        var trOl = appendPlainText("STYLE 1 / STYLE 2 / STYLE 3\n");
        appendPlainText("\n");
        setFontSize(new FontSize(35), trOl);
        var optOl = new OutlineOptions(Color.Green, 100, 100, "2pt",
"solid", CharacterOutlineJoinStyle.MiterJoin);
        setOutlineOptions(optOl, new TextRange(trOl.begin, trOl.begin +
7));

        optOl.brush = new ColorBrush(Color.Blue);
        optOl.joinStyle = CharacterOutlineJoinStyle.BevelJoin;
    optOl.active = true;
        setOutlineOptions(optOl, new TextRange(trOl.begin + 10, trOl.begin
+ 17));

        optOl.brush = new ColorBrush(Color.Magenta);
        optOl.joinStyle = CharacterOutlineJoinStyle.RoundJoin;
        setOutlineOptions(optOl, new TextRange(trOl.begin + 20, trOl.begin
+ 27));


        // Character scaling
        appendPlainText("\n\n\n");
        var trScl = appendPlainText("Character scaling: 0% 200% 400%\n");
        appendPlainText("\n");
        setCharacterHeightScale(200, trScl.end - 10, trScl.end - 6);
        setCharacterHeightScale(400, trScl.end - 5, trScl.end - 1);
        // Character spacing
        var trSpc = appendPlainText("Character spacing: POSITIVE-
NEGATIVE\n");
        appendPlainText("\n");
        setFontSize(new FontSize(20), trSpc);
        setCharacterSpacing(0.5, trSpc.end - 20, trSpc.end - 11);
```

```
        setCharacterSpacing(-0.2, trSpc.end - 10, trSpc.end - 1);
        // Character rotation
        var trRot = appendPlainText("Character rotation: 90 degrees / 180
degrees\n");
        appendPlainText("\n");
        setFontSize(new FontSize(15), trRot);
        setCharacterRotation(CharacterRotation.Angle90, trRot.end - 25,
              trRot.end - 15);
        setCharacterRotation(CharacterRotation.Angle180, trRot.end - 12,
              trRot.end - 1);
        // Character frame
        appendPlainText("\n");
        var trFrm = appendPlainText("Character frame: negative indent /
positive indent \n");
        appendPlainText("\n");
        setFontSize(new FontSize(15), trFrm);

        var charFrame1 = new CharacterFrame(true, Color.Green, 100, 100,
"-2mm", "-2mm",              "-7pt", "-7pt", "2pt", "dotted");
        setCharacterFrame(charFrame1, trFrm.end - 37, trFrm.end - 22);

        var charFrame2 = new CharacterFrame();
        charFrame2.active = true;
        charFrame2.brush = new ColorBrush(Color.Red);
        charFrame2.shade = 100;
        charFrame2.opacity = 100;
        charFrame2.leftIndent = "2mm";
        charFrame2.rightIndent = "2mm";
        charFrame2.topIndent = "2pt";
        charFrame2.bottomIndent = "2pt";
        charFrame2.frameWidth = "2pt";
        charFrame2.lineStyle = "dashed-2-2";
        setCharacterFrame(charFrame2, trFrm.end - 17, trFrm.end - 2);

        // Text alignment
        var trAlig;
        for (i = 0; i < 5; i++)
            trAlig = appendPlainText("Right text alignment is demonstrated
in this paragraph.");
        appendPlainText("\n\n");
        setTextAlignment(ParagraphAlignment.Right, trAlig.begin,
trAlig.end);
        for (i = 0; i < 5; i++)
            trAlig = appendPlainText("Center text alignment is demonstrated
in this paragraph.");
        appendPlainText("\n\n");
        setTextAlignment(ParagraphAlignment.Center, trAlig.begin,
trAlig.end);
        for (i = 0; i < 4; i++)
            trAlig = appendPlainText("Forced justified alignment is
demonstrated in this paragraph.");
        appendPlainText("See the he last line of the paragraph. It is
```

```
stretched to whole width of the text object.");
        appendPlainText("\n\n");
        setTextAlignment(ParagraphAlignment.ForcedJustified, trAlig.begin,
trAlig.end);
        // Object inserted in the formatted text
        var objGr = doc.objects().create(ObjectType.Graphic);
        if (objGr) {
            with (objGr) {
                form = FrameForm.Rectangle;
                pageGeometryBounds = ["0", "0", "30mm", "20mm"];
                frameColor = new ColorBrush(Color.Red);
                frameWidth = "2pt";
                fillColor = new ColorBrush(Color.Yellow);
                fillShade = 100;
            }
            var trObj = appendPlainText("Some object is inserted here
inside the formatted text.\n");
            appendPlainText("\n");
            setObject(objGr, trObj.begin + 29);
        }
        // Character position correction.
        var trChc = appendPlainText("Position of the letter S is corrected
by 20 points down and 40 points right.\n");
        appendPlainText("\n");
        setPositionCorrection(new CharacterPositionCorrection(20, 40),
              trChc.begin + 23);
    }
}
```

Class: TextRanges

The class ***TextRanges*** holds the collection of ***TextRange*** objects.

Properties:

length	*int Read*

Returns number of items in the collection.

Methods:

at*(int index)*	*TextRange*

Returns an item of TextRange object from the ***index*** position of the collection.

Class: TextRange

The class ***TextRange*** implements an object that describes a fragment of the text, i.e. its beginning and end.

Constructor:

new **TextRange***(int begin, int end)*

Properties:

begin	*int*	*Read/Write*

Position of the beginning of the text fragment inside the original text.

end	*int*	*Read/Write*

Position of the end of the text fragment inside the original text. The value is the index of the character behind the last character that is included into the range.

length	*int*	*Read*

Length of the text fragment, i.e. the number of characters and all objects that are inserted into the text fragment.

Methods:

setRange*(int begin, int end)*	*void*

Set the beginning and end of the text range.

begin: Index of the first character of the text range inside the whole text.

end: Index of the position behind the last character of the text range.

Class: TextCursor

The object *TextCursor* holds the text that is currently edited with the text cursor. Use the property *Document.textCursor* to get it; it is *null* if no text is edited.

Properties:

text	*FormattedText*	*Read*

Returns objects of the type FormattedText that hold all text.

selection	*TextRange*	*Read/Write*

Set/get the range of text that is selected.

textChain	*TextChain*	*Read*

Returns the text chain (the linked text objects) that holds the current text.

selections	*TextRanges*	*Read*

Returns all selected text ranges, or *null* if no text is selected.

Object shapes

Version 12 — November 2026

Object shapes

Class: Point

The class *Point* implements an object that describes one point of the polygon point list with a flag description. The point structure is an array with two or three items: *[x coordinate, y coordinate, flags]*.

There are two types of points in the polygon points list: sizing point and control point. The sizing point defines the position of the point in the polygon and the control point defines a curvature in the neighborhood of a point. The sizing point can be defined without flag information (i.e. *[x,y]*) or with flag information (i.e. *[x,y,flags]*). The control point must always define the flags information. A sizing point may have one, two or no control points. The first control point is defined before the sizing point and the second one is defined after sizing point. Further polygons can be joined to one object with flags that are defined with the sizing point.

List of usable flags for the control point:
"*A*" - after the control point, the point must be defined after the sizing point in the polygon point list

"*AC*" - after the control point, the point is bounded in the before control point. An object does not keep the constraint: read back, the point has the flag "A".

"*B*" - before the control point, the point must be defined before the sizing point in the polygon point list

"*BC*" - before the control point, the point is bounded in the after control point. An object does not keep the constraint: read back, the point has the flag "B".

List of usable flags for the sizing point:
"*F*" - first sizing point of a subpolygon; it starts a new subpolygon in the polygon point list. A list with only one polygon does not need it.

"*L*" - last sizing point of a subpolygon. It is optional: the next "F" point ends the subpolygon.

Constructor:
new `Point`*(x,y)*
new `Point`*(x,y,flags)*
new `Point`*([x, y, flags])*
x, *y*: Coordinates as numbers in points or as strings with unit, e.g. *"15mm"*.

flags: A number, the values of the enumeration type *PolygonPointFlag* combined with *|*, e.g. *PolygonPointFlag.Control | PolygonPointFlag.Before* for a "B" control point. The letters ("A", "B", "F" ...) are only understood in the arrays of a *PointList*.

Properties:

x	*UnitValueString*	*Read/Write*

Set/get x coordinate of the point. Reading returns the value in points as a string without unit.

y	*UnitValueString*	*Read/Write*

Set/get y coordinate of the point. Reading returns the value in points as a string without unit.

flags	*short*	*Read/Write*

Set/get flag information. Each flag is represented by one bit in the word. The bits are the values of the enumeration type *PolygonPointFlag*: *Control* (1), *Before* (2), *Constraint* (4), *First* (8) and *Last* (16). A sizing point without flags has the value *Default* (0); an "after" control point has *Control* only. *Align* (32) is reserved.

flagAfter	*bool*	*Read/Write*

Set/get/clear flag combination for "after" control point.

flagAfterConstraint	*bool*	*Read/Write*

Set/get/clear flag combination for "after constraint" control point.

flagBefore	*bool*	*Read/Write*

Set/get/clear flag combination for "before" control point.

flagBeforeConstraint	*bool*	*Read/Write*

Set/get/clear flag combination for "before constraint" control point.

flagFirst	*bool*	*Read/Write*

Set/get/clear flag combination for "first" sizing point.

flagLast	*bool*	*Read/Write*

Set/get/clear flag combination for "last" sizing point.

Setting one of the *flag* properties replaces all flags of the point; setting it to *false* clears all flags. Changing the flags of a point from *PointList.getPoint()* does not change the list: write the point back with *setPoint()*. Changing *x* or *y* changes the list at once.

Examples:

```
  var point = [335.56, 370.928, "F"];
  var point = [304.554, 643.618];
  var point = new Point(["15.3cm", "462pt", PolygonPointFlag.Control |
PolygonPointFlag.Before]);
```

Class: PointList

The class *PointList* implements an object that describes a polygon points list. The polygon point list is composed with the points. (See the chapter "Class: Point" for the point description.). All points are shifted to one level array consecutively. The control points must be set before or after the sizing point that is affected by these control points. When the polygon point list contains several subpolygons, each subpolygon starts with a sizing point with the flag "F" (first). The point with the index 0 is the first point of the path.

The point coordinates are in points. They relate to the page (*pageGeometryPoints*) or to the spread (*spreadGeometryPoints*).

A point list read from an object marks the first sizing point of every subpolygon with "F" and the last one with "L". It gives both control points of every curved segment; a control point that has no effect lies on its sizing point. A curved closing segment (from the last sizing point back to the first one) has an "A" control point after the last sizing point and a "B" control point before the first sizing point of the subpolygon.

Changes to a point list read from an object take effect when the list is assigned to the object again.

Constructor:
new `PointList()`
new `PointList(Array points)`
points: An array of points *[x, y]* or *[x, y, flag]*; the flag is one of the letters *"F"*, *"L"*, *"A"*, *"B"*, *"AC"* and *"BC"* (see "Class: Point").

Properties:

length	*int*	*R*

Returns the number of points in the polygon point list.

isGroup	*bool*	*R*

Returns *true* for every point list, also for a list with only one polygon. Use *numberOfSubPolygons* to find out whether the list contains several subpolygons.

numberOfSizingPoints	*int*	*R*

Returns the number of sizing points.

numberOfSubPolygons	*int*	*R*

Returns the number of subpolygons in the polygon points list (the number of points with the flag "F", at least 1).

Methods:

getPoint *(int index)*	*Point*

Returns the Point object from the polygon point list from its *index* position.

setPoint *(Point point, int index)*	*void*

Replaces the point object to the polygon point list at its *index* position.

insertPoint *(Point point, int index)*	*void*

Inserts a new point object in the polygon point list at its *index* position. The current object and all following objects move up.

appendPoint *(Point point)*	*void*

Appends a new point object at the end of the polygon point list.

deletePoint *(int index)*	*void*

Deletes the point object from its index position in the polygon point list.

Examples 1: The polygon point list contains two subpolygons.

```
var pointArray = [[335.56, 370.928, "F"], [244.928, 465.535], [304.554,
643.618],
   [512.848, 662.945], [562.139, 436.914], [457.992, 380.468, "B"],
   [461.172, 350.258, "L"], [464.352, 320.047, "A"], [350.665, 464.74, "F"],
   [380.08, 544.241], [420.626, 547.421, "B"], [420.626, 547.421],
   [400.626, 527.421, "A"], ["15,3cm", "462pt"], [391.689, 407.379, "B"],
   [392.638, 447.249, "L"], [393.58, 487.119, "A"]];

var pointList = new PointList(pointArray);
var doc = application.currentDocument;
if (doc) {
```

```
  var obj = doc.objects().create(ObjectType.Text);
  if (obj) {
    obj.form = FrameForm.Polygon;
  obj.pageGeometryPoints = pointList;
  doc.currentPage.add(obj);
  }
}
```

Examples 2:

```
var pointArray = [[335.56, 370.928, "F"], [244.928, 465.535], [304.554,
643.618],
  [512.848, 662.945], [562.139, 436.914], [457.992, 380.468, "B"],
  [461.172, 350.258, "L"], [464.352, 320.047, "A"], [350.665, 464.74, "F"],
  [380.08, 544.241], [420.626, 547.421, "B"], [420.626, 547.421],
  [400.626, 527.421, "A"], ["15,3cm", "462pt"], [391.689, 407.379, "B"],
  [392.638, 447.249, "L"], [393.58, 487.119, "A"]];

var arr = new PointList(pointArray);

for (var j = 0; j < arr.length; j++) {
  var p = arr.getPoint(j);
  messageBox.information("Msg:", "Point: [" + p.x + "," + p.y + ","
+ p.flags +      "]" + "\nFlagAfter: " + p.flagAfter +
       "\nFlagAfterConstraint: " + p.flagAfterConstraint +
       "\nFlagBefore: " + p.flagBefore +
       "\nFlagBeforeConstraint: " + p.flagBeforeConstraint +
       "\nFlagFirst: " + p.flagFirst +
       "\nFlagLast: " + p.flagLast);
}

arr.setPoint(new Point("40mm", "200mm", PolygonPointFlag.First), 0);
var p = arr.getPoint(5);
p.flagBeforeConstraint = true;
arr.setPoint(p, 5);
arr.appendPoint(new Point("14cm", "250mm"));
```

Class: BoundingBox

This simple class ***BoundingBox*** contains 4 numbers which represent the bounding box of the object. The properties ***skew***, ***rotation*** and ***mirrored*** have an impact on the values.

Constructor:

```
new BoundingBox(UnitValueString x, UnitValueString y,
     UnitValueString width,  UnitValueString height)
new BoundingBox(Array values)
```

The values are numbers in points or strings with unit, e.g. *"10mm"*. ***values*** is an array *[x, y, width, height]*. Reading a property returns the value in points as a string without unit.

Properties:

x *UnitValueString* *Read/Write*

X-coordinate of the object.

y	*UnitValueString*	*Read/Write*

Y-coordinate of the object.

width	*UnitValueString*	*Read/Write*

Width of the object.

height	*UnitValueString*	*Read/Write*

Height of the object.

Class: Matrix

The class ***Matrix*** holds 6 parameters for the object shape transformation. The transformation formula is as follows:

```
xnew = m11 * x + m21 * y + dx
ynew = m22 * y + m12 * x + dy
```

The class is used in the class ***Object***.

Constructor:

***new* Matrix();**
***new* Matrix(*float m11, float m12, float m21, float m22, UnitValueString dx, UnitValueString dy*)**
***new* Matrix(*Array values*)**

dx, ***dy***: Translation as a number in points or as a string with unit, e.g. *"10mm"*. ***values*** is an array *[m11, m12, m21, m22, dx, dy]*.

Properties:

m11	*float*	*Read/Write*
m12	*float*	*Read/Write*
m21	*float*	*Read/Write*
m22	*float*	*Read/Write*
dx	*float*	*Read/Write*
dy	*float*	*Read/Write*

m11 and ***m22*** scale, ***m12*** and ***m21*** shear and rotate, ***dx*** and ***dy*** move (in points).



Brushes and Colors

Version 12 — November 2026

Brushes and Colors

Introduction

There are two types of brush objects: ***ColorBrush*** and ***BlendBrush***. The class ***Brush*** is a root class of both brush objects. The class ***Brushes*** holds the repository of all (predefined and created) brushes.

Color Constant Enumeration Types:
Color: *Black, Blue, Bronze, Brown, Cyan, Gold, Green, Magenta, Orange, Pink, Red, Salmon, Silver, Turquoise, Violet, White, Yellow*

Class: Brushes

The predefined repository of brushes can be obtained from the document using the property ***brushes*** of the Document object:

```
var mybrushes = doc.brushes;
```

The class works as a factory for creating a new brush.

Properties:

length *int* *Read*

Returns number of brushes in the repository.

Methods:

addColor*(name)* *ColorBrush*

Adds a new brush with "name" to the list of brushes and returns a ColorBrush object. If "name" is not unique, an exception is thrown.

addBlend*(name)* *BlendBrush*

Adds a brush with "name" to the list of brushes and returns BlendBrush. If "name" is not unique, an exception is thrown.

at*(index)* *Brush*

Returns ***ColorBrush*** or ***BlendBrush*** from the index position in the repository.

find*(name)* *Brush*

Returns the ***ColorBrush*** or ***BlendBrush*** with the unique name ***name*** (e.g. ***"[Red]"*** for a predefined color) or ***null*** if nothing is found.

Class: Brush

The class ***Brush*** is a root class of brush objects.

Properties:

uniqueName *String* *Read/Write*

Get/Set the unique name of brush object.

localizedName	*String*	*Read*

Get the localized name of brush object, i.e. the name in the language of the user interface (e.g. *"[Rot]"* for *"[Red]"* in German).

hidden	*bool*	*Read/Write*

Get/Set the flag hidden.

Methods:

remove()	*bool*

Removes Brush from the list of brushes.

Class: ColorBrush

The object defines a color brush.

Constructor:

new **ColorBrush***(Color color)*

Returns the predefined color brush of the current document (or of the application if no document is open) for a value of the enumeration type *Color*, e.g. *new ColorBrush(Color.Red)*. Use it where a property expects a *ColorBrush*, e.g. *StrikethroughOptions.brush*. New colors are created with *Brushes.addColor()*.

Properties:

rgb	*Rgb*	*Read/Write*

Get/Set the object of the RGB color model (object Rgb with needed values)

cmyk	*Cmyk*	*Read/Write*

Get/Set the object of the CMYK color model (object Cmyk with needed values)

hsv	*Hsv*	*Read/Write*

Get/Set the object of the HSV color model (object Hsv with needed values)

lab	*Lab*	*Read/Write*

Get/Set the object of the LAB color model (object Lab with needed values)

colorSeparation	*ColorSeparation (enum)*	*Read/Write*

Get/Set a type of color separation. See the enumeration type ColorSeparation.

screenAngle	*ScreenAngle (enum)*	*Read/Write*

Get/Set a type of screen angle. See the enumeration type ScreenAngle. It is used for ColorSeparation.SpotColor option only

colorModel	*ColorModel (enum)*	*Read*

Get the current color model. See the enumeration type ColorModel.

indexModel	*IndexModel (enum)*	*Read*

Get the current index model. See the enumeration type IndexModel

indexKey	*String*	*Read*

Get the current index key.

Class: BlendBrush

The object defines the brush type that is composed of a colors and the transition between them. Two Stop objects are created by default when new ***BlendBrush*** object is created.

Properties:

type *BlendType (enum)* *Read/Write*

Get/Set type of the blend brush. See the enumeration type ***BlendType.***

stopCount *int* *Read*

Get the number of Stop objects.

Methods:

stopAt *(int index)* *Stop*

Get the Stop object from the index position.

setStopAt *(int index, Stop object)* *void*

Update the Stop object on the index position in the BlendBrush object.

removeStopAt *(int index)* *bool*

Removes the Stop object from the index position in the BlendBrush object.

Returns true if object was deleted succesfully, otherwise false value.

appendStop *(Stop object)* *void*

Appends the Stop object at the end of the BlendBrush object.

Examples:

```
var doc = application.currentDocument;
if (doc) {

    var brushes = doc.brushes;
    if (brushes) {

        var stop1 = new Stop(Color.Green, 5, 6, 7, 8);
        var stop2 = new Stop(Color.Yellow);
        stop2.shade = 15;
        stop2.opacity = 16;
        stop2.position = 17;
        stop2.focusPosition = 18;
        var stop3 = new Stop(Color.Red, 25, 26, 27, 28);
        var stop4 = new Stop(Color.Blue, 35, 36, 37, 38);

        try {
            var bc = brushes.addBlend("BlendNew");
            bc.type = BlendType.Radial;
            messageBox.information("INFO", "Stop Count: " + bc.stopCount);
            bc.setStopAt(1, stop1);
```

```
            bc.appendStop(stop2);
            bc.appendStop(stop3);
            bc.appendStop(stop4);
            messageBox.information("INFO", "Stop Count: " + bc.stopCount);
            bc.removeStopAt(3);
            messageBox.information("INFO", "Stop Count: " + bc.stopCount);
        }
        catch (err) {
            // color exists
        }
    }
}
```

Class: Stop

The Stop object holds the data about one color in the ***BlendBrush*** object.

Constructor:

new* Stop*(ColorBrush brush)
new* Stop*(ColorBrush brush, float shade, float opacity,
float position, float focusPosition)

brush: A ***ColorBrush*** object or a value of the enumeration type ***Color***, e.g. ***Color.Green***. Without the other parameters, ***shade*** and ***opacity*** are 100, ***position*** is 0 and ***focusPosition*** is 50.

Properties:

colorBrush	*ColorBrush*	*Read/Write*

Get/Set the color brush.

shade	*float*	*Read/Write*

Get/Set the shade of the color. The range is 0-100. [%]

opacity	*float*	*Read/Write*

Get/Set the opacity of the color. The range is 0-100. [%]

position	*float*	*Read/Write*

Get/Set the position of the color. The range is 0-100. [%]

focusPosition	*float*	*Read/Write*

Get/Set the focus position. The range is 0-100. [%]

Class: Rgb

The class defines the RGB color model.

Constructor:

new Rgb (*int red*, *int green*, *int blue*)

Properties:

red	*int (0-255)*	*Read/Write*

Get/Set a red part of the color model.

green *int (0-255)* *Read/Write*
Get/Set a green part of the color model.

blue *int (0-255)* *Read/Write*
Get/Set a blue part of the color model.

Class: Cmyk

The class defines the CMYK color model.

Constructor:

new Cmyk (*int cyan*, *int magenta*, *int yellow*, *int key*)

Each part is a value from 0 to 255 (255 means 100%), not a percentage.

Properties:

cyan *int (0-255)* *Read/Write*
Get/Set a cyan part of the color model.

magenta *int (0-255)* *Read/Write*
Get/Set a magenta part of the color model.

yellow *int (0-255)* *Read/Write*
Get/Set a yellow part of the color model.

key *int (0-255)* *Read/Write*
Get/Set the black (key) part of the color model.

Class: Hsv

The class defines the HSV color model.

Constructor:

new **Hsv***(int hue, int saturation, int value)*

Properties:

hue *int (0-359) [degree]* *Read/Write*
Get/Set a hue for the color model.

saturation *int (0-255)* *Read/Write*
Get/Set a saturation for the color model.

value *int (0-255)* *Read/Write*
Get/Set a value for the color model.

Class: Lab

The class defines the LAB color model.

Constructor:

new **Lab***(int lightness, int positionA, int positionB)*

Properties:

lightness	*int (0-100)*	*Read/Write*

Get/Set a lightness for the color model.

a	*int (-128-127)*	*Read/Write*

Get/Set a position between green and red for the color model. Negative values indicate green while positive values indicate red.

b	*int (-128-127)*	*Read/Write*

Get/Set a position between blue and yellow for the color model. Negative values indicate blue and positive values indicate yellow.

Examples

Example 1:

```
var doc = application.currentDocument;
if (doc) {
  var brushes = doc.brushes;

  var bg1 = brushes.addBlend("MyBlendBrush");
  var bc1 = brushes.addColor("MyColorBrush");

  var names = "";
  for (var index = 0; index < brushes.length; index++) {
    var brush = brushes.at(index);
    names = names + brush.uniqueName + " : " + brush.hidden + "\n";
  }
  messageBox.information("Brush Unique Names", names);
}
```

Example 2:

```
var doc = application.currentDocument;
if (doc) {
  var cb = doc.brushes.addColor("MyColor2");
  cb.rgb = new Rgb(200, 100, 0);
  var tp = new TextPreferences();
  tp.lineNumbersColor = cb;
  doc.textPreferences = tp;
}
```

Example 3:

```
var vRgb = new Rgb(12, 23, 130);
var doc = application.currentDocument;
if (doc) {
```

```
  var brushes = doc.brushes;
  if (brushes) {

    try {
      var bc1 = brushes.addColor("Color1");
    }
    catch (err) {
      // color exists
    }

    bc1.rgb = vRgb;
    bc1.hsv = new Hsv(55, 33, 44);
    bc1.cmyk = new Cmyk(11, 111, 12, 113);
    bc1.colorSeparation = ColorSeparation.SpotColor;
    bc1.screenAngle = ScreenAngle.Cyan;

    messageBox.information(
      "CMYK", "Cyan: " + bc1.cmyk.cyan +
      "; Magenta: " + bc1.cmyk.magenta +
      "; Yellow: " + bc1.cmyk.yellow +
      "; Key: " + bc1.cmyk.key);
    messageBox.information("CMYK2", "Cyan: " +
      doc.brushes.find("Color1").cmyk.cyan + "; Magenta: " +
      doc.brushes.find("Color1").cmyk.magenta + "; Yellow: " +
      doc.brushes.find("Color1").cmyk.yellow + "; Key: " +
      doc.brushes.find("Color1").cmyk.key);
    if(bc1.colorModel == ColorModel.Cmyk)
      messageBox.information("Msg:", "ColorModel is 'CMYK'");
    if(bc1.colorSeparation == ColorSeparation.SpotColor)
      messageBox.information("Msg:", "ColorSeparation of color " +
    bc1.uniqueName + " is 'SpotColor'");

    var cba = brushes.find("Color1");
    if(cba)
      cba.remove();

      var stop1 = new Stop(Color.Gold, 55, 60, 0, 33);
    var bc2 = null;
      try {
           bc2 = brushes.addBlend("Color2");
      }
       catch (err) {
           // color exists
       }

      bc2.type = BlendType.Focus;
       bc2.setStopAt(0, stop1);
      bc2.setStopAt(1, new Stop(Color.Green, 88, 90, 95, 15));
  }
}
```



Version 12 — November 2026

StyleSheets

Class: StyleSheets

The predefined repository of style sheets can be obtained from the document using the ***styleSheets*** property of the Document object:

```
var mystylesheets = doc.styleSheets;
```

The class works as a factory for creating new style sheets.

Methods:

length (*StyleSheetFamily familyType) int*

Returns the number of style sheets of the type 'familyType'. See the enumeration type ***StyleSheetFamily***.

create *(String name, StyleSheetFamily familyType)* *StyleSheet*

Creates and returns a new style sheet with "familyType". If "name" is not unique, the ***null*** is returned.

at *(int index, StyleSheetFamily familyType)* *StyleSheet*

Returns one of the style sheet objects depending on the familyType from the index position in the repository. See the enumeration type ***StyleSheetFamily***.

find (*String name)* *StyleSheet*

Returns the StyleSheet object found or ***null*** if nothing is found.

Examples:

```
var doc = application.currentDocument;
if (doc) {
    var ss = doc.styleSheets;

    var ssChar1 = ss.find("MyCharacter1");
    if (!ssChar1)
        ssChar1 = ss.create("MyCharacter1", StyleSheetFamily.Character);
    var ssChar2 = ss.find("MyCharacter2");
    if (!ssChar2)
        ssChar2 = ss.create("MyCharacter2", StyleSheetFamily.Character);
    var ssPar = ss.find("MyParagraph");
    if (!ssPar)
        ssPar = ss.create("MyParagraph", StyleSheetFamily.Paragraph);

    var txt1 = "Number of Character Stylesheets: " +
ss.length(StyleSheetFamily.Character) + "\n";
    txt1 = txt1 + "Number of Paragraph Stylesheets: " +
ss.length(StyleSheetFamily.Paragraph) + "\n";
    messageBox.information("StyleSheets", txt1);

    var txt2 = "";
    for (var i = 0; i < ss.length(StyleSheetFamily.Character) ; i++) {
        var ssr = ss.at(i, StyleSheetFamily.Character);
        txt2 = txt2 + ssr.name + " or " + doc.styleSheets.at(i,
```

```
  StyleSheetFamily.Character).name + "\n";
    }
    messageBox.information("StyleSheets", txt2);

    ssPar.clear();
}
```

Class: StyleSheet

The class ***StyleSheet*** is a root class of all types of style sheet objects: ***CharacterStyleSheet***, ***ParagraphStyleSheet***, ***LayoutStyleSheet***, ***PictureStyleSheet***, ***GraphicStyleSheet***, ***TableStyleSheet***, ***TableRowStyleSheet***, ***TableColumnStyleSheet***, ***TableCellStyleSheet***.

Properties:

name	*String*	*Read/Write*

Get/Set a name of the style sheet.

templateStyleSheet	*StyleSheet*	*Read/Write*

Get/Set a template style sheet. The same type of style sheet must be used.

Methods:

remove()	*boolean*

Removes the style sheet from the style sheet repository.

clear()	*void*

Clears a current setting of the style sheet.

Examples:

```
var doc = application.currentDocument;
var stls = doc.styleSheets;
if (stls)
{
 messageBox.information("MSG", stls.length(StyleSheetFamily.Character));

 doc.styleSheets.at(0,StyleSheetFamily.Character).name = "Base";
 messageBox.information("MSG", doc.styleSheets.at(0,
StyleSheetFamily.Character).name);

 var ssBase = doc.styleSheets.at(0, StyleSheetFamily.Character);
 var templSS = doc.styleSheets.create("TemplateChar",
StyleSheetFamily.Character);
    ssBase.templateStyleSheet = templSS;
 messageBox.information("MSG",
       doc.styleSheets.find("Base").templateStyleSheet.name);

}
```

Class: CharacterStyleSheet

The class holds a group of style parameters that are applied to the characters from the text range. The property ***errorStatus*** is set when the error is occurred or the property is not set in the style sheet.

Properties:

font *FontName* *Read/Write*

Returns the font name and the font type or ***null*** when the font is not set.

Sets the font name and the font type.

The parameter format is ***<font name>-<font type>***.

Sample: ***'Arial-Bold', 'Georgia-Bold Italic'***

fontSize *FontSize* *Read/Write*

Returns the font size object.

Sets the font size and font size reference type.

language *Language* *Read/Write*

Returns the language id or the value ***NoLang*** when the language is not set.

Sets the language.

brush *ColorBrush* *Read/Write*

Returns the color of the brush or ***null*** when the color is not set.

Sets the color of the brush.

shade *float* *Read/Write*

Returns the shade of the color.

Sets the shade of the color.

opacity *float* *Read/Write*

Returns the opacity of the color.

Sets the opacity of the color.

characterPosition	*CharacterPosition (enum)*	*Read/Write*
characterUpperLowerCase	*CharacterUpperLowerCase (enum)*	*Read/Write*
underline	*UnderlineOptions*	*Read/Write*
strikethrough	*StrikethroughOptions*	*Read/Write*
characterHeightScale	*float*	*Read/Write*
characterWidthScale	*float*	*Read/Write*
characterSpacing	*float*	*Read/Write*
characterKerning	*CharacterKerning*	*Read/Write*
characterBaselineShift	*float*	*Read/Write*

characterBaselineOrientation	*CharacterBaselineOrientation*	*Read/Write*
characterRotation	*CharacterRotation (enum)*	*Read/Write*
characterLigatures	*bool*	*Read/Write*
outlineOptions	*OutlineOptions*	*Read/Write*
characterFrame	*CharacterFrame*	*Read/Write*
characterBackground	*CharacterBackground*	*Read/Write*
characterRule	*CharacterRule*	*Read/Write*

Assign a ***CharacterFrame***, ***CharacterBackground*** or ***CharacterRule*** object; its property ***active*** switches the setting on or off. ***null*** removes the setting from the style sheet.

tablesAbbreviationEntry	*TablesAbbreviation*	*Read/Write*
tablesBibliographyEntry	*TablesBibliography*	*Read/Write*
tablesContentsEntry	*TablesContents*	*Read/Write*
tablesIndexEntry	*TablesIndex*	*Read/Write*
tablesPictureEntry	*TablesPicture*	*Read/Write*
tablesRunningTitleEntry	*TablesRunningTitle*	*Read/Write*
link	*Link*	*Read/Write*
openType	*OpenType*	*Read/Write*
characterNoKashida	*bool*	*Read/Write*

Get/Set whether the characters are kept out of kashida justification (see ***kashidaLevel*** of the class ***ParagraphStyleSheet***).

characterDigits	*Digits*	*Read/Write*

Get/Set the digits of numbers: of list, note and page numbers and of the digits in the text. ***Digits.Default*** uses the digits of the language of the text. See the enumeration type ***Digits***.

Methods:

remove *(String propertyName)*	*void*

The method removes the property setting from the style sheet. The value propertyName must be set to used property name from following list: ‘font’, ‘fontSize’, ‘language’, ‘brush’, ‘shade’, ‘opacity’, ‘underline’, ‘strikethrough’, ‘characterPosition’, ‘characterUpperLowerCase’, ‘outlineOptions’, ‘characterHeightScale’, ‘characterWidthScale’, ‘characterBaselineShift’, ‘characterBaselineOrientation’, ‘characterSpace’, ‘characterKerning’, ‘characterRotation’, ‘characterLigatures’, ‘characterNoKashida’, ‘characterDigits’, ‘characterFrame’, ‘characterRule’, ‘characterBackground’, ‘tablesIndex’, ‘tablesContents’, ‘tablesRunningTitle’, ‘tablesBibliography’, ‘tablesPicture’, ‘tablesAbbreviation’. ‘characterSpace’ removes the property characterSpacing.

Class: ParagraphStyleSheet

The property ***errorStatus*** is set when the error is occurred or the property is not set in the style sheet.

Properties:

characterStyleSheet	*CharacterStyleSheet*	*Read/Write*
font	*FontName*	*Read/Write*

Returns the font name and the font type or *null* when the font is not set.

Sets the font name and the font type.

The parameter format is *<font name>-<font type>*.

Sample: *'Arial-Bold', 'Georgia-Bold Italic'*

fontSize	*FontSize*	*Read/Write*

Returns the font size object. Sets the font size and the reference size type.

language	*Language*	*Read/Write*

Returns the language id or the value NoLang when the language is not set. Sets the language.

brush	*ColorBrush*	*Read/Write*

Returns the color of the brush or *null* when the color is not set. Sets the color of the brush.

shade	*float*	*Read/Write*

Returns the shade of the color. Sets the shade of the color.

opacity	*float*	*Read/Write*

Returns the opacity of the color.

Sets the opacity of the color.

underline	*UnderlineOptions*	*Read/Write*
strikethrough	*StrikethroughOptions*	*Read/Write*

Assign an *UnderlineOptions* or *StrikethroughOptions* object. Its type *FontStyleLineType.Automatic* draws the line with the offset and width of the text preferences (see *TextPreferences*), *FontStyleLine-Type.Manual* with the values of the object, *FontStyleLineType.None* switches the line off. *rangeType* (*FontStyleLineRangeType.All* or *FontStyleLineRangeType.Words*) draws the line under the whole text or word by word.

characterPosition	*CharacterPosition (enum)*	*Read/Write*
characterUpperLowerCase	*CharacterUpperLowerCase (enum)*	*Read/Write*
characterHeightScale	*float*	*Read/Write*
characterWidthScale	*float*	*Read/Write*
characterSpacing	*float*	*Read/Write*
characterKerning	*CharacterKerning*	*Read/Write*
characterBaselineShift	*float*	*Read/Write*
characterBaselineOrientation	*CharacterBaselineOrientation*	*Read/Write*
characterRotation	*CharacterRotation (enum)*	*Read/Write*
characterLigatures	*bool*	*Read/Write*

outlineOptionsData	*OutlineOptions*	*Read/Write*
characterFrame	*CharacterFrame*	*Read/Write*
characterBackground	*CharacterBackground*	*Read/Write*
characterRule	*CharacterRule*	*Read/Write*

Assign a *CharacterFrame*, *CharacterBackground* or *CharacterRule* object; its property *active* switches the setting on or off. *null* removes the setting from the style sheet.

tablesAbbreviationEntry	*TablesAbbreviation*	*Read/Write*
tablesBibliographyEntry	*TablesBibliography*	*Read/Write*
tablesContentsEntry	*TablesContents*	*Read/Write*
tablesIndexEntry	*TablesIndex*	*Read/Write*
tablesPictureEntry	*TablesPicture*	*Read/Write*
tablesRunningTitleEntry	*TablesRunningTitle*	*Read/Write*
link	*Link*	*Read/Write*
openType	*OpenType*	*Read/Write*
automaticCharacterSpacing	*AutomaticCharacterSpacing*	*Read/Write*
automaticCharacterWidth	*AutomaticCharacterWidth*	*Read/Write*
automaticWordSpacing	*AutomaticWordSpacing*	*Read/Write*

Assign an *AutomaticCharacterSpacing*, *AutomaticCharacterWidth* or *AutomaticWordSpacing* object with the minimum, optimum and maximum values in percent. *null* removes the setting from the style sheet.

writingDirection	*WritingDirection (enum)*	*Read/Write*
opticalAlignment	*OpticalAlignment (enum)*	*Read/Write*
supressLineNumbering	*bool*	*Read/Write*
paragraphIndent	*UnitValueString*	*Read/Write*
paragraphLeftIndent	*UnitValueString*	*Read/Write*
paragraphRightIndent	*UnitValueString*	*Read/Write*
hyphenation	*Hyphenation*	*Read/Write*
enumerations	*Enumerations*	*Read/Write*
dropCaps	*DropCaps*	*Read/Write*
textAlignment	*ParagraphAlignment (enum)*	*Read/Write*
paragraphLineSpacing	*UnitValueString*	*Read/Write*

Set word “auto” to set automatically.

spaceToPreviousParagraph	*UnitValueString*	*Read/Write*
spaceToNextParagraph	*UnitValueString*	*Read/Write*

widowsOrphans	*WidowsOrphans*	*Read/Write*

Get/Set the widows and orphans control: a ***WidowsOrphans*** object with ***mode***, ***firstLines*** and ***lastLines***. ***null*** removes the setting from the style sheet.

keepParagraphTogether	*bool*	*Read/Write*
baselineGridType	*BaselineGridType (enum)*	*Read/Write*
paragraphFrame	*ParagraphFrame*	*Read/Write*
paragraphBackground	*ParagraphBackground*	*Read/Write*
paragraphRuleAbove	*ParagraphRule*	*Read/Write*
paragraphRuleBelow	*ParagraphRule*	*Read/Write*
paragraphTabs	*Tabulators*	*Read/Write*
characterNoKashida	*bool*	*Read/Write*

Get/Set whether the characters are kept out of kashida justification (see ***kashidaLevel***).

characterDigits	*Digits*	*Read/Write*

Get/Set the digits of numbers: of list, note and page numbers and of the digits in the text. ***Digits.Default*** uses the digits of the language of the text. See the enumeration type ***Digits***.

kashidaLevel	*KashidaLevel*	*Read/Write*

Get/Set the kashida level (Arabic justification): how far justification may stretch Arabic words with tatweels. ***Short***, ***Medium*** and ***Long*** stretch a word by up to one, two or three tatweel widths; ***Stylistic*** distributes the length over the joins by their priority, at most as ***Long***; ***None*** switches kashida off. See the enumeration type ***KashidaLevel***.

kashidaRules	*String*	*Read/Write*

Get/Set the rule set of kashida: ***"naskh"***, ***"simple"*** or an empty string for automatic (the rule set follows the font). Other names are kept as they are. Level and rules are one setting: setting one keeps the other, and ***remove("kashidaLevel")*** or ***remove("kashidaRules")*** removes both.

Methods:

setWidowsAndOrphans *(WidowsAndOrphans mode, int startLines, int endLines) void*

Sets the widows and orphans control with a value of the enumeration type ***WidowsAndOrphans*** and the numbers of lines. The property ***widowsOrphans*** does the same with a ***WidowsOrphans*** object; use it in new scripts.

remove *(String propertyName) void*

The method removes the property setting from the style sheet. The value propertyName must be set to the used property name from the following list: ‘font’, ‘fontSize’, ‘language’, ‘brush’, ‘shade’, ‘opacity’, ‘underline’, ‘strikethrough’, ‘characterPosition’, ‘characterUpperLowerCase’, ‘outlineOptions’, ‘characterHeightScale’, ‘characterWidthScale’, ‘characterBaselineShift’, ‘characterBaselineOrientation’, ‘characterSpace’, ‘characterKerning’, ‘characterRotation’, ‘characterLigatures’, ‘characterNoKashida’, ‘characterDigits’, ‘characterFrame’, ‘characterRule’, ‘characterBackground’, ‘tablesIndex’, ‘tablesContents’, ‘tablesRunningTitle’, ‘tablesBibliography’, ‘tablesPicture’, ‘tablesAbbreviation’, ‘writingDirection’, ‘opticalAlignment’, ‘kashidaLevel’, ‘kashidaRules’, ‘supressLineNumbering’, ‘paragraphFrame’, ‘paragraphBackground’, ‘paragraphRuleAbove’, ‘paragraphRuleBelow’,

'automaticCharacterSpacing', 'automaticCharacterWidth', 'automaticWordSpacing', 'paragraphLeftIndent', 'paragraphRightIndent', 'paragraphIndent', 'enumerations', 'dropCaps', 'hyphenation', 'textAlignment', 'widowsAndOrphans', 'keepParagraphTogether', 'baselineGrid', 'paragraphLineSpacing', 'spaceToPreviousParagraph', 'spaceToNextParagraph', 'paragraphTabs'. Some names differ from the property names: 'characterSpace' removes characterSpacing, 'outlineOptions' removes outlineOptionsData, 'widowsAndOrphans' removes widowsOrphans and 'baselineGrid' removes baselineGridType.

Example:

```
  var doc = application.currentDocument;
  var ps = doc.styleSheets.create("Arabic Body",
StyleSheetFamily.Paragraph);
  ps.textAlignment = ParagraphAlignment.Justified;
  ps.kashidaLevel = KashidaLevel.Medium;
  ps.kashidaRules = "naskh";
  ps.characterDigits = Digits.ArabicIndic;
  var ul = new UnderlineOptions(FontStyleLineType.Automatic);
  ul.rangeType = FontStyleLineRangeType.Words;
  ps.underline = ul;
  var wo = new WidowsOrphans(WidowsOrphansMode.FirstAndLastLines);
  wo.firstLines = 2;
  wo.lastLines = 2;
  ps.widowsOrphans = wo;
  ps.remove("characterDigits");
```

Class: LayoutStyleSheet

Properties:

lineOrientation	*LayoutLineDirection (enum)*	*Read/Write*
lineJustification	*LayoutLineJustification (enum)*	*Read/Write*
justifiedSpaceBetweenLinesMaximum	*UnitValueString*	*Read/Write*
paragraphSpaceMaximum	*UnitValueString*	*Read/Write*
spaceToPreviousLayout	*UnitValueString*	*Read/Write*
spaceToNextLayout	*UnitValueString*	*Read/Write*
columns	*LayoutColumnsInfo*	*Read/Write*
footnotes	*LayoutFootnotes*	*Read/Write*

Methods:

remove *(String propertyName)* *void*

The method removes the property setting from the stylesheet. The value propertyName must be set to used property name from following list: ‚lineOrientation', ‚lineJustification', ‚spaceToPreviousLayout', ‚spaceToNextLayout', ‚maximumSpaceBetweenLines', ‚maximumSpaceBetweenParagraphs', ‚columns', ‚footnotes'. ‚maximumSpaceBetweenLines' removes justifiedSpaceBetweenLinesMaximum, ‚maximumSpaceBetweenParagraphs' removes paragraphSpaceMaximum.

Class: PictureStyleSheet

Properties:

horizontalScale	*float*	*Read/Write*
verticalScale	*float*	*Read/Write*
offsetX	*String*	*Read/Write*
offsetY	*String*	*Read/Write*
rotation	*float*	*Read/Write*
skew	*float*	*Read/Write*
mirroredHorizontally	*bool*	*Read/Write*
mirroredVertically	*bool*	*Read/Write*
halftoneAngle	*float*	*Read/Write*
halftoneScreen	*float*	*Read/Write*
adjustments	*Object*	*Read/Write*

Get/Set the picture corrections of the style as a plain object with the same keys as the property ***adjustments*** of the class ***PictureContent***, e.g. ***exposure***, ***contrast***, ***saturation***, ***temperature*** or ***sharpenAmount***. Reading gives a copy: change values and assign the object back. Keys left out keep their values. Without corrections the property returns an empty object.

The automatic corrections ***autoTone***, ***autoContrast***, ***autoColor*** and ***autoWhiteBalance*** switched on in the style are measured on each picture that takes the style. ***remove("adjustments")*** takes all corrections off the style.

Methods:

remove *(String propertyName)* *void*

The method removes the property setting from the style sheet. The value propertyName must be set to used property name from following list: horizontalScale, verticalScale, offsetX, offsetY, rotation, skew, mirroredHorizontally, mirroredVertically, halftoneAngle, halftoneScreen, adjustments

Example:

```
  var doc = application.currentDocument;
  var pst = doc.styleSheets.create("Warm Pictures",
StyleSheetFamily.Picture);
  pst.adjustments = { autoTone: true, temperature: 20 };
  pst.adjustments = { saturation: 10 };   // autoTone and temperature stay
  var a = pst.adjustments;
  a.contrast = 15;
  pst.adjustments = a;
```

Class: GraphicStyleSheet

Properties:

x	*String*	*Read/Write*

y	*String*	*Read/Write*
height	*String*	*Read/Write*
width	*String*	*Read/Write*
rotation	*float*	*Read/Write*
skew	*float*	*Read/Write*
mirrored	*bool*	*Read/Write*
fillBrush	*Brush*	*Read/Write*
fillShade	*float*	*Read/Write*
fillBlendAngle	*float*	*Read/Write*
baseOpacity	*float*	*Read/Write*
fillOpacity	*float*	*Read/Write*
contentOpacity	*float*	*Read/Write*
printable	*bool*	*Read/Write*
guideObject	*bool*	*Read/Write*
lineColor	*ColorBrush*	*Read/Write*
lineShade	*float*	*Read/Write*
lineOpacity	*float*	*Read/Write*
lineWidth	*String*	*Read/Write*
lineCornerRadius	*String*	*Read/Write*
lineStyle	*String*	*Read/Write*
lineStart	*LineCap*	*Read/Write*
lineEnd	*LineCap*	*Read/Write*
runaround	*Runaround*	*Read/Write*

Get/Set the runaround as a ***Runaround*** object, e.g. *`gs.runaround = new Runaround(RunaroundMode.All, RunaroundShapeType.Block, "2mm")`*.

pictureIndividualAliasContent	*bool*	*Read/Write*
leftTextIndent	*String*	*Read/Write*
topTextIndent	*String*	*Read/Write*
rightTextIndent	*String*	*Read/Write*
bottomTextIndent	*String*	*Read/Write*
linkableWithOriginal	*bool*	*Read/Write*
individualAliasContent	*bool*	*Read/Write*
baselinePosition	*BaselinePosition*	*Read/Write*

baselineDescenders	*bool*	*Read/Write*
columnCount	*int*	*Read/Write*
columnGutter	*String*	*Read/Write*
columnLines	*ColumnLines*	*Read/Write*
baselineGrid	*BaselineGrid*	*Read/Write*
header	*HeaderFooter*	*Read/Write*
footer	*HeaderFooter*	*Read/Write*

Get/Set the header or footer area of text objects as a ***HeaderFooter*** object with the properties ***active*** and ***height***.

footnotes	*GraphicFootnotes*	*Read/Write*
dropShadow	*Shadow*	*Read/Write*
innerShadow	*Shadow*	*Read/Write*
innerGlow	*InnerGlow*	*Read/Write*
outerGlow	*OuterGlow*	*Read/Write*
colorOverlay	*ColorOverlay*	*Read/Write*
reflection	*Reflection*	*Read/Write*
transparency	*Transparency*	*Read/Write*
stroke	*Object*	*Read/Write*
gradientOverlay	*Object*	*Read/Write*
bevelEmboss	*Object*	*Read/Write*

Get/Set the layer effects stroke, gradient overlay and bevel and emboss of the style. Each is a plain object with the same keys as the property of the same name of the class ***GraphicObject***, e.g. ***active***, ***width*** and ***position*** of the stroke. Reading gives a copy: change values and assign the object back. Keys left out keep their values. Without the effect the property returns an empty object.

Method:

remove *(String propertyName)* *void*

The method removes the property setting from the stylesheet. The value propertyName must be set to used property name from following list: 'x', 'y', 'height', 'width', 'rotation', 'skew', 'mirrored', 'fillBrush', 'fillShade', 'fillBlendAngle', 'fillOpacity', 'baseOpacity', 'contentOpacity', 'printable', 'guideObject', 'lineColor', 'lineShade', 'lineOpacity', 'lineWidth', 'lineCornerRadius', 'lineStyle', 'lineStart', 'lineEnd', 'runaround', 'pictureIndividualAliasContent', 'leftTextIndent', 'topTextIndent', 'rightTextIndent', 'bottomTextIndent', 'linkableWithOriginal', 'individualAliasContent', 'baselinePosition', 'baselineDescenders', 'columnCount', 'columnGutter', 'columnLines', 'baselineGrid', 'header', 'footer', 'footnotes', 'dropShadow', 'innerShadow', 'innerGlow', 'outerGlow', 'colorOverlay', 'reflection', 'transparency', 'stroke', 'gradientOverlay', 'bevelEmboss'

Example:

```
var doc = application.currentDocument;
```

```
  var gs = doc.styleSheets.create("Framed", StyleSheetFamily.Graphic);
  gs.stroke = { active: true, width: 3, color: "[Black]", position:
"inside" };
  gs.bevelEmboss = { active: true, size: 4 };
  gs.runaround = new Runaround(RunaroundMode.All, RunaroundShapeType.Block,
"2mm");
  var hf = new HeaderFooter();
  hf.active = true;
  hf.height = "10mm";
  gs.header = hf;
```

Class: TableStyleSheet

Properties:

fillBrush	*Brush*	*Read/Write*
fillShade	*float*	*Read/Write*
fillBlendAngle	*float*	*Read/Write*
lineBrush	*ColorBrush*	*Read/Write*
lineShade	*float*	*Read/Write*
lineStyle	*LineStyleString*	*Read/Write*
lineWidth	*UnitValueString*	*Read/Write*
separator	*TableLineSeparator*	*Read/Write*
headerRows	*bool*	*Read*
headerRowsData	*RowColumnStyle*	*Read*
footerRows	*bool*	*Read*
footerRowsData	*RowColumnStyle*	*Read*
alternatingRows	*bool*	*Read*
alternatingRowsData	*RowColumnStyleAlt*	*Read*
leftColumns	*bool*	*Read*
leftColumnsData	*RowColumnStyle*	*Read*
rightColumns	*bool*	*Read*
rightColumnsData	*RowColumnStyle*	*Read*
alternatingColumns	*bool*	*Read*
alternatingColumnsData	*RowColumnStyleAlt*	*Read*

Methods:

setHeaderRows *(bool on, RowColumnStyle data)*
setFooterRows *(bool on, RowColumnStyle data)*

setAlternatingRows *(bool on, RowColumnStyleAlt data)*	
setLeftColumns *(bool on, RowColumnStyle data)*	
setRightColumns *(bool on, RowColumnStyle data)*	
setAlternatingColumns *(bool on, RowColumnStyleAlt data)*	
remove *(String propertyName)*	*void*

The method removes the property setting from the stylesheet. The value propertyName must be set to used property name from following list: 'fillBrush', 'fillShade', 'fillBlendAngle', 'lineBrush', 'lineShade', 'lineStyle', 'lineWidth', 'separator', 'headerRows', 'footerRow', 'alternatingRows', 'leftColumns', 'rightColumns', 'alternatingColumns'

Class: RowColumnStyle

The class is used in the class TableStyleSheet.

Constructor:
new **RowColumnStyle()**
new **RowColumnStyle***(int count, String styleSheetName)*

Properties:

active	*bool*	*Read*
count	*int*	*Read/Write*
styleSheet	*StyleSheet*	*Read/Write*
styleSheetByName	*String*	*Read/Write*

Set name of row or column stylesheet.

Class: RowColumnStyleAlt

The class is used in the class TableStyleSheet.

Constructor:

```
new RowColumnStyleAlt()
new RowColumnStyleAlt(
   int firstCount, String firstStyleSheetName,
   int secondCount, String secondStyleSheetName,
   int skipFirstCount, int skipLastCount)
```

Properties:

active	*bool*	*Read*
firstCount	*int*	*Read/Write*
firstStyleSheetByName	*String*	*Read/Write*
firstStyleSheet	*StyleSheet*	*Read/Write*
secondCount	*int*	*Read/Write*

secondStyleSheetByName	*String*	*Read/Write*
secondStyleSheet	*StyleSheet*	*Read/Write*
skipFirstCount	*int*	*Read/Write*
skipLastCount	*int*	*Read/Write*

Class: TableRowStyleSheet

Properties:

fillBrush	*Brush*	*Read/Write*
fillShade	*float*	*Read/Write*
fillBlendAngle	*float*	*Read/Write*
horizontalSeparator	*bool*	*Read*
horizontalSeparatorData	*LineStyle*	*Read*
verticalSeparator	*bool*	*Read*
verticalSeparatorData	*LineStyle*	*Read*

Methods:

setHorizontalSeparator *(bool on, LineStyle data)*

setVerticalSeparator *(bool on, LineStyle data)*

remove *(String propertyName)* *void*

The method removes the property setting from the stylesheet. The value propertyName must be set to used property name from following list: 'fillBrush', 'fillShade', 'fillBlendAngle', 'horizontalSeparator', 'verticalSeparator'

Class: TableColumnStyleSheet

Properties:

fillBrush	*Brush*	*Read/Write*
fillShade	*float*	*Read/Write*
fillBlendAngle	*float*	*Read/Write*
horizontalSeparator	*bool*	*Read*
horizontalSeparatorData	*LineStyle*	*Read*
verticalSeparator	*bool*	*Read*
verticalSeparatorData	*LineStyle*	*Read*

Method:

setHorizontalSeparator *(bool on, LineStyle data)*

setVerticalSeparator *(bool on, LineStyle data)*	
remove *(String propertyName)*	*void*

The method removes the property setting from the stylesheet. The value propertyName must be set to used property name from following list: 'fillBrush', 'fillShade', 'fillBlendAngle', 'horizontalSeparator', 'verticalSeparator'

Class: TableCellStyleSheet

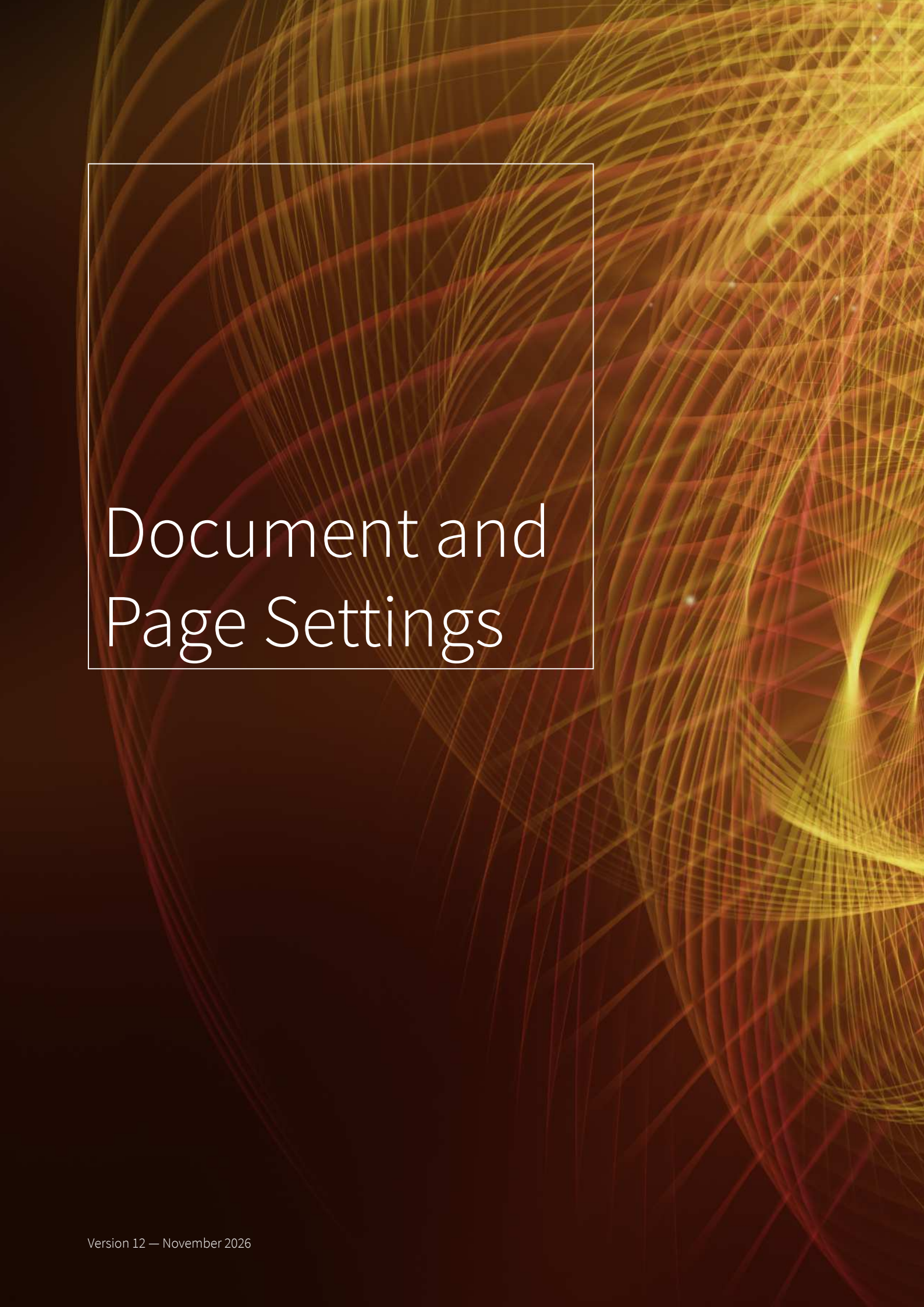
Properties:

fillBrush	*Brush*	*Read/Write*
fillShade	*float*	*Read/Write*
fillBlendAngle	*float*	*Read/Write*
horizontalAlignment	*HorizontalAlignment (enum)*	*Read/Write*
verticalAlignment	*VerticalAlignment (enum)*	*Read/Write*
rotation	*ContentRotation (enum)*	*Read/Write*
leftIndent	*UnitValueString*	*Read/Write*
topIndent	*UnitValueString*	*Read/Write*
rightIndent	*UnitValueString*	*Read/Write*
bottomIndent	*UnitValueString*	*Read/Write*
leftSeparator	*bool*	*Read*
leftSeparatorData	*LineStyle*	*Read*
topSeparator	*bool*	*Read*
topSeparatorData	*LineStyle*	*Read*
rightSeparator	*bool*	*Read*
rightSeparatorData	*LineStyle*	*Read*
bottomSeparator	*bool*	*Read*
bottomSeparatorData	*LineStyle*	*Read*

Methods:

setLeftSeparator *(bool on, LineStyle data)*	*void*
setTopSeparator *(bool on, LineStyle data)*	*void*
setRightSeparator *(bool on, LineStyle data)*	*void*
setBottomSeparator *(bool on, LineStyle data)*	*void*
remove *(String propertyName)*	*void*

The method removes the property setting from the stylesheet. The value propertyName must be set to used property name from following list: ‘fillBrush’, ‘fillShade’, ‘fillBlendAngle’, ‘rotation’, ‘horizontalAlignment’, ‘verticalAlignment’, ‘leftIndent’, ‘topIndent’, ‘rightIndent’, ‘bottomIndent’, ‘leftSeparator’, ‘topSeparator’, ‘rightSeparator’, ‘bottomSeparator’



Document and Page Settings

Version 12 — November 2026

Document and Page Settings

Class: DocumentSettings

An object holds the document parameters that are used when a new document is created. The property ***documentSettings*** of the class Document returns the settings of an open document; a property changed there changes the document at once.

Constructor:
An object can be created using the constructor:

```
new DocumentSettings()
```

and the document parameters are set using the following properties:

Properties:

pageSize	*PageSize*	*Read/Write*

Get/Set a page format of the document created. See the enumeration type PageSize.

pageOrientation	*PageOrientation*	*Read/Write*

Get/Set a page orientation of the document created. See the enumeration type PageOrientation.

pageType	*PageType*	*Read/Write*

Get/Set a page type of the document created. See the enumeration type PageType.

pageOrder	*PageOrder*	*Read/Write*

Get/Set a page order of the document created. See the enumeration type PageOrder.

customWidth	*UnitValueString*	*Read/Write*

Get/Set a width of the document page when for PageSize = 'Custom' is set.

customHeight	*UnitValueString*	*Read/Write*

Get/Set a height of the document page when for PageSize = 'Custom' is set.

automaticTextObject	*bool*	*Read/Write*

Get/Set a flag value. If the value is set to true, the document will be created with a text object according to the document settings.

layoutGrid	*bool*	*Read/Write*

Get/Set a flag value. If the value is set to true, the document will be created with a margin grid.

topMargin	*UnitValueString*	*Read/Write*

Get/Set a top margin for the new document page.

bottomMargin	*UnitValueString*	*Read/Write*

Get/Set a bottom margin for the new document page.

innerMargin	*UnitValueString*	*Read/Write*

Get/Set an inner margin for the new document page. For single pages it is the left margin: ***innerMargin*** and ***leftMargin*** are the same value.

outerMargin *UnitValueString* *Read/Write*

Get/Set an outer margin for the new document page. For single pages it is the right margin: ***outerMargin*** and ***rightMargin*** are the same value.

leftMargin *UnitValueString* *Read/Write*

Get/Set a left margin for the new document page. It is the same value as ***innerMargin***.

rightMargin *UnitValueString* *Read/Write*

Get/Set a right margin for the new document page. It is the same value as ***outerMargin***.

numberOfColumns *int* *Read/Write*

Get/Set a number of columns to be used on a new document page.

columnGutter *UnitValueString* *Read/Write*

Get/Set a gutter between the columns.

title *String* *Read/Write*

Get/Set the title of the file information (File > Document Setup). It is also written into exported PDF files.

description *String* *Read/Write*

Get/Set the description of the file information. It is also written into exported PDF files.

keywords *String* *Read/Write*

Get/Set the keywords of the file information. It is also written into exported PDF files.

language *Language* *Read/Write*

Get/Set the main language of the document. It is the language of exported PDF files. See the enumeration type Language.

Example 1:

```
var ds = new DocumentSettings();
ds.pageSize = PageSize.Custom;
ds.pageOrientation = PageOrientation.Landscape;
ds.pageType = PageType.FacingPages;
ds.pageOrder = PageOrder.RightLeft;
ds.customWidth = "150mm";
ds.customHeight = "20cm";
ds.automaticTextObject = true;
ds.layoutGrid = true;
ds.topMargin = "5mm";
ds.bottomMargin = "8mm";
messageBox.information("Msg:",
  convertUnitValue(ds.topMargin, "m", true));
```

Example 2:

```
var doc = application.currentDocument;
var ds = doc.documentSettings;
ds.title = "Annual Report 2026";
```

```
ds.keywords = "report, finance";
ds.language = Language.English_US;
```

Class: PageSettings

An object holds the page parameters that are used when new pages are created.

Constructor:
An object can be created using the constructor:

new **PageSettings()**
and the page parameters are set using the following properties:

Properties:

insertPosition *PageInsertPosition* *Read/Write*
Position for newly added pages. See the enumeration type ***PageInsertPosition***.

afterPageNumber *int* *Read/Write*
Index of page after which new page is inserted

copyObjects *bool* *Read/Write*
Objects are taken over from the current page

copyOnlySelectedObjects *bool* *Read/Write*
Selected objects are taken over from the current page only

copyAsAlias *bool* *Read/Write*
Objects are taken over as an Alias instead of as a Copy

linkToTextChain *bool* *Read/Write*
Text objects of new pages are linked to the text chain of the old page.

Class: ChapterInfo

The class is used in the class ***DocumentPage*** (property ***chapterInfo***).

Constructor:
new **ChapterInfo()**
new **ChapterInfo***(PageNumberingFormat pnk, short pageNumber, String prefix)*

Properties:

numberingFormat *PageNumberingFormat Read/Write*
Get/Set the numbering format of the page numbers, e.g. ***PageNumberingFormat.Arabic***, ***PageNumberingFormat.LeadingZeros2*** (01, 02, 03) or ***PageNumberingFormat.Abjad***. See the enumeration type ***PageNumberingFormat***.

newPageNumber *int* *Read/Write*
Get/Set the page number the chapter starts with.

prefix *String* *Read/Write*

Get/Set the prefix of the page numbers.

Class: DocumentStatistics

An object holds the statistics information of the document (Document Statistics). It is used in the class Document.

Constructor:
An object can be created using the constructor: *new* **DocumentStatistics***()* and the document parameters are set using the following properties or using the constructors with parameters:

```
new DocumentStatistics(
      PrintDeviceType printDeviceType,
      PrintMaterialType printMaterialType,
      PrintImageType printImageType,
      ExposureType exposureType,
      PrintResolutionType resolutionType,
      ColorSeparationState colorSeparationState)
new DocumentStatistics(
      PrintDeviceType printDeviceType,
      PrintMaterialType printMaterialType,
      PrintImageType printImageType,
      ExposureType exposureType,
      PrintResolutionType resolutionType,
      ColorSeparationState colorSeparationState,
      String scale,
      String author,
      String orderNumber,
      String orderedBy,
      String responsible,
      String phone,
      String fax,
      String deadline)
```

Properties:

printDeviceType	*PrintDeviceType*	*Read/Write*
printMaterial	*PrintMaterialType*	*Read/Write*
printImage	*PrintImageType*	*Read/Write*
exposure	*ExposureType*	*Read/Write*
printResolution	*PrintResolutionType*	*Read/Write*
colorSeparation	*ColorSeparationState*	*Read/Write*
scale	*String*	*Read/Write*
author	*String*	*Read/Write*
orderNumber	*String*	*Read/Write*
orderedBy	*String*	*Read/Write*

responsible	*String*	*Read/Write*
phone	*String*	*Read/Write*
fax	*String*	*Read/Write*
deadline	*String*	*Read/Write*
workingTime	*int*	*Read*

Class: SearchPaths

The object holds a list of search paths for the document. It is used in the class Document.

Constructor:
An object can be created using the constructor: ***new SearchPaths()***. The paths are added with the methods below; assign the object to the property ***searchPaths*** of the document.

Properties:

length	*int*	*Read*

Returns the number of search paths.

Methods:

addPath *(String path, bool withSubdir)*	*void*

The char '\' must be doubled.

removePath *(String path, bool withSubdir)*	*void*

The char '\' must be doubled.

removePath *(String path)*	*void*

The char '\' must be doubled.

removeAll()	*void*
exists *(String path, bool withSubdir)*	*bool*
exists (*String path)*	*bool*

The char'\' must be doubled.

pathAt *(int index)*	*String*
subdirFlagAt *(int index)*	*bool*

Examples:

```
var doc = application.currentDocument;
if (doc)
{
  var sp1 = new SearchPaths();
  sp1.addPath("c:\\Development", true);
  sp1.addPath("c:\\aaaa", true);
  sp1.addPath("c:\\aaaa", false);
  doc.searchPaths = sp1;
  application.currentDocument.searchPaths.addPath("c:\\xxxx", true);
```

```
  application.currentDocument.searchPaths.addPath("c:\\yyyy", false);
  application.currentDocument.searchPaths.addPath("c:\\zzzz", true);

  var sp = doc.searchPaths;
  messageBox.information("Len: ",
application.currentDocument.searchPaths.length);
  messageBox.information("PathAt: ", sp.pathAt(3));
  messageBox.information("FlagAt: ", sp.subdirFlagAt(0));
  var txt = "";
  for (var i = 0; i < application.currentDocument.searchPaths.length; i++)
{
    txt = txt + "\nIndex=" + i + "; Path=" +
    application.currentDocument.searchPaths.pathAt(i) + "; SubdirFlag=" +
    application.currentDocument.searchPaths.subdirFlagAt(i);
  }
  messageBox.information("List of paths", txt);

  if (sp.exists("c:\\xxxx"))
    sp.removePath("c:\\xxxx");

  var txt = "";
  for (var i = 0; i < application.currentDocument.searchPaths.length; i++)
{
    txt = txt + "\nIndex=" + i + "; Path=" +
    application.currentDocument.searchPaths.pathAt(i) + "; SubdirFlag=" +
      application.currentDocument.searchPaths.subdirFlagAt(i);
  }
  messageBox.information("List of paths", txt);

  doc.searchPaths.removeAll();
  messageBox.information("Search Paths Count: ", doc.searchPaths.length);
}
```

Preferences

Version 12 — November 2026

Preferences

Class: EmbeddedFontsPreferences

Properties:

disableSystemFonts	*bool*	*Read/Write*
embedAllUsedFonts	*bool*	*Read/Write*

Class: LanguageInfo

The class is used in the ***LanguagePreferences*** class.

Properties:

language	*Language*	*Read*
singleQuotesType	*SingleQuotesType*	*Read/Write*
doubleQuotesType	*DoubleQuotesType*	*Read/Write*

Get/Set the single and the double quotation marks of the language. See the enumeration types ***SingleQuotesType*** and ***DoubleQuotesType***.

hyphenationModuleIdent	*int*	*Read*
spellModuleIdent	*int*	*Read*
thesaurusModuleIdent	*int*	*Read*

The three properties return the identifiers of the hyphenation, spelling and thesaurus modules of the language.

Class: LanguagePreferences

The class is used in the Document class.

Methods:

languageInfo*(Language lang)* *LanguageInfo*

Returns the settings of the language ***lang***, or ***null*** for ***Language.NoLang*** and ***Language.Universal***.

setLanguageInfo*(LanguageInfo info) void*

Sets the settings of the language of ***info***.

Examples:

```
var doc = application.currentDocument;
if (doc) {
    var pref = doc.languagePreferences;
    var langInfo = pref.languageInfo(Language.German_DE);
    messageBox.information("MSG",
     "SpellModuleIdent: " + langInfo.spellModuleIdent +
```

```
 "\nDouble Quotes Type: " + langInfo.doubleQuotesType);
langInfo.doubleQuotesType = DoubleQuotesType.DoubleQuotes4;
pref.setLanguageInfo(langInfo);
doc.languagePreferences = pref;}
```

Class: LayoutPreferences

Properties:

showGuides	*bool*	*Read/Write*
showSmartGuides	*bool*	*Read/Write*
showRuler	*bool*	*Read/Write*
showTextRuler	*bool*	*Read/Write*
showGuideObjects	*bool*	*Read/Write*
showAliasObjects	*bool*	*Read/Write*
aliasObjectsSelectable	*bool*	*Read/Write*
showBaselineGrid	*bool*	*Read/Write*
guideObjectsMagnetic	*bool*	*Read/Write*
guideObjectsSelectable	*bool*	*Read/Write*
visualizeStyleSheets	*bool*	*Read/Write*
quickTextView	*bool*	*Read/Write*
quickPictureView	*bool*	*Read/Write*
viewZoomFactor	*float*	*Read/Write*
showMarginsAndColumns	*bool*	*Read/Write*

Get/Set whether the margins and columns of the pages are shown (View > Margins & Columns).

Class: ObjectPreferences

Properties:

snapDistance	*float*	*Read/Write*
graphicStyleSheet	*GraphicStyleSheet*	*Read/Write*
horizontalPasteOffset	*String*	*Read/Write*
verticalPasteOffset	*String*	*Read/Write*
askForImages	*bool*	*Read/Write*
colorDepth	*ColorDepth*	*Read/Write*
embedAllImages	*bool*	*Read/Write*
pictureStyleSheet	*PictureStyleSheet*	*Read/Write*

Class: PagePreferences

Properties:

saveSmallPreview	*bool*	*Read/Write*
saveLargePreview	*bool*	*Read/Write*
smallPreviewsMaximumSize	*int*	*Read/Write*
from 32 pixels to 640 pixel by step 16		
maximumPreviews	*int*	*Read/Write*
0 = all pages		
previewQuality	*int*	*Read/Write*
10 – 100 [%]		

Class: PreflightPreferences

The object holds the preflight preferences. It is used in the class ***Document***.

Constructor:
An object can be created using the constructor:

```
new PreflightPreferences()
```

and the document parameters are set using the following properties or using the constructor with parameters:

```
new PreflightPreferences(
   preflightDocument = false,
   colorImagesMinimumResolution = 0,
   grayImagesMinimumResolution = 0,
   bwImagesMinimumResolution = 0,
   warningForRgbColorMode = false,
   warningForGrayColorMode = false,
   warningForBwColorMode = false,
   warningForTransparency = false,
   warningForEffects = false,
   tolerance = 30)
```

Properties:

preflightDocument	*bool*	*Read/Write*
colorImagesMinimumResolution	*float*	*[ppi] Read/Write*
grayImagesMinimumResolution	*float*	*[ppi] Read/Write*
bwImagesMinimumResolution	*float*	*[ppi] Read/Write*

All three properties get/set the resolution value in ppi.

To disable the option set 0 option.

warningForRgbColorMode	*bool*	*Read/Write*

warningForGrayColorMode	*bool*	*Read/Write*
warningForBwColorMode	*bool*	*Read/Write*
warningForTransparency	*bool*	*Read/Write*
warningForEffects	*bool*	*Read/Write*
tolerance	*float*	*<0-100>[%] Read/Write*

Class: TextPreferences

The object holds the text preferences. The class is used in class ***Document***.

Constructor:
An object can be created using the constructor: ***new TextPreferences()*** and the document parameters are set using the following properties:

Properties:

underlineOffsetVerticalLayout *float* *Read/Write*
unit: [%], valid values interval: [-100, 100], default: 60%

underlineOffsetHorizontalLayout *float* *Read/Write*
unit: [%], valid values interval: [-100, 100], default: 10%

underlineLineWidth *float* *Read/Write*
unit: [%], valid values interval: [1, 100], default: 5%

strikethroughOffsetVerticalLayout *float Read/Write*
unit: [%], valid values interval: [-100, 100], default: 0%

strikethroughOffsetHorizontalLayout *float Read/Write*
unit: [%], valid values interval: [-100, 100], default: -30%

strikethroughLineWidth *float* *Read/Write*
unit: [%], valid values interval: [5, 100], default: 5%

superscriptOffset *float* *Read/Write*
unit: [%], valid values interval: [0, 100], default: 33%

superscriptCharacterHeight *float* *Read/Write*
unit: [%], valid values interval: [1, 100], default: 50%

superscriptCharacterWidth *float* *Read/Write*
unit: [%], valid values interval: [1, 100], default: 50%

subscriptOffset *float* *Read/Write*
unit: [%], valid values interval: [0, 100], default: 33%

subscriptCharacterHeight *float* *Read/Write*
unit: [%], valid values interval: [1, 100], default: 50%

subscriptCharacterWidth	*float*	*Read/Write*
unit: [%], valid values interval: [1, 100], default: 50%		
superiorCharacterHeight	*float*	*Read/Write*
unit: [%], valid values interval: [1, 100], default: 50%		
superiorCharacterWidth	*float*	*Read/Write*
unit: [%], valid values interval: [1, 100], default: 50%		
smallCapsCharacterHeight	*float*	*Read/Write*
unit: [%], valid values interval: [1, 100], default: 75%		
smallCapsCharacterWidth	*float*	*Read/Write*
unit: [%], valid values interval: [1, 100], default: 75%		
lineNumbersFont	*FontName*	*Read/Write*
get/set name of the font		
lineNumbersFontSize	*UnitValueString*	*Read/Write*
get/set font size		
lineNumbersColor	*ColorBrush*	*Read/Write*
get/set a color of LineNumbers		
lineNumbersMode	*LineNumbersMode*	*Read/Write*
get/set mode of LineNumbers		
lineNumbersData	*LineNumbers*	*Read/Write*
get/set LineNumbers data which depends on LineNumbersMode		
textLinesAutomaticSpacing	*float*	*Read/Write*
unit: [%], valid values interval: [0, 100], default: 0%		
textLinesMethod	*AutoLineSpacingMethod*	*Read/Write*
language	*Language*	*Read/Write*
greekBelow	*UnitValueString*	*Read/Write*
automaticGrammarCheck	*bool*	*Read/Write*
Returns the same value as ***automaticSpellCheck***; setting it has no effect.		
automaticSpellCheck	*bool*	*Read/Write*
showInvisibles	*bool*	*Read/Write*
showTextRuler	*bool*	*Read/Write*
distanceBetweenStandardTabs	*UnitValueString*	*Read/Write*
typographicQuotationMarks	*bool*	*Read/Write*
forceLineBreak	*bool*	*Read/Write*
visualizeStyleSheets	*bool*	*Read/Write*

firstBaseline	*BaselinePosition*	*Read/Write*
baselineGrid	*BaselineGrid*	*Read/Write*
characterStyleSheet	*CharacterStyleSheet*	*Read/Write*

Set/get the default character style sheet for the text. The object CharacterStyleSheet is used to identify the stylesheet.

characterStyleSheetByName	*String*	*Read/Write*

Set/get the default character style sheet. Name of style sheet is used for identify of the style sheet.

paragraphStyleSheet	*ParagraphStyleSheet*	*Read/Write*
paragraphStyleSheetByName	*String*	*Read/Write*
layoutStyleSheet	*LayoutStyleSheet*	*Read/Write*
layoutStyleSheetByName	*String*	*Read/Write*
footnoteReferenceStyleSheet	*CharacterStyleSheet*	*Read/Write*
footnoteLabelStyleSheet	*CharacterStyleSheet*	*Read/Write*
footnoteTextStyleSheet	*ParagraphStyleSheet*	*Read/Write*
footnoteReferenceStyleSheetByName	*String*	*Read/Write*
footnoteLabelStyleSheetByName	*String*	*Read/Write*
footnoteTextStyleSheetByName	*String*	*Read/Write*
endnoteReferenceStyleSheet	*CharacterStyleSheet*	*Read/Write*
endnoteLabelStyleSheet	*CharacterStyleSheet*	*Read/Write*
endnoteTextStyleSheet	*ParagraphStyleSheet*	*Read/Write*
endnoteReferenceStyleSheetByName	*String*	*Read/Write*
endnoteLabelStyleSheetByName	*String*	*Read/Write*
endnoteTextStyleSheetByName	*String*	*Read/Write*
changeTracking	*bool*	*Read/Write*
endnotes	*TextEndnotes*	*Read/Write*
footnotes	*TextFootnotes*	*Read/Write*

Example 1:

```
application.currentDocument.textPreferences.underlineOffsetVerticalLayout =
77;
application.currentDocument.textPreferences.baselineGrid.start = "13pt";
messageBox.information("UnderlineOffsetVerticalLayout",
   application.currentDocument.textPreferences.underlineOffsetVerticalLayout
+ "\n" +
   application.currentDocument.textPreferences.baselineGrid.start);
```

Example 2:

```
var doc = application.currentDocument;
```

```
if (doc) {
    var tp = new TextPreferences();
    tp.underlineOffsetVerticalLayout = 33;
    tp.lineNumbersFont = "Times New Roman-Bold";
    tp.lineNumbersFontSize = "12pt";
    var cb = doc.brushes.addColor("MyColor2");
    cb.rgb = new Rgb(200, 100, 0);
    tp.lineNumbersColor = cb;
    var blg = new BaselineGrid("22pt", "12pt",
BaselineGridNumbering.ShowAllNumbers,
  BaselineGridDirection.RightToLeft);
    tp.baselineGrid = blg;

    doc.textPreferences = tp;

    var tp1 = doc.textPreferences;
    var txt = "UnderlineOffsetVertLayout = " +
tp1.underlineOffsetVerticalLayout + "\nUnderlineOffsetHorizontalLayout = " +
tp1.underlineOffsetHorizontalLayout;

    messageBox.information("TextPreferences", txt)
}
```



Helped Data Objects

Version 12 — November 2026

Helped Data Objects

Class: Annotation

The object of the class ***Annotation*** represents a ruby annotation in a ***FormattedText***: a short annotation text above, below or beside its base text. ***FormattedText.annotation()*** returns it, ***FormattedText.insertAnnotation()*** creates it.

Properties:

position	*RubyPosition*	*Read*

Returns the value of the enumeration type ***RubyPosition***.

range	*TextRange*	*Read*

Returns a ***TextRange*** object with the position where the base text (where the annotation text is applied) and the annotation text are placed together.

baseTextRange	*TextRange*	*Read*

Returns a ***TextRange*** object with the position where only the base text (where the annotation text is applied) is placed.

annotationTextRange	*TextRange*	*Read*

Returns a ***TextRange*** object with the position where only the annotation text is placed.

mainText	*FormattedText*	*Read*

Returns all text after the position where the annotation text was applied.

Examples:

```
var doc = application.currentDocument;
if (doc) {
  var objP = doc.objects().create(ObjectType.Text);
  objP.pagex = "10mm";
  objP.pagey = "22mm";
  objP.width = "120mm";
  objP.height = "40mm";

  with (objP.content) {
    insertPlainText("Sample text for testing.");
    formattedText.insertAnnotation("AnnotationText",
      new TextRange(5, 10),
      RubyPosition.Below);
  }
  doc.currentPage.add(objP);
  var pc = doc.currentPage.objects().at(0).content;
  var txt = "";
  var t = pc.formattedText.annotation(5);

  txt = txt + "Position: " + t.position + "\n";
  txt = txt + "Range: [" + t.range.begin + "," + t.range.end + "; Text: " +
        t.mainText.formattedText(t.range.begin, t.range.end).plainText()
```

```
        + "]\n";
   txt = txt + "BaseTextRange: [" + t.baseTextRange.begin + "," +
     t.baseTextRange.end + "; Text: " +
t.mainText.formattedText(t.baseTextRange.begin,
t.baseTextRange.end).plainText() + "]\n";
   txt = txt + "AnnotationTextRange: [" + t.annotationTextRange.begin + ","
+         t.annotationTextRange.end + "; Text: " +
t.mainText.formattedText(t.annotationTextRange.begin,
t.annotationTextRange.end).plainText() + "]\n";
   txt = txt + "Text: " + t.mainText.plainText() + "\n";

   messageBox.information("TextInfo", txt);
}
```

Class: Attributes

The class is used in the classes *Object*, *PictureContent*, *TextContent*, *Table*, *TableCell*, *TableColumn*, *TableRow* and *TableSeparator*.

Constructor:

new **Attributes()**

Class: AutomaticCharacterSpacing

The class is used in the class *FormattedText*, *ParagraphStyleSheet*.

Constructor:

new **AutomaticCharacterSpacing()**

Properties:

active	*bool*	*Read/Write*
minimum	*float*	*Read/Write [%]*
optimum	*float*	*Read/Write [%]*
maximum	*float*	*Read/Write [%]*

Class: AutomaticCharacterWidth

The class is used in the class *FormattedText*, *ParagraphStyleSheet*.

Constructor:

new **AutomaticCharacterWidth()**

Properties:

minimum	*float*	*Read/Write [%]*
optimum	*float*	*Read/Write [%]*
maximum	*float*	*Read/Write [%]*

Class: AutomaticWordSpacing

The class is used in the class *FormattedText*, *ParagraphStyleSheet*.

Constructor:

new **AutomaticWordSpacing()**

Properties:

minimum	*float*	*Read/Write [%]*
optimum	*float*	*Read/Write [%]*
maximum	*float*	*Read/Write [%]*

Class: BaselineGrid

The class is used in the class *TextPreferences*.

Constructor:

new **BaselineGrid()**
new **BaselineGrid***(String start, String spacing, BaselineGridDirection direction, BaselineGridNumbering numbering)*

Properties:

active	*bool*	*Read/Write*
start	*UnitValueString*	*Read/Write*
spacing	*UnitValueString*	*Read/Write*
direction	*BaselineGridDirection*	*Read/Write*
numbering	*BaselineGridNumbering*	*Read/Write*

Class: BaselinePosition

The class is used in the classes TextPreferences and TextContent.

Constructor:

new **BaselinePosition**()
new **BaselinePosition**(*BaselineType firstBaseline, BaselineOffset applyValue, float value*)

Properties:

active	*bool*	*Read/Write*
firstBaseline	*BaselineType*	*Read/Write*
Get/Set value of the BaselineType enumeration type.		
applyValue	*BaselineOffset*	*Read/Write*
Get/Set value of the BaselineOffset enumeration type.		
value	*UnitValueString*	*Read/Write*

Get/Set offset value.

Examples:

```
var objectcontent = ...; //get some object content
var bp = new BaselinePosition();
bp.firstBaseline = BaselineType.AscentHeight;
bp.applyValue = BaselineOffset.Fixed;
bp.value = "10pt";
objectcontent.baselinePosition = bp;

var bp1 = objectcontent.baselinePosition;
bp1.value = 15;
  bp1.firstBaseline = BaselineType.LineSpacing;
bp1.applyValue = BaselineOffset.Fixed;
bp1.value = "20pt";

var bp2 = new BaselinePosition(
     BaselineType.CapHeight, BaselineOffset.Fixed, "2cm");
objectcontent.baselinePosition = bp2;
```

Class: ColorOverlay

Constructor:

new **ColorOverlay()**

Properties:

active	*bool*	*Read/Write*		
color	*ColorBrush*	*Read/Write*		
shade	*float*	*Read/Write*	*[%]*	*<0,100>*
opacity	*float*	*Read/Write*	*[%]*	*<0,100>*
mode	*ColorMode (enum)*	*Read/Write*		

Class: ColumnLines

The class is used in the class GraphicStyleSheet.

Constructor:

new **ColumnLines()**

new **ColumnLines**(*ColorBrush lineColor, float lineShade, float lineOpacity,*
String lineWidth, String lineStyle,
String topIndent, String bottomIndent,
TextColumnLineAnchor topAnchor, TextColumnLineAnchor bottomAnchor)

Properties:

active	*bool*	*Read/Write*
lineColor	*ColorBrush*	*Read/Write*
lineShade	*float*	*Read/Write*

lineOpacity	*float*	*Read/Write*
lineWidth	*String*	*Read/Write*
lineStyle	*String*	*Read/Write*
topIndent	*String*	*Read/Write*
bottomIndent	*String*	*Read/Write*
topAnchor	*TextColumnLineAnchor*	*Read/Write*
bottomAnchor	*TextColumnLineAnchor*	*Read/Write*

Class: Comment

The class is used in the class FormattedText.

Constructor:

new **Comment()**

Properties:

text	*String*	*Read/Write*
author	*String*	*Read/Write*
creationDate	*String*	*Read/Write*

Format: “dd.MM.yyyy hh:mm:ss” is expected.

modificationDate	*String*	*Read/Write*

Format: “dd.MM.yyyy hh:mm:ss” is expected.

valid	*bool*	*Read*

Returns ***true*** if the object holds a comment. ***FormattedText.comment()*** returns an object with ***valid*** = ***false*** where no comment is placed. A new object becomes valid when ***creationDate*** is set.

Class: DropCaps

The feature DropCaps sets the drop caps at the beginning of each paragraph that is placed in the text range. The class is used in the classes ***FormattedText*** and ***ParagraphStyleSheet***.

Constructor:

new **DropCaps()**
new **DropCaps**(*DropCapsType mode*)

Properties:

active	*bool*	*Read*
type	*DropCapsType*	*Read/Write*

The feature DropCaps sets the drop caps at the beginning of each paragraph that is placed in the text range.

linesType	*DropCapsLinesType*	*Read*

Get value of the DropCapsLinesType enumeration type.

minimumLines *int* *Read*

Defines the minimum number of lines for the DropCaps feature. It works for ***DropCapsLinesType.FixedLineHeight*** only.

maximumLines *int* *Read*

Defines the maximum number of lines for the DropCaps feature.

characterCount *int* *Read/Write*

Defines the number of characters used for DropCaps. It works for ***DropCapsType.Sequence*** only.

runaroundType *DropCapsRunaroundType* *Read/Write*

Defines the type of runaround.

distanceToText *UnitValueString* *Read/Write*

Defines distance from the DropCaps text to the following text.

scale *float* *Read/Write*

Defines the scale of the DropCaps text.

Methods:

setFixLines *(int minimum, int maximum)* *void*

Set drop caps features. The `DropCapsLinesType = FixedLineHeight` and set the minimum and the maximum.

setDynamicLines *(int maximum)* *void*

Set drop caps features. The ***DropCapsLinesType = DynamicLineHeight*** and set the maximum.

Examples:

```
var ft = … //set FormattedText object
var dc1 = new DropCaps();
dc1.type = DropCapsType.Sequence;
dc1.setFixLines(4,8);
dc1.characterCount = 2;
dc1.runaroundType = DropCapsRunaroundType.RunaroundBox;
dc1.distanceToText = "13mm";
dc1.scale = 60;
messageBox.information("MSG", ft.dropCaps(30).type);
ft.setDropCaps(dc1, new TextRange(3, 50));
messageBox.information("MSG", ft.dropCaps(30).type);
//clear drop caps
ft.setDropCaps(new DropCaps(DropCapsType.None), 3, 50);
messageBox.information("MSG", ft.dropCaps(30).type)
```

Class: Enumerations

The feature is named "Bullets & Numbering" in the VivaDesigner interface. Defines features of numbering paragraphs in the selected text. The text related to Enumerations object parameters is added before each paragraph.

The class is used in class FormattedText, ParagraphStyleSheet.

Constructor:

new **Enumerations()**

The constructor for EnumerationsType = Numbering is the following:

```
new Enumerations(EnumerationsNumberingMode mode,
   int startWith, int level,
   EnumerationsAlignment justification,
   EnumerationsNumberFormat format,
   bool tabulator,
   String prefix, String postfix,
   String characterStyleSheet)
```

The constructor for EnumerationsType = **Bullets**

```
new Enumerations(BulletCharacters bulletType,
   Char bullet, (insignificant for bulletType != Custom)
   int level,
   EnumerationsAlignment justification,
   bool tabulator,
   String prefix, String postfix,
   String characterStyleSheet)
```

Properties:

on	*bool*	*Read*

Returns ***true*** if bullets or numbering are switched on (***type*** is not ***EnumerationsType.Off***).

type	*EnumerationsType*	*Read/Write*
mode	*EnumerationsNumberingMode*	*Read/Write*
startWith	*int*	*Read/Write (1-)*
level	*int*	*Read/Write (1-10)*
alignment	*EnumerationsJustification*	*Read/Write*
bullet	*Char*	*Read*

Returns the bullet character. To set it, use the method ***setBullet()***.

prefix	*String*	*Read/Write (maxlen=10)*
postfix	*String*	*Read/Write (maxlen=10)*
tabulator	*bool*	*Read/Write*
characterStyleSheetByName	*String*	*Read/Write*
characterStyleSheet	*CharacterStyleSheet*	*Read/Write*
numberFormat	*EnumerationsNumberFormat*	*Read/Write*

Get/Set the number format of a numbering: Arabic numbers (also with two, three or four digits), letters, Roman numbers, Arabic letters in abjad or alphabetical order, Hebrew numerals, or no number (*Format_None*). See the enumeration type *EnumerationsNumberFormat*. *Format_Asterisks* and *Format_Symbols* are meant for notes.

Methods:

setBullet *(BulletCharacters type, Char customChar)* *void*

Sets the bullet character: one of the characters of the enumeration type *BulletCharacters*, or with *BulletCharacters.Custom* the character *customChar*. *customChar* is only needed for *BulletCharacters.Custom*.

Examples:

```
  var ft = … // set formatted text

  //set empty constructor
  var en = new Enumerations();
  en.mode = EnumerationsNumberingMode.BeginNew;
  en.startWith = 3;
  en.level = 2;
  en.alignment = EnumerationsAlignment.Center;
  en.setBullet(BulletCharacters.MiddleDot);
  en.prefix = "AAAA";
  en.postfix = "BBBB"
  en.tabulator = true;
  en.characterStyleSheetByName = "MyCharacter";
  en.numberFormat = EnumerationsNumberFormat.Format_i_ii_iii;

    //constructor for EnumerationsType.Numbering
  var en2 = new Enumerations(EnumerationsNumberingMode.BeginNew, 3, 2,
      EnumerationsAlignment.Center,
EnumerationsNumberFormat.Format_i_ii_iii, true,
      "AAA", "BBB", "MyCharacter");

  //constructor for EnumerationsType.Bullets
  var en3 = new Enumerations(BulletCharacters.MiddleDot, 'a', 2,
    EnumerationsAlignment.Center, true,
      "AAA", "BBB", "MyCharacter");

  ft.setEnumerations(EnumerationsType.Numbering, en2, 4, 30);
  ft.setEnumerations(EnumerationsType.Bullets, en3, 60);

  messageBox.information("MSG", "Type: " + ft.getEnumerationsType(25));
  var en4 = ft.getEnumerationsData(25);
  messageBox.information("MSG", "Data: StartWith, Tabulator: " +
          en4.startWith + "," + en4.tabulator);
  messageBox.information("MSG", "Data: StartWith, Tabulator: " +
          ft.getEnumerationsData(25).startWith + "," +
ft.getEnumerationsData(25).tabulator);
```

Class: FontSize

The class is used in the classes *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:

new **FontSize**(*float size*)
new **FontSize**(*float size, FontReferenceSizeType type*)

Properties:

type	*FontReferenceSizeType*		*Read/Write*
size	*float*	*Read/Write*	

Class: GraphicFootnotes

The class is used in the classes GraphicStyleSheet and TextContent.

Constructor:

new **GraphicFootnotes()**

Properties:

position	*FootnotesPosition*	*Read/Write*
columnCount	*int*	*Read/Write*
columnDistance	*String*	*Read/Write*
startColumn	*int*	*Read/Write*
width	*int*	*Read/Write*

Class: HeaderFooter

The class is used in the classes GraphicStyleSheet and TextContent.

Constructor:

new **HeaderFooter()**

Properties:

active	*bool*	*Read/Write*
height	*UnitValueString*	*Read/Write*

Class: Hyphenation

The class is used in the classes ParagraphStyleSheet and FormattedText.

Constructor:

```
new Hyphenation()
new Hyphenation(short smallestWord,
      short maxHyphensInARow,
      short minPrefix,
      short minSuffix,
```

HyphenationQuality quality)

Properties:

active	*bool*	*Read/Write*
smallestWord	*int*	*Read/Write*
maxHyphensInARow	*int*	*Read/Write*
minPrefix	*int*	*Read/Write*
minSuffix	*int*	*Read/Write*
quality	*HyphenationQuality*	*Read/Write*

Examples:

```
  var ft = … // get formatted text
  var hyp = new Hyphenation();
  hyp.active = true;
  hyp.smallestWord = 10;
  hyp.maxHyphensInARow = 11;
  hyp.minPrefix = 12;
  hyp.minSuffix = 13;
  hyp.quality = HyphenationQuality.GoodQuality;

  var hyp2 = new Hyphenation(20, 21, 22, 23,
HyphenationQuality.StandardQuality);
  ft.setHyphenation(true, hyp);
  ft.setHyphenation(true, hyp2, 10, 20);

  messageBox.information("Msg:", "Hyphenation IsOn : " + ft.hyphenation());
  var hyp1 = ft.hyphenationData(15);
  messageBox.information("Msg:", "Hyphenation: On: " + hyp1.active);
  messageBox.information("Msg:", "Hyphenation: smallestWord: " +
              hyp1.smallestWord);

  ft.setHyphenation(false);
```

Class: ChangePictureColorSpaceSettings

The class holds settings for the picture color space changing.

Constructor:

new **ChangePictureColorSpaceSettings();**

Properties:

format	*PictureFormat (enum)*	*Read/Write*

Valid only if method outFilePath is empty.

colorSpace	*ColorSpace(enum)*	*Read/Write*
colorProfile	*String*	*Read/Write*
prefix	*String*	*Read/Write*

postfix	*String*	*Read/Write*
outFilePath	*String*	*Read/Write*

Embed the image if empty.

Class: CharacterBackground

The object of the class holds the parameters of the character background.

The class is used in classes FormattedText, CharacterStyleSheet and ParagraphStyleSheet.

Constructor:

new **CharacterBackground** *()*
new **CharacterBackground** *(bool active)*
ColorBrush brush, float shade, float opacity,
String leftIndent, String rightIndent,
String topIndent, String bottomIndent)

Properties:

active	*bool*	*Read/Write*
brush	*ColorBrush*	*Read/Write*
shade	*float*	*Read/Write*
opacity	*float*	*Read/Write*
leftIndent	*UnitValueString*	*Read/Write*
rightIndent	*UnitValueString*	*Read/Write*
topIndent	*UnitValueString*	*Read/Write*
bottomIndent	*UnitValueString*	*Read/Write*

Examples:

```
var ft = … // get formatted text

var chb = new CharacterBackground();
chb.brush = Color.Red;
chb.shade = 34;
chb.opacity = 56;
chb.topIndent = "1pt";
chb.leftIndent = "0pt";
chb.rightIndent = "0pt";
chb.bottomIndent = "2pt";

var chb2 = new CharacterBackground( true, Color.Green,
          100, 99, "0pt", "0pt", "-3mm", "-8mm");
ft.setCharacterBackground(true, chb2, new TextRange(10, 34));
ft.setCharacterBackground(true, chb, new TextRange(15, 25));
var chb1 = ft.characterBackgroundData(16);
var txt = "Brush: " + chb1.brush.uniqueName + "\nShade: " +
          chb1.shade + "\nOpacity: " + chb1.opacity;
messageBox.information("CharacterBackground", txt);
```

```
ft.setCharacterBackground(false, chb, 28, 30);
```

Class: CharacterFrame

The object of the class holds the parameters of the character frame.

The class is used in classes FormattedText, CharacterStyleSheet and ParagraphStyleSheet.

Constructor:

new **CharacterFrame()**
new **CharacterFrame**(*bool active,*
ColorBrush brush, float shade, float opacity,
UnitValueString leftIndent, UnitValueString rightIndent,
UnitValueString topIndent, UnitValueString bottomIndent,
UnitValueString frameWidth, String lineStyle)

Properties:

active	*bool*	*Read/Write*
brush	*ColorBrush*	*Read/Write*
shade	*float*	*Read/Write*
opacity	*float*	*Read/Write*
leftIndent	*UnitValueString*	*Read/Write*
rightIndent	*UnitValueString*	*Read/Write*
topIndent	*UnitValueString*	*Read/Write*
bottomIndent	*UnitValueString*	*Read/Write*
frameWidth	*UnitValueString*	*Read/Write*
lineStyle	*LineStyleString*	*Read/Write*

Examples:

```
var ft = … // get formatted text

var chf = new CharacterFrame();
chf.active = true;
chf.brush = Color.Red;
chf.shade = 34;
chf.opacity = 56;
chf.frameWidth = "1pt";
chf.lineStyle = "SolidLine";
chf.topIndent = "0mm";
chf.leftIndent = "0mm";
chf.rightIndent = "0mm";
chf.bottomIndent = "0mm";
ft.setCharacterFrame(true, chf, 10, 11);

var chf1 = new CharacterFrame(true, Color.Green,
     55, 77, "2pt", "3pt", "1pt", "-8pt", "2pt", "DashDotLine");
ft.setCharacterFrame(chf1, new TextRange(15, 30));
```

```
ft.setCharacterFrame(chf, new TextRange(18, 20));
```

Class: CharacterKerning

The class is used in the class *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:

new **CharacterKerning()**

Properties:

type	*KerningType*	*Read/Write (enum)*
value	*float*	*Read/Write*

Class: CharacterPositionCorrection

The class is used in the class FormattedText.

Constructor:

new **CharacterPositionCorrection()**
new **CharacterPositionCorrection**(*int horizontal, int vertical*)

Properties:

horizontalCorrection	*int*	*Read/Write*
verticalCorrection	*int*	*Read/Write*

Examples:

```
var ft = … // get formatted text
var cps = new CharacterPositionCorrection();
cps.horizontalCorrection = 35;
cps.verticalCorrection = -15;
ft.setPositionCorrection(cps, 10);
messageBox.information(“MSG”,
ft.positionCorrection(10).verticalCorrection);
```

Class: CharacterRule

The object of the class holds parameters of the character rule function for the text.

The class is used in classes *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:

new **CharacterRule()**
new **CharacterRule**(*bool active, ColorBrush brush, float shade, float opacity,*
UnitValueString lineWidth, String lineStyle, UnitValueString offset,
UnitValueString leftIndent, UnitValueString rightIndent)

Properties:

active	*boolean*	*Read/Write*

brush	*ColorBrush*	*Read/Write*
shade	*float*	*Read/Write*
opacity	*float*	*Read/Write*
lineWidth	*UnitValueString*	*Read/Write*
lineStyle	*LineStyleString*	*Read/Write*
offset	*UnitValueString*	*Read/Write*
leftIndent	*UnitValueString*	*Read/Write*
rightIndent	*UnitValueString*	*Read/Write*

Examples:

```
  var ft = … // get formatted text
  var chr1 = new CharacterRule( true, Color.Green, 60, 50, "2pt",
"SolidLine",
              "0mm", "0mm", "0mm");
  var chr = new CharacterRule();
  chr.active = true;
  chr.brush = Color.Magenta;
  chr.shade = 91;
  chr.opacity = 82;
  chr.lineWidth = "3mm";
  chr.lineStyle = "DotLine";
  chr.offset = "2mm";
  chr.leftIndent = "1mm";
  chr.rightIndent = "1mm";
  ft.setCharacterRule(chr, new TextRange(10, 20));
  ft.setCharacterRule(chr, 15, 17);
  ft.setCharacterRule(chr1, new TextRange(25,35));
```

Class: Image

The object of the class holds a bitmap picture (its pixels).

The class is used in the class ***PictureContent*** (properties ***image*** and ***preview***).

Constructor:

new **Image()**
new **Image(String data)**

Properties:

data	String	*Read*

Returns the picture as a JSON string with the keys ***width***, ***height***, ***bytesPerLine***, ***format***, ***dotsPerMeterX***, ***dotsPerMeterY***, ***devicePixelRatio*** and ***data*** (the pixel bytes, Base64 encoded). An empty picture returns an empty string.

width	*int*	*Read*
height	*int*	*Read*

format	*ImageFormat*	*Read*

Methods:

setData(*String data*)	*void*

Sets the picture from a JSON string in the format of ***data***. If the values do not fit together (format, size, bytes per line and the length of the pixel data), the picture stays unchanged and ***errorStatus*** is ***IsNot-Valid***.

Class: InnerGlow

The class is used in the class GraphicObject for all objects and in the class ***GraphicStyleSheet***. The class holds the parameters used for the "Inner Glow" effect of the object.

Constructor:

new **InnerGlow()**

Properties:

active	*bool*	*Read/Write*		
color	*ColorBrush*	*Read/Write*		
shade	*float*	*Read/Write*	*[%]*	*<0-100>*
opacity	*float*	*Read/Write*	*[%]*	*<0-100>*
mode	*ColorMode (enum)*	*Read/Write*		
blur	*UnitValueString*	*Read/Write [mm]*	*<0,20>*	

Class: LayoutColumnsInfo

The class is used in the classes ***LayoutStyleSheet*** and ***FormattedText***.

Constructor:

new **LayoutColumnsInfo()**

Properties:

count	*int*	*Read*

Methods:

setCount *(int n, UnitValueString margin)*	*void*
setWidthAt *(int index, UnitValueString width)*	*void*
widthAt *(int index)*	*UnitValueString*
setMarginAt *(int index, UnitValueString margin)*	*void*
marginAt *(int index)*	*UnitValueString*

Examples:

```
var ft = … // get formatted text
```

```
  var lci = new LayoutColumnsInfo();

  // set columns for type "Automatic"
  lci.setCount(4, "5mm");

  // set columns for type "Manual"
  lci.setWidthAt(0, "33pt");
  lci.setMarginAt(0, "2pt");
  lci.setWidthAt(1, "50pt");
  lci.setMarginAt(1, "5pt");
  ft.setColumns(lci, new TextRange(10, 15));

  var lci1 = ft.columnsData(12);
  var txt = "Width(1)=" + lci1.widthAt(1) + "; Margin(1)=" +
lci1.marginAt(1) +      "; Count=" + lci1.count;
  messageBox.information("MSG", txt);
```

Class: LayoutFootnotes

The class is used in the classes *LayoutStyleSheet and FormattedText*.

Constructor:

new **LayoutFootnotes()**

Properties:

mode	*LayoutFootnotesMode*	*Read/Write*
columnCount	*int*	*Read/Write*
margin	*UnitValueString*	*Read/Write*
placeStartColumn	*int*	*Read/Write*
placeMaximumColumnCount	*int*	*Read/Write*

Examples:

```
  var ft = … // get formatted text

  var fn = new LayoutFootnotes();
  fn.columnCount = 4;
  fn.margin = "1cm";
  fn.placeStartColumn = 2;
  fn.placeMaximumColumnCount = 3;
  ft.setLayoutFootnotes(LayoutFootnotesMode.LayoutDependent, fn,
         new TextRange(10, 20));

  var fn1 = ft.layoutFootnotesData(15);
  var info = "LayoutFootnotes: \n" +
     "  Mode: " + fn1.mode + "\n" +
     "  Column Count: " + fn1.columnCount + "\n" +
     "  Margin: " + fn1.margin + "\n" +
     "  PlaceStartColumn: " + fn1.placeStartColumn + "\n" +
     "  PlaceMaximumColumnCount: " + fn1.placeMaximumColumnCount + "\n";
  messageBox.information("MSG", info);
```

Class: LineStyle

The class is used in the classes *TableCellStyleSheet*, *TableColumnStyleSheet*, *TableRowStyleSheet*.

Constructor:

```
new LineStyle()
new LineStyle(ColorBrush color, float shade,
              String lineStyle, String lineWidth)
```

Properties:

active	*bool*	*Read*
lineBrush	*ColorBrush*	*Read/Write*
lineShade	*float*	*Read/Write*
lineStyle	*LineStyleString*	*Read/Write*
Returns LineStyle name.		
lineWidth	*UnitValueString*	*Read/Write*
Returns line width in [pt].		

Class: LineNumbers

The class represents Line Counter data. It is used in classes *TextPreferences and FormattedText* .

Constructor:

```
new LineNumbers()

new LineNumbers (LineNumbersMode mode,
   int startsWith, int step,
   UnitValueString offset,
   LineNumbersRestartType restartType = LineNumbersRestartType.Continue,
   bool supressEmptyParagraphs)
```

Properties:

on	*bool*	*Read*
mode	*LineNumbersMode*	*Read/Write*
startsWith	*int*	*Read/Write*
step	*int*	*Read/Write*
offset	*UnitValueString*	*Read/Write*
restartType	*LineNumbersRestartType*	*Read/Write*
supressEmptyParagraphs	*bool*	*Read/Write*

Class: Link

The class is used in the class *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:

```
new Link()

new Link (bool active,
   String documentName,
   LinkAction action,
   int pageNumber=1)
```

Properties:

active	*bool*	*Read/Write*

Get/Set whether the text is linked to a page: ***true*** sets ***destination*** to ***LinkDestination.Page***, ***false*** to ***LinkDestination.None***. Use ***destination*** instead.

documentName	*String*	*Read/Write*

Get/Set the target of the link: for ***LinkDestination.Page*** the name of the target document (empty for the own document), for ***LinkDestination.Url*** the URL, for example ***"https://www.viva.de"***.

linkAction	*LinkAction*	*Read*

Returns the action of a page link (***LinkDestination.Page***). See the enumeration type ***LinkAction*** and the method ***setAction()***.

pageNumber	*int*	*Read*

Returns the page number of a page link with the action ***LinkAction.GotoPageNumber***.

destination	*LinkDestination*	*Read/Write*

Get/Set the kind of link: ***LinkDestination.None*** (no link), ***LinkDestination.Page*** (a page of a document) or ***LinkDestination.Url*** (the URL in ***documentName***). See the enumeration type ***LinkDestination***.

Methods:

setAction *(LinkAction action, int pageNumber=1)*	*void*

Sets the action of a page link: go to the page ***pageNumber*** (***LinkAction.GotoPageNumber***) or to the first, previous, next or last page. It applies to ***LinkDestination.Page***.

Example:

```
var ft = textObject.content.formattedText;
var link = new Link();
link.documentName = "https://www.viva.de";
link.destination = LinkDestination.Url;
ft.setLink(link, 0, 5);
var pageLink = new Link(true, "", LinkAction.GotoPageNumber, 3);
ft.setLink(pageLink, 6, 11);
```

Class: Note

The class is used in a class `FormattedText`.

Properties:

type	NoteType	*Read/Write*

text	*String*	*Read/Write*
anchorText	*String*	*Read/Write*
position	int *Read*	
mainText	FormattedText	*Read*

Class: OpenType

Constructor:

new **OpenType()**

Properties:

contextualAlternates	*ThreeStates*	*Read/Write*
capitalSpacing	*ThreeStates*	*Read/Write*
fractions	*ThreeStates*	*Read/Write*
ordinals	*ThreeStates*	*Read/Write*
swash	*ThreeStates*	*Read/Write*
slashedZero	*ThreeStates*	*Read/Write*
caseSensitiveForms	*ThreeStates*	*Read/Write*

stylisticAlternates	*int*	*Read/Write*
stylistic1	*int*	*Read/Write*
stylistic2	*int*	*Read/Write*
stylistic3	*int*	*Read/Write*
stylistic4	*int*	*Read/Write*
stylistic5	*int*	*Read/Write*
stylistic6	*int*	*Read/Write*
stylistic7	*int*	*Read/Write*
stylistic8	*int*	*Read/Write*
stylistic9	*int*	*Read/Write*
stylistic10	*int*	*Read/Write*
stylistic11	*int*	*Read/Write*
stylistic12	*int*	*Read/Write*
stylistic13	*int*	*Read/Write*
stylistic14	*int*	*Read/Write*

stylistic15	*int*	*Read/Write*
stylistic16	*int*	*Read/Write*
stylistic17	*int*	*Read/Write*
stylistic18	*int*	*Read/Write*
stylistic19	*int*	*Read/Write*
stylistic20	*int*	*Read/Write*

placement	*OpenTypePlacement*	*Read/Write*
numerals	*OpenTypeNumerals*	*Read/Write*
positionalForms	*OpenTypePositionalForms*	*Read/Write*

smallCapitalsFromCapitals	*ThreeStates*	*Read/Write*
smallCapitals	*ThreeStates*	*Read/Write*
petiteCapitalsFromCapitals	*ThreeStates*	*Read/Write*
petiteCapitals	*ThreeStates*	*Read/Write*
titling	*ThreeStates*	*Read/Write*
unicase	*ThreeStates*	*Read/Write*

discretionaryLigatures	*ThreeStates*	*Read/Write*
historicalLigatures	*ThreeStates*	*Read/Write*
`featuresString`	*String*	*Read/Write*

Get/Set the features as one string: the OpenType feature tags of the properties above with their values, separated by commas, for example *`"smcp=1,onum=1,ss01=1"`*. Reading returns only the features that are set; other tags are ignored.

Class: OptimizePictureSettings

The class holds sittings for the picture optimizing.

Constructor:

```
new OptimizePictureSettings();
```

Properties:

`crop`	*bool*	*Read/Write*
`prefix`	*String*	*Read/Write*
`postfix`	*String*	*Read/Write*

format	*PictureFormat (enum)*	*Read/Write*

Valid only if the method outFilePath is empty.

bleed	*float*	*Read/Write [pt]*
colorImageResolution	*float*	*Read/Write [ppi]*
grayImageResolution	*float*	*Read/Write [ppi]*
bwImageResolution	*float*	*Read/Write [ppi]*
outFilePath	*String*	*Read/Write*

Embed the image if empty.

Class: OuterGlow

The class is used in the class ***GraphicObject*** for all objects and the class ***GraphicStyleSheet***. The class holds the parameters used for the "Outer Glow" effect of the object.

Constructor:

new **OuterGlow()**

Properties:

active	*bool*	*Read/Write*		
color	*ColorBrush*	*Read/Write*		
shade	*float*	*Read/Write*	*[%]*	*<0-100>*
opacity	*float*	*Read/Write*	*[%]*	*<0-100>*
mode	*ColorMode (enum)*	*Read/Write*		
blur	*UnitValueString*	*Read/Write*	*[mm]*	*<0, 20>*

Class: OutlineOptions

The object of the class holds the parameters of the text outline.

The class is used in classes ***FormattedText***, ***CharacterStyleSheet*** and ***ParagraphStyleSheet***.

Constructor:

new **OutlineOptions()**

new **OutlineOptions**(*ColorBrush brush, float shade, float opacity, UnitValueString lineWidth, String lineStyle, CharacterOutlineJoinStyle joinStyle*)

Properties:

active	*bool*	*Read/Write*
brush	*ColorBrush*	*Read/Write*
shade	*float*	*Read/Write*

opacity	*float*	*Read/Write*
lineWidth	*UnitValueString*	*Read/Write*
lineStyle	*LineStyleString*	*Read/Write*
joinStyle	*CharacterOutlineJoinStyle*	*Read/Write*

Examples:

```
var ol = new OutlineOptions(Color.Red, 44, 56, "1mm", "SolidLine",
       CharacterOutlineJoinStyle.RoundJoin);
ft.setOutlineOptions(true, ol, new TextRange(10, 15, ft));
//switch off the feature
ft.setOutlineOptions(false, ol);
```

Class: PackageSettings

The class is used in classes *Document*.

Constructor:

new **PackageSettings()**

Properties:

includeProtocol	*bool*	*Read/Write*
includeFonts	*bool*	*Read/Write*
includeImages	*bool*	*Read/Write*
includeDocument	*bool*	*Read/Write*
cleanupFolder	*bool*	*Read/Write*
overwriteExistingFiles	*bool*	*Read/Write*
useSubfolders	*bool*	*Read/Write*
fontsSubfolder	*String*	*Read/Write*
imagesSubfolder	*String*	*Read/Write*
zip	*bool*	*Read/Write*

Class: ParagraphBackground

The class is used in the classes *ParagraphStyleSheet* and *FormattedText*.

Constructor:

new **ParagraphBackground()**

new **ParagraphBackground**(*bool active,*
ColorBrush brush, float shade, float opacity,
UnitValueString leftIndent, UnitValueString rightIndent,

UnitValueString topIndent, UnitValueString bottomIndent,
ExtendToNextParagraphType extNext, bool extToColumn)

Properties:

active	*bool*	*Read/Write*
brush	*ColorBrush*	*Read/Write*
shade	*float*	*Read/Write*
opacity	*float*	*Read/Write*
leftIndent	*UnitValueString*	*Read/Write*
rightIndent	*UnitValueString*	*Read/Write*
topIndent	*UnitValueString*	*Read/Write*
bottomIndent	*UnitValueString*	*Read/Write*
extendToNextParagraph	*ExtendToNextParagraphType*	*Read/Write*
extendToColumnBorder	*bool*	*Read/Write*

Examples:

```
  var pb = new ParagraphBackground();
  pb.brush = Color.Yellow;
  pb.shade = 82;
  pb.opacity = 73;
  pb.topIndent = "-2mm";
  pb.leftIndent = "-2mm";
  pb.rightIndent = "-1mm";
  pb.bottomIndent = "-1mm";
  pb.extendToNextParagraph = ExtendToNextParagraphType.Auto;
  pb.extendToColumnBorder = true;

  var pb2 = new ParagraphBackground( true, Color.Green, 82, 73,
        "-2mm", "-2mm", "-1mm", "-1mm",
ExtendToNextParagraphType.Auto);
  ft.setParagraphBackground(pb, new TextRange(10, 11));

  //remove the feature
  ft.setParagraphBackground(pb, new TextRange(10, 11));
```

Class: ParagraphFrame

The class is used in the classes *ParagraphStyleSheet* and *FormattedText*.

Constructor:

new **ParagraphFrame()**

new **ParagraphFrame**(*bool active,*
ColorBrush brush, float shade, float opacity,
UnitValueString leftIndent, UnitValueString rightIndent,
UnitValueString topIndent, UnitValueString bottomIndent,

UnitValueString lineWidth, String lineStyle,
bool leftBorder, bool topBorder, bool rightBorder,
bool bottomBorder, ExtendToNextParamType extendToNextType,
bool extendToColumn)

Properties:

active	*bool*	*Read/Write*
brush	*ColorBrush*	*Read/Write*
shade	*float*	*Read/Write*
opacity	*float*	*Read/Write*
leftIndent	*UnitValueString*	*Read/Write*
rightIndent	*UnitValueString*	*Read/Write*
topIndent	*UnitValueString*	*Read/Write*
bottomIndent	*UnitValueString*	*Read/Write*
frameWidth	*UnitValueString*	*Read/Write*
lineStyle	*LineStyleString*	*Read/Write*
leftBorder	*bool*	*Read/Write*
topBorder	*bool*	*Read/Write*
rightBorder	*bool*	*Read/Write*
bottomBorder	*bool*	*Read/Write*
extendToNextParagraph	*ExtendToNextParagraphType*	*Read/Write*
extendToColumnBorder	*bool*	*Read/Write*

Examples:

```
var pf = new ParagraphFrame();
pf.brush = Color.Blue;
pf.shade = 34;
pf.opacity = 56;
pf.frameWidth = "1mm";
pf.lineStyle = "DotLine";
pf.topIndent = "1mm";
pf.leftIndent = "2mm";
pf.rightIndent = "3mm";
pf.bottomIndent = "2mm";
pf.leftBorder = true;
pf.topBorder = true;
pf.rightBorder = true;
pf.bottomBorder = true;
pf.extendToNextParagraph = ExtendToNextParagraphType.Auto;
pf.extendToColumnBorder = true;
ft.setParagraphFrame(pf, new TextRange(10, 20));
```

Class: ParagraphRule

The class is used in the classes ***ParagraphStyleSheet*** and ***FormattedText***.

Constructor:

```
new ParagraphRule()
new ParagraphRule(bool active,
      ColorBrush brush, float shade, float opacity,
      UnitValueString lineWidth, String lineStyle,
      UnitValueString offset,
      UnitValueString leftIndent, UnitValueString rightIndent,
      ParagraphRuleLengthType lengthType)
```

Properties:

active	*bool*	*Read/Write*	
brush	*ColorBrush*	*Read/Write*	
shade	*float*	*Read/Write*	
opacity	*float*	*Read/Write*	
lineWidth	*UnitValueString*	*Read/Write*	
lineStyle	*String*	*Read/Write*	
offset	*UnitValueString*	*Read/Write*	
leftIndent	*UnitValueString*	*Read/Write*	
rightIndent	*UnitValueString*	*Read/Write*	
lengthType	*ParagraphRuleLengthType*		*Read/Write*

Examples:

```
var pre = new ParagraphRule();
pre.active = true;
pre.brush = Color.Green;
pre.shade = 67;
pre.opacity = 78;
pre.lineWidth = "1mm";
pre.lineStyle = "DashLine";
pre.offset = "-12mm";
pre.leftIndent = "1mm";
pre.rightIndent = "7mm";
pre.lengthType = ParagraphRuleLengthType.Space;
ft.setParagraphRuleBelow(pre, new TextRange( 33, 35));
```

Class: PDFExportSettings

The object of the class holds the settings of the PDF export. It is used in the method ***exportToPDF()*** of the classes ***Document*** and ***Collection***.

Constructor:

```
new PDFExportSettings()
```

Properties:

pagesRangeType	*PrintPagesRangeType*	*Read/Write*
fromPage	*int*	*Read/Write*
toPage	*int*	*Read/Write*
pageSelection	*String*	*Read/Write*
printSpreads	*bool*	*Read/Write*
printEmptyPages	*bool*	*Read/Write*
printSpellCorrection	*bool*	*Read/Write*
layersRange	*PrintLayersRange*	*Read/Write*
printSeparately	*bool*	*Read/Write*
pageNumbering	*PrintPageNumbering*	*Read/Write*
numberOfDigits	*int*	*Read/Write*
nameChangePolicy	*PrintNameChangePolicy*	*Read/Write*
pagesPerJob	*int*	*Read/Write*
registrationMarks	*bool*	*Read/Write*
cropmarks	*bool*	*Read/Write*
cropMarkLength	*UnitValueString*	*Read/Write*
cropMarkDistance	*UnitValueString*	*Read/Write*
colorSeparation	*PrintColorSeparation*	*Read/Write*
useBleed	*bool*	*Read/Write*
bleedLeft	*UnitValueString*	*Read/Write*
bleedTop	*UnitValueString*	*Read/Write*
bleedRight	*UnitValueString*	*Read/Write*
bleedBottom	*UnitValueString*	*Read/Write*
horizontalAlignment	*PrintHorizontalAlignment*	*Read/Write*
verticalAlignment	*PrintVerticalAlignment*	*Read/Write*
outputPath Get/Set the path of the PDF file.	*String*	*Read/Write*
fontEmbedding	*PrintFontEmbedding*	*Read/Write*
ignoreTrapping	*bool*	*Read/Write*
imageQuality	*PrintImageQuality*	*Read/Write*
opi	*PrintOPI*	*Read/Write*

Has no effect: VivaDesigner does not create OPI information. The value is kept so that existing scripts still run.

pdfCompatibility	*PDFCompatibility*	*Read/Write*
pdfOptimize	*bool*	*Read/Write*
pdfCreateLayers	*bool*	*Read/Write*
pdfCreateBookmarks	*bool*	*Read/Write*

Get/Set whether the PDF file gets bookmarks.

pdfCompressText	*bool*	*Read/Write*
pdfWarningScale	*bool*	*Read/Write*
pdfWarningScaleBelow	*int*	*Read/Write*
pdfWarningScaleAbove	*int*	*Read/Write*
pdfWarningTextOverflow	*bool*	*Read/Write*
pdfExportNotes	*bool*	*Read/Write*

Get/Set whether the notes of objects and texts are exported as PDF comments. This does not apply to PDF/X and PDF/A files and to tagged PDF files.

pdfCompressionMethod	*PDFCompressionMethod*	*Read/Write*
pdfCompressionQuality	*int*	*Read/Write*
pdfColorConversion	*PDFColorConversion*	*Read/Write*
pdfTargetProfile	*String*	*Read/Write*
pdfxOutputIntent	*String*	*Read/Write*
pdfxGrayProfile	*String*	*Read/Write*
pdfxAllowEmbeddedICCProfiles	*bool*	*Read/Write*

Class: Reflection

Constructor:

new **Reflection()**

Properties:

active	*bool*	*Read/Write*		
size	*float*	*Read/Write*	*[%]*	*<0,100>*
offset	*UnitValueString*	*Read/Write*	*[pt]*	*<-60000,60000>*
opacity	*float*	*Read/Write*	*[%]*	*<0,100>*

Class: Runaround

The class is used in the classes GraphicStyleSheet and Object. It determines the way text

wraps around the object.

Constructor:

new **Runaround()**

new **Runaround**(*RunaroundMode mode, RunaroundShapeType shapeType, UnitValueString distance*)

new **Runaround**(*RunaroundMode mode, RunaroundShapeType shapeType, UnitValueString left, UnitValueString top, UnitValueString right, UnitValueString bottom*)

Properties:

mode	*RunaroundMode*	*Read/Write*	
shapeType	*RunaroundShapeType*	*Read/Write*	
distance	*UnitValueString*	*Read/Write*	*<0, 20000> [mm]*
left	*UnitValueString*	*Read*	*<0, 20000> [mm]*
top	*UnitValueString*	*Read*	*<0, 20000> [mm]*
right	*UnitValueString*	*Read*	*<0, 20000> [mm]*
bottom	*UnitValueString*	*Read*	*<0, 20000> [mm]*

Method:

setBlock *(UnitValueString left, UnitValueString top, UnitValueString right, UnitValueString bottom)* *void*

Sets the four distances of the runaround separately (for ***RunaroundShapeType.Block***). Values without a unit are points. The properties ***left***, ***top***, ***right*** and ***bottom*** return them.

Examples:

```
var ra1 = new Runaround( RunaroundMode.Right,
RunaroundShapeType.ObjectContours, "1mm");

var ra2 = new Runaround(RunaroundMode.Left, RunaroundShapeType.Block,
            "2mm", "12mm", "13mm", "14mm");
textObject1.runaround = ra1;
textObject2.runaround = ra2;
```

Class: Shadow

The class is used in the class ***GraphicObject*** for all objects and the class ***GraphicStyleSheet***. The class holds the parameters used for the "Drop Shadow" effect of the object.

Constructor:

new **Shadow()**

Properties:

active	*bool*	*Read/Write*	
color	*ColorBrush*	*Read/Write*	
shade	*float*	*Read/Write*	*[%] <0-100>*

opacity	*float*	*Read/Write*	*[%] <0-100>*
mode	*ColorMode (enum)*	*Read/Write*	
angle	*float*	*Read/Write*	*[degree] <-360,360>*
offset	*UnitValueString*	*Read/Write*	*[pt] <-60000,60000>*
blur	*UnitValueString*	*Read/Write*	*[mm] <0,20>*
scale	*float*	*Read/Write*	*[%] <0,20000>*
coverShadow	*bool*	*Read/Write*	
applyElementOpacity	*bool*	*Read/Write*	

Examples:

```
var sh = new Shadow();
sh.active = true;
sh.color = Color.Red;
sh.shade = 90;
sh.opacity = 88;
sh.mode = ColorMode.HardLight;
sh.angle = 166;
sh.offset = "55mm";
sh.blur = "1pt";
sh.scale = 100;
sh.coverShadow = false;
sh.applyElementOpacity = false;

if (textObject) {
  textObject.shadow = sh;
}
```

Class: SpellLang

The object of the class stands for a language. It is used in the method ***text()*** of the class ***Variable***.

Constructor:

new **SpellLang**(*Language type*)

Properties:

type	*Language*	*Read*

Returns the language as a value of the enumeration type ***Language***.

name	*String*	*Read*

Returns the key of the language as ***FormattedText.setLanguageByName()*** takes it, for example *"english_US"* or *"german_DE_ref"*.

bcp47	*String*	*Read*

Returns the BCP 47 language tag, for example *"en-US"*, *"de-DE"* or *"sr-Cyrl-RS"*. It is empty for ***Language.NoLang*** and ***Language.Universal***.

Example:

```
var lang = new SpellLang(Language.German_DE_Reformed);
var tag = lang.bcp47;      // "de-DE"
var key = lang.name;       // "german_DE_ref"
```

Class: StrikethroughOptions

The object of this class holds the parameters of a strikethrough line.

The class is used in classes FormattedText, CharacterStyleSheet and ParagraphStyleSheet.

Constructor:
An object can be created using the constructor: new StrikethroughOptions() and the document parameters are set using properties. Alternatively you can use the following constructor with parameters:

```
new StrikethroughOptions(FontStyleLineType type,
        ColorBrush brush, float shade, float opacity,
        UnitValueString lineWidth, String lineStyle,
        UnitValueString offset, FontStyleLineRangeType type)
     or
new StrikethroughOptions(FontStyleLineType type)
```

Properties:

type	*FontStyleLineType*	*Read/Write*
brush	*ColorBrush*	*Read/Write*
Get/Set the text color.		
shade	*float*	*Read/Write*
Get/Set the shade of the text color.		
opacity	*float*	*Read/Write*
Get/Set the opacity of the text color.		
lineWidth	*UnitValueString*	*Read/Write*
Get/Set the line width of the line.		
lineStyle	*String*	*Read/Write*
Get/Set the line style of the line. See the enumeration type *LineStyle*.		
offset	*UnitValueString*	*Read/Write*
Get/Set the distance under the text where the line is applied.		
rangeType	*FontStyleLineRangeType*	*Read/Write*
Get/Set the line range. See the enumeration type *FontStyleLineRangeType.*		

Class: TablesAbbreviation

The class is used in classes *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:
An object can be created using the constructor: *new TablesAbbreviation()* and the document parameters are set using properties. Alternatively you can use the following constructor with parameters:

new **TablesAbbreviation***(bool active, String alternativeText)*

Properties:

active	*bool*	*Read/Write*
alternativeText	*String*	*Read/Write*

Class: TablesBibliography

The class is used in classes *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:

An object can be created using the constructor: *new TablesBibliography()* and the document parameters are set using properties. Alternatively you can use the following constructor with parameters:

new **TablesBibliography***(bool active, String alternativeText)*

Properties:

active	*bool*	*Read/Write*
alternativeText	*String*	*Read/Write*

Class: TablesContents

The class is used in classes *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:
An object can be created using the constructor: *new TablesContents()* and the document parameters are set using properties. Alternatively you can use the following constructor with parameters:

new **TablesContents***(bool active, String alternativeText, int level)*

Properties:

active	*bool*	*Read/Write*
alternativeText	*String*	*Read/Write*
level	*int*	*Read/Write*

Class: TablesIndex

The class is used in classes *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:
An object can be created using the constructor: *new TablesIndex()* and the document parameters are set using properties. Alternatively you can use the following constructor with parameters:

new **TablesIndex***(bool active, String alternativeText, bool appendPageNumber)*

Properties:

active	*bool*	*Read/Write*
alternativeText	*String*	*Read/Write*
appendPageNumber	*bool*	*Read/Write*

Get/Set whether the page number is added to the index entry.

Class: TablesPicture

The class is used in classes *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:
An object can be created using the constructor: *new TablesPicture()* and the document parameters are set using properties. Alternatively you can use the following constructor with parameters:

new **TablesPicture***(bool active, String alternativeText)*

Properties:

active	*bool*	*Read/Write*
alternativeText	*String*	*Read/Write*

Class: TablesRunningTitle

The class is used in classes *FormattedText*, *CharacterStyleSheet* and *ParagraphStyleSheet*.

Constructor:
An object can be created using the constructor: *new TablesRunningTitle()* and the document parameters are set using properties. Alternatively you can use the following constructor with parameters::

new **TablesRunningTitle***(bool active, String alternativeText, int level)*

Properties:

active	*bool*	*Read/Write*

alternativeText	*String*	*Read/Write*
level	*int*	*Read/Write*

Class: Tabulator

The class is used in the class ***Tabulators***.

Constructor:

new **Tabulator** *()*
new **Tabulator (String position, TabulatorAlignment alignment, String fillCharacter)**

Properties:

position	*UnitValueString*	*Read/Write*
alignment	*TabulatorAlignment*	*Read/Write*
alignCharacter	*Char*	*Read/Write*
fillCharacter	*String*	*Read/Write*

Class: Tabulators

The class is used in the classes ParagraphStyleSheet and FormattedText.

Constructor:

new **Tabulators** *(TabsType type)*
new **Tabulators** *(TabsType type, bool standardTabs, bool indent)*

Properties:

type	*TabsType*	*Read/Write*
active	*bool*	*Read*
count	*int*	*Read*
applyStandardTabs	*bool*	*Read/Write*
applyAsIndent	*bool*	*Read/Write*

Methods:

getTabulator *(int index)*	*Tabulator*	
addTabulator *(Tabulator tabulator)*	*void*	
removeTabulator *(int index)*	*void*	
removeAllTabulators *()*	*void*	
setTabulator *(int index, Tabulator tabulator)*		*void*

Replaces the tabulator at the position ***index*** (0 .. ***count*** - 1) with ***tabulator***.

Class: TaggedPdfOptions

The object of the class holds how an object is exported into a tagged (accessible) PDF file. It is used in the property ***taggedPdfOptions*** of the class ***Object***. The property returns a copy: assign the object to the property again after a change.

Constructor:

```
new TaggedPdfOptions()
new TaggedPdfOptions(TaggedPdfType type, String alternativeText)
```

Properties:

tagType	*TaggedPdfType*	*Read/Write*

Get/Set how the object is tagged: ***TaggedPdfType.Automatic*** (default; a picture gets its file name as alternative text), ***TaggedPdfType.Artifact*** (decoration, not read aloud) or ***TaggedPdfType.Figure*** (a figure with the alternative text). See the enumeration type ***TaggedPdfType***.

alternativeText	*String*	*Read/Write*

Get/Set the alternative text of the object. Screen readers read it instead of the object. A figure without alternative text is exported as decoration.

Example:

```
var obj = application.currentDocument.currentPage.objects().at(0);
obj.taggedPdfOptions = new TaggedPdfOptions(TaggedPdfType.Figure,
  "Sales 2026 by region");
```

Class: TextEndnotes

The class is used in class TextPreferences and FormattedText.

Constructor:

An object can be created using a constructor:

```
new TextEndnotes ()
```

and the document parameters are set using the following Properties:

Properties:

displayPosition	*EndnotesBreakType*	*Read/Write*
saveTo	*EndnotesPositionType*	*Read/Write*
textOffset	*UnitValueString*	*Read/Write*
prefix	*String*	*Read/Write*
postfix	*String*	*Read/Write*
numberFormat	*EnumerationsNumberFormat*	*Read/Write*

Get/Set the number format of the endnote numbers. See the enumeration type ***EnumerationsNumberFormat***; ***Format_Asterisks*** (** ** ****) and ***Format_Symbols*** (** …*) are meant for notes.

numberingType	*ReferenceAnchorNumberingType*	*Read/Write*

Get/Set where the numbering starts anew: ***ContinuousNumbering*** (never), ***ChapterWiseNumbering*** (in every chapter), ***PageRestartNumbering*** (on every page), ***SpreadRestartNumbering*** (on every spread) or ***SectionRestartNumbering*** (in every section of the page numbering). See the enumeration type ***ReferenceAnchorNumberingType***.

startNumber	*int*	*Read/Write*
separatorVisible	*bool*	*Read/Write*
separatorVerticalOffset	*UnitValueString*	*Read/Write*
separatorHorizontalOffset	*UnitValueString*	*Read/Write*
separatorLineWidth	*UnitValueString*	*Read/Write*
separatorLineBrush	*ColorBrush*	*Read/Write*
separatorLineStyle	*String*	*Read/Write*
separatorLineLength	*UnitValueString*	*Read/Write*

Class: TextFootnotes

The class is used in the class TextPreferences and FormattedText.

Constructor:
An object can be created using constructor:

```
new TextFootnotes ()
```

and the document parameters are set using following Properties:

Properties:

textOffset	*UnitValueString*	*Read/Write*
prefix	*String*	*Read/Write*
postfix	*String*	*Read/Write*
numberFormat	*EnumerationsNumberFormat*	*Read/Write*

Get/Set the number format of the footnote numbers. See the enumeration type ***EnumerationsNumberFormat***; ***Format_Asterisks*** (`* ** ***`) and ***Format_Symbols*** (`*` ...) are meant for notes.

numberingType	*ReferenceAnchorNumberingType*	*Read/Write*

Get/Set where the numbering starts anew: ***ContinuousNumbering*** (never), ***ChapterWiseNumbering*** (in every chapter), ***PageRestartNumbering*** (on every page), ***SpreadRestartNumbering*** (on every spread) or ***SectionRestartNumbering*** (in every section of the page numbering). See the enumeration type ***ReferenceAnchorNumberingType***.

startNumber	*int*	*Read/Write*
separatorVisible	*bool*	*Read/Write*
separatorVerticalOffset	*UnitValueString*	*Read/Write*
separatorHorizontalOffset	*UnitValueString*	*Read/Write*

separatorLineWidth	*UnitValueString*	*Read/Write*
separatorLineBrush	*ColorBrush*	*Read/Write*
separatorLineStyle	*String*	*Read/Write*
separatorLineLength	*UnitValueString*	*Read/Write*

Class: TransparencyStop

Constructor:

new **Stop** *()*
new **Stop(***float opacity, float position, float focusPosition***)**

Properties:

opacity	*float*	*Read/Write*	*[%]*	*<0,100>*
position	*float*	*Read/Write*	*[%]*	*<0,100>*
focusPosition	*float*	*Read/Write*	*[%]*	*<0,100>*

Class: Transparency

Constructor:

new **Transparency** *(ColorBrush brush)*

Properties:

active	*bool*	*Read/Write*		
type	*TransparencyType (enum)*	*Read/Write*		
angle	*float*	*Read/Write*	*[degree]*	*<-360,360>*
stopCount	*int*	*Read*		

Methods:

stopAt *(int index)* *TransparencyStop*

Returns the stop at the position ***index*** (0 .. ***stopCount*** - 1).

setStopAt *(int index, TransparencyStop stop)* *void*

Replaces the stop at the position ***index***.

removeStopAt *(int index)* *void*

Removes the stop at the position ***index***.

appendStop *(TransparencyStop stop)* *void*

Appends a stop.

Class: UnderlineOptions

The object of this class holds the parameters of the text underline.

The class is used in the classes FormattedText, CharacterStyleSheet and ParagraphStyleSheet.

Constructor:
An object can be created using the constructor:

```
new UnderlineOptions ()
```

or

```
new UnderlineOptions (FontStyleLineType type)
```

and the parameters are set using properties. The following constructor can also be used with parameters:

```
new UnderlineOptions (FontStyleLineType type,
ColorBrush brush, float shade, float opacity,
UnitValueString lineWidth, String lineStyle,
UnitValueString offset, FontStyleLineRangeType type)
```

Properties:

type	*FontStyleLineType*	*Read/Write*
brush	*ColorBrush*	*Read/Write*
Get/Set the text color.		
shade	*float*	*Read/Write*
Get/Set the shade of the text color.		
opacity	*float*	*Read/Write*
Get/Set the opacity of the text color.		
lineWidth	*UnitValueString*	*Read/Write*
Get/Set the line width of the line.		
lineStyle	*String*	*Read/Write*
Get/Set the line style (line type). See the enumeration type LineStyle.		
offset	*UnitValueString*	*Read/Write*
Get/Set the distance under the text where the line is applied.		
rangeType	*FontStyleLineRangeType*	*Read/Write*
Get/Set the line range. See the enumeration type FontStyleLineRangeType.		

Examples:

```
var opt = new UnderlineOptions();
opt.brush = Color.Red;
opt.shade = 92;
opt.opacity = 99;
opt.lineWidth = "2pt";
opt.lineStyle = "DashLine";
opt.offset = "0mm";
opt.rangeType = FontStyleLineRangeType.Words;
ft.setManualUnderline(opt);
ft.setAutomaticUnderline(FontStyleLineRangeType.Words);
```

Class: Variable

The object of the class stands for a variable in a text. The class is used in the class FormattedText (*variable()*, *insertVariable()*).

Constructor:

new **Variable***(VariableType type)*
new **Variable***(String typeName)*

type: a value of the enumeration type *VariableType*. Each value creates the variable of its name: *Date*, *Note*, *NoteNumber*, *DocumentCreationDate*, *DocumentModificationDate* or *RunningTitle*. If the installation does not offer the variable, the constructor throws an error.

typeName: the type name of the variable, for example *"Viva Technology/Page Number"*. This is the way to every type without a value of its own in *VariableType* (for example the variables of add-ins).

Properties:

type	*String*	*Read*

Returns the type name of the variable, for example *"Viva Technology/Document Creation Date"*.

Methods:

text *(SpellLang language, TextObject textObject)*	*String*

Returns the text the variable shows in the text object *textObject* for the language *language*, for example a date in the format of the language.

Example:

```
var obj = application.currentDocument.currentPage.objects().at(0);
var ft = obj.content.formattedText;
ft.insertVariable(new Variable(VariableType.DocumentCreationDate), 0);
var v = ft.variable(0);
var s = v.text(new SpellLang(Language.English_US), obj);
```

Class: WidowsOrphans

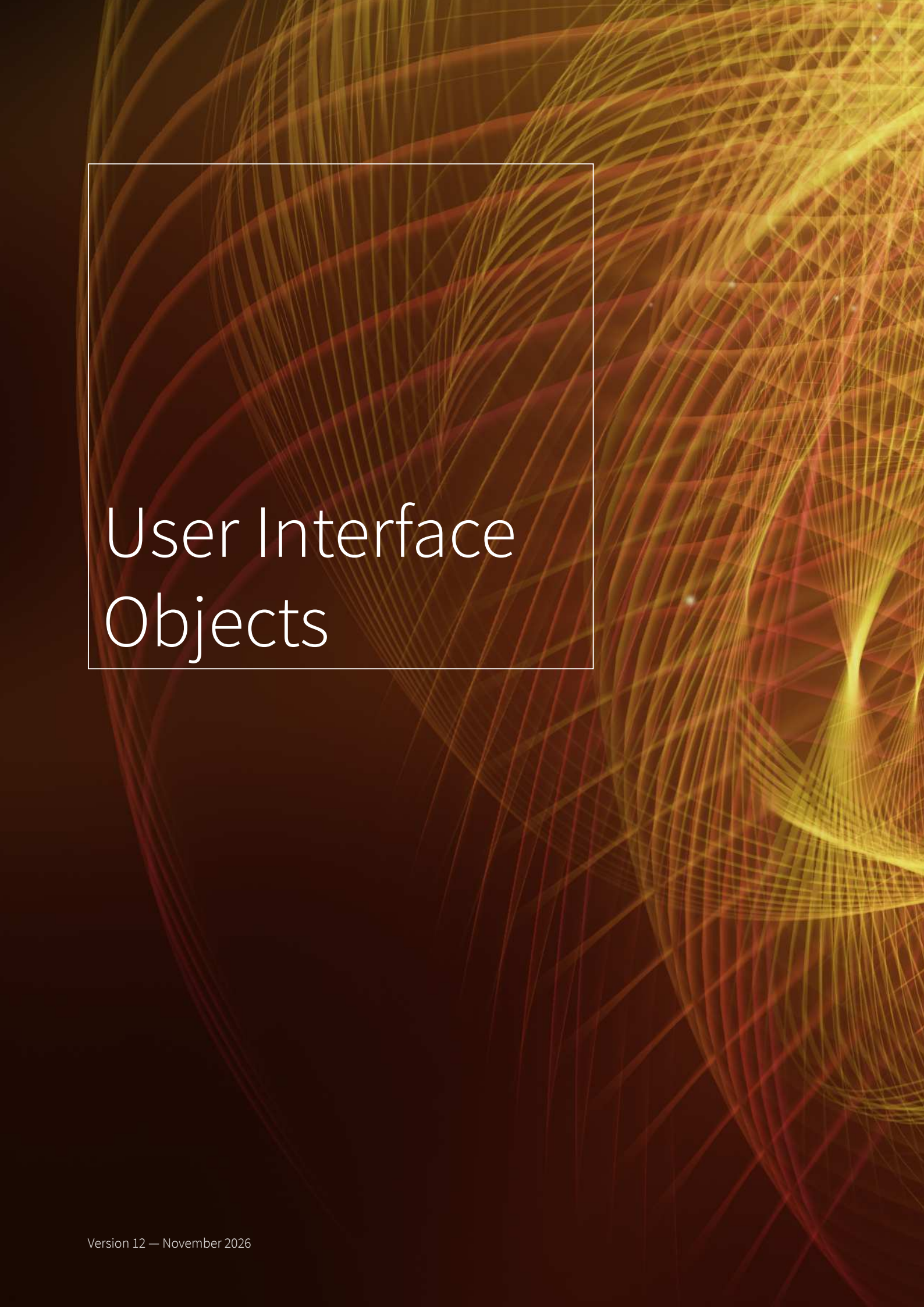
The class is used in the class *FormattedText* and *ParagraphStyleSheet*.

Constructor:

new **WidowsOrphans***(WidowsOrphansMode mode)*
new **WidowsOrphans***()*

Properties:

mode	*WidowsOrphansMode*	*Read/Write*
firstLines	*int*	*Read/Write*
lastLines	*int*	*Read/Write*



Version 12 — November 2026

User Interface Objects

Class: MessageBox

Methods:

`information (String title, String message)`	*void*
`warning (String title, String message)`	*void*

Class: Directory

The class represents a directory or a folder.

Constructor:
An object can be created using a constructor:

```
new Directory(String path)
```

Properties:

`fullPath`	*String*	*Read*
Get full path string.		
`name`	*String*	*Read*
Get relative path name for the directory.		
`path`	*String*	*Read/Write*
`count`	*int*	*Read*
`isReadable`	*bool*	*Read*
`exists`	*bool*	*Read*
`isRoot`	*bool*	*Read*
`isRelative`	*bool*	*Read*
`isAbsolute`	*bool*	*Read*

Methods:

`makeDir(String dir)`	*bool*
Creates the subdirectory `dir` in the current directory. Returns TRUE if the directory is created successfully.	
`removeDir(String dir)`	*bool*
Removes the subdirectory `dir` in the current directory. Returns TRUE if the directory is removed successfully.	
`changeDir(String dir)`	*bool*
Changes the current directory to the directory dir. Returns TRUE if the directory is changed successfully.	

renameDir(*String* **oldName**, *String* **newName**) *bool*

Renames the subdirectory ***oldName*** in the current directory to the new name ***newName***. Returns TRUE if the directory is renamed successfully.

getFileList(*String nameFilters, DirectoryFilter filters=DirectoryFilter.NoFilter, DirectorySortFlag sort = DirectorySortFlag.NoSort*) *String[]*

Returns list of file names from the directory.

nameFilters: wildcard patterns separated by semicolons, for example *"*.jpg;*.png"*.

getFileList(*DirectoryFilter filters = DirectoryFilter.NoFilter, DirectorySortFlag sort = DirectorySortFlag.NoSort*) *String[]*

Returns list of file names from the directory.

zip (*String filePath = null*) *bool*

Packs the files of the directory into a ZIP file (ZIP64, so files of 4 GB and more are possible). Subfolders are not packed. Returns TRUE if the ZIP file was created successfully, otherwise returns FALSE.

filePath - a full file name of a result zip file. If it is empty, the name of a source directory is used with 'zip' extension.

Class: File

The class represents a general file.

Constructor:

An object can be created using a constructor:

new **File**(*String fullFilePath*)

Properties:

fileName *String* *Read*

Returns the full file path.

exists *bool* *Read*

Returns the information if the file exists or not.

size *int* *Read*

Returns the size of file in bytes.

Methods:

copy(*String newName*) *bool*

Copies the current file to the new position `newName`.

newName: full path file name

rename(*String newName*) *bool*

Renames the current file to the new name.

newName: full path file name

remove() *bool*

Removes the file.

---

zip(*String filePath = null*) *bool*

Packs the file into a ZIP file (ZIP64, so files of 4 GB and more are possible). Returns ***TRUE*** if the ZIP file was created successfully, otherwise returns ***FALSE***.

filePath - a full file name of a result zip file. If it is empty, the name of a source file is used with 'zip' extension.

---

unzip(*String directoryPath = null*) *bool*

Unpacks the ZIP file. The folders of the archive are created; entries that would lie outside the target folder (absolute paths or "../") are skipped. Returns ***TRUE*** if the file was unpacked successfully, otherwise returns ***FALSE***.

directoryPath - a full result directory name. If it is empty, the same directory as a source file has is used.

Class: TextFile

The class represents a file as a text file. The class ***TextFile*** extends the ***File*** class. The data is read line by line.

Constructor:
An object can be created using a constructor:

new **TextFile**(*String fullFilePath*)

Properties:

---

eof *bool* *Read*

Checks the reading file is on the end of the file.

Methods:

---

open(*OpenFileMode fileMode*) *bool*

Opens the file in a defined mode. If the file doesn't exist, the new file is created. Returns true if the file is opened successfully.

---

close() *void*

Close the open file.

---

readLine() *String*

Read one line of the text from the open text file at a current position. Returns ***null*** if no following line exists in the file.

---

writeLine(*String str*) *void*

Write one line of the text to the open text file at a current position. The new line character is added at the end of the line.

---

flush() *bool*

Writes the buffered data of the open file to the disk. Returns TRUE if this succeeded.

Examples:

```
// correct the directory
var d = new Directory(appScriptsPath);
d.makeDir("mydir");
d.changeDir("mydir");

// open or create text file
var file = new TextFile(d.fullPath + "/TestFileName.txt");
messageBox.information("MSG", "The file name is " + file.fileName + ".
\nDoes the file exist? " + file.exists);
if (file.open(OpenFileMode.ReadWrite))
{
  // write new texts to the file
  file.writeLine("Test row in the text file 1!");
  file.writeLine("Test row in the text file 2!");
  file.writeLine("Test row in the text file 3!");
  file.writeLine("Test row in the text file 4!");
  // close file
  file.close();
}
// open or create the file
var rText = "";
if (file.open(OpenFileMode.ReadWrite))
{
  // read all text from the file to one variable
  while (!file.eof)
  {
    rText = rText + file.readLine();
  }
}
file.close();

// create new file and write the one row text in this file
var nfile = new TextFile(d.fullPath + "/TestNewFileName.txt");
if (nfile.open(OpenFileMode.ReadWrite))
{
  nfile.writeLine(rText);
}
nfile.close();

// create new directory
var oldFullPath = d.fullPath;
d.changeDir("..");
d.makeDir("yourdir");
d.changeDir("yourdir");

// copy new file name and remove source file
nfile.copy(d.fullPath + "/" + nfile.fileName.split('/').pop());
nfile.remove();

// rename old file
file.rename(file.fileName + ".old");
```

Class: DataTable

The object of the class holds the cells of an Excel worksheet (XLSX file) as a table of strings. The first row holds the column names. Only the visible rows and columns are read; formatted text comes as HTML.

Constructor:

```
new DataTable()
```

Properties:

rowCount *int* *Read*

Returns the number of rows, the row with the column names included.

columnCount *int* *Read*

Returns the number of columns.

Methods:

loadExcelSheet*(String filePath, String sheetName, String colorColumnName)* *bool*

loadExcelSheet*(String filePath, String sheetName, Array colorColumnNames)* *bool*

Loads a worksheet of an Excel file. Returns TRUE if rows were loaded. If the file does not exist or cannot be read (for example a workbook of more than 100 MB or one million cells), the method returns FALSE, *errorStatus* is *IsNotValid* and the table is empty.

sheetName: the name of the worksheet; empty for the first worksheet.

colorColumnName, *colorColumnNames*: the names of columns (as in the first row, in any case) whose cells get the background color of the cell appended, for example *"red|RGB 255/0/0"*. Empty for none.

columnIndex*(String columnName)* *int*

Returns the index of the column with the name *columnName* (in any case), or -1.

rowAt*(int index)* *String[]*

Returns the cells of the row *index* (0 .. *rowCount* - 1), or an empty array.

rowByFieldContent*(String fieldContent, int columnIndex)* *String[]*

Returns the first row whose cell in the column *columnIndex* is *fieldContent*, or an empty array.

Example:

```
var table = new DataTable();
if (table.loadExcelSheet("C:/Data/prices.xlsx", "", "")) {
  var col = table.columnIndex("Price");
  var row = table.rowByFieldContent("A100", 0);
  if (row.length > 0)
    messageBox.information("Price", row[col]);
}
```

Class: FileDialog

The class works with ***FileDialog*** for selecting the directory or the files.

Constructor:

```
new FileDialog()
```

Properties:

acceptMode	*FileDialogAcceptMode*	*Read/Write*
fileMode	*FileDialogFileMode*	*Read/Write*
viewMode	*FileDialogViewMode*	*Read/Write*
caption	*String*	*Read/Write*

Get/Set the window title of the dialog.

lookInLabel	*String*	*Read/Write*
fileNameLabel	*String*	*Read/Write*
fileTypeLabel	*String*	*Read/Write*
acceptLabel	*String*	*Read/Write*
rejectLabel	*String*	*Read/Write*
directory	*String*	*Read/Write*
directoryFilter	*int*	*Read/Write*
nameFilter	*String*	*Read/Write*
mimeTypeFilter	*String*	*Read/Write*
selectedFiles	*StringList*	*Read*

Methods:

open*()*	*bool*

Opens the file dialog depending on the parameters that are set to the objects before opening. Returns the value true if the selection is accepted. Or it returns the value false if the dialog is canceled.

addOption*(FileDialogOption option)*	*void*
removeOption*(FileDialogOption option)*	*void*
testOption*(FileDialogOption option)*	*bool*
testDirectoryFilter *(DirectoryFilter filter)*	*bool*
getExistingDirectory*(String caption, String dir, FileDialogOption options)*	*String*
getExistingDirectory*()*	*String*
getOpenFileName*(String caption, String dir, String filter, FileDialogOption opts)*	*String*
getOpenFileName*()*	*String*

filter: The filters must be divided by double semicolon char (*.jpg;;*.png)

getSaveFileName(*String caption, String dir, String filter, FileDialogOption opts*)	*String*
getSaveFileName()	*String*

filter: The filters must be divided by double semicolon char (*.jpg;;*.png)

Class: ProgressReporter

The class represents the window with the progress bar and with the information about the progress some operation.

Constructor:

```
new ProgressReporter(String windowTitle, ProgressReporterType type,
  int startPosition, int endPosition)
```

Properties:

progressBarText	*String*	*Write*
Sets the text of the progress window.		
progressBarSubText	*String*	*Write*
Sets the second text line (*ProgressReporterType.TwoTextItems*).		
progressBarPosition	*int*	*Read/Write*
progressBarLength	*int*	*Read*

Methods:

close()	*void*
forceRedraw()	*void*

Redraws the progress window at once: the pending events are processed, user input is ignored. Call it in long loops in which the window is not updated.

progressBarReset(*int begin, int end*)	*void*
progressBarFinish()	*void*

The method finishes the progress bar. It set the property progressBarPosition to value less then the begin value of the progress bar.

Class: Dialog

The class represents a user dialog.

Constructor:

new **Dialog**()

Properties:

caption	*String*	*Read/Write*
minimumWidth	*int*	*Read/Write*

maximumWidth	*int*	*Read/Write*
resizable	*bool*	*Read/Write*

Get/Set whether the user can change the size of the dialog.

Methods:

addLabel*(String label)*	*Label*
addLabel*(int id, String label)*	*Label*
addComboBox*(String label)*	*ComboBox*
addComboBox*(int id, String label)*	*ComboBox*
addCheckBox*(String label, bool defaultValue)*	*CheckBox*
addCheckBox*(int id, String label, bool defaultValue)*	*CheckBox*
addLineEdit*(String label, String defaultValue, LineEditType type)*	*LineEdit*
addLineEdit*(String label, String defaultValue, LineEditType type, int width)*	*LineEdit*
addLineEdit*(int id, String label, String defValue, LineEditType type, int width)*	*LineEdit*
addTextEdit*(String label, String defaultValue)*	*TextEdit*
addTextEdit*(int id, String label, String defaultValue)*	*TextEdit*
addTextEdit*(String label, String defaultValue, int width, int height)*	*TextEdit*
addTextEdit*(int id, String label, String defaultValue, int width, int height)*	*TextEdit*

The text box gets the fixed size ***width*** x ***height*** in pixels; -1 keeps the automatic width or height.

addGroupBox*(String label)*	*GroupBox*
addGroupBox*(int id, String label)*	*GroupBox*

The method adds a box for radio buttons.

addButton*(int id, String label, String clickedMethodName)*	*Button*
addAcceptButton*(int id, String label)*	*Button*
addRejectButton*(int id, String label)*	*Button*
addButtons*()*	*void*
addButtons*(String acceptLabel, String rejectLabel)*	*void*
addButtons*(String acceptLabel, String rejectLabel, bool center)*	*void*

The methods add an accept button and a cancel button in the dialog. With center = true the buttons are centered.

exec*()*	*int*

The method shows a prepared dialog. It returns the code 1=OK button, 0=CANCEL button or the changeEventCode from the combo box if it is set.

accept*()*	*void*

Closes the dialog; ***exec()*** returns 1.

---

done*(int resultCode)* *void*

Closes the dialog; ***exec()*** returns ***resultCode***.

---

reject*()* *void*

Closes the dialog; ***exec()*** returns 0. The three methods can be called from the callback method of a button.

Example:

```
var d = new Dialog();
d.caption = "New Dialog";

var line1 = d.addLineEdit("MyDouble: ", 10.3, LineEditType.Double);
var line2 = d.addLineEdit("MyString: ", "default line text",
LineEditType.String);
var line3 = d.addLineEdit("MyInteger: ", 11, LineEditType.Integer);

var chb = d.addCheckBox("MyCheckBox:", true);
var text = d.addTextEdit("MyTextBoxLabel:", "abcd efgh");

var label = d.addLabel("Some label text");

var groupbox = d.addGroupBox("MyRadioBox");
var rb1 = groupbox.addRadioButton("RB1", false);
var rb2 = groupbox.addRadioButton("RB2", true);

var combo = d.addComboBox("MyCombo");
combo.addItem("ITM1", 1);
combo.addItem("ITM2", 2);
combo.addItem("ITM3", 3);
combo.addItem("ITM4", 4);
combo.addItem("ITM5", 5);

d.addButtons("MyOK", "MyCancel");

var res = d.exec();

messageBox.information("MSG", d.caption + ": Result = " + res + "\n" +
    "LABEL: " + label.text + "\n" +
    "TEXT EDIT: " + text.value + "\n" +
    "LINES EDIT: " + line1.value + ", " + line2.value + ", " + line3.value
+ "\n" +
    "CHECK BOX: " + chb.value + "\n" +
    "RADIO BUTTONS: " + rb1.value + "," + rb2.value + "\n" +
    "COMBO: " + combo.label + "," + combo.value);
```

Class: Label

The class represents a item of a user dialog.

Properties:

`id`	*int*	*Read/Write*
`text`	*String*	*Read/Write*
`isEnabled`	*boolean*	*Read*
`isHidden`	*boolean*	*Read*

Methods:

`hide()`	*void*
`show()`	*void*
`enable()`	*void*
`disable()`	*void*

Class: CheckBox

The class represents a item of a user dialog.

Properties:

`id`	*int*	*Read/Write*
`text`	*String*	*Read/Write*
`isEnabled`	*boolean*	*Read*
`isHidden`	*boolean*	*Read*
`value`	*boolean*	*Read/Write*

It returns ***TRUE*** if the checkbox is checked or ***FALSE*** if it is not checked.

`stateChangedMethod`	*String*	*Read/Write*

A callback method has two parameters: **`id`**, **`value`**.

Methods:

`hide()`	*void*
`show()`	*void*
`enable()`	*void*
`disable()`	*void*

Class: ComboBox

The class represents an item of a user dialog.

Properties:

`id`	*int*	*Read/Write*

Returns a unique id of the combo object.

parentId	*int*	*Read*

Returns a unique id of the parent of the combo object.

text	*String*	*Read/Write*

Returns a label of a selected item.

value	*int*	*Read/Write*

Returns a value of a selected item.

isEnabled	*boolean*	*Read*
isHidden	*boolean*	*Read*
currentIndexChangedMethod	*String*	*Read/Write*

Set// returns a name of the script callback method. The method process is a 'change index' event. The method has three parameters that replace '?' in the method name: combobox id, value, item position.

Sample setting: combo.currentIndexChangedMethod = "processComboBoxEvent(?)";

Sample method definition: processComboBoxEvent(id, value, index) {...}

activatedMethod	*String*	*Read/Write*

A callback method has three parameters: id, value, item position

highlightedMethod	*String*	*Read/Write*

A callback method has three parameters: id, value, item position.

currentIndex	*int*	*Read/Write*

Get/Set the position of the selected item (0 .. *count* - 1).

count	*int*	*Read*

Returns the number of items, separators included.

iconWidth	*int*	*Read/Write*
iconHeight	*int*	*Read/Write*

Get/Set the size of the item icons in pixels (see *addItem()* with an object).

iconFrame	*bool*	*Read/Write*

Get/Set whether the icons of objects get a frame. Set it before the items are added.

width	*int*	*Read*
fixedWidth	*int*	*Write*

Returns the width of the combo box in pixels / sets a fixed width.

Methods:

addItem*(String label, int value)*	*void*
addItem*(String label, int value, boolean selected)*	*void*

The method adds an item into an end of a combo box list.

addItem*(Object obj, String label, int value, bool selected=false)*	*void*

Adds an item with the preview of the object ***obj*** as icon (see ***iconWidth***, ***iconHeight*** and ***iconFrame***).

addSeparator*()*	*void*
insertSeparator*(int index)*	*void*

Adds a separator line at the end of the list or inserts it at the position ***index***.

setItemKey*(int itemIndex, String key)*	*void*
itemKey*(int itemIndex)*	*String*

Set/Get a key of the item at the position ***itemIndex***, for example an identifier that does not depend on the label.

hide*()*	*void*

The method hides a combo box object.

show*()*	*void*

The method shows (makes visible) a combo box object and sets the item to the current position.

enable*()*	*void*
disable*()*	*void*

Class: GroupBox

The class represents a item of a user dialog. It is a box for a radiobutton group.

Properties:

id	*int*	*Read/Write*
clickedMethod	*String*	*Read/Write*

A callback method has one parameter: ID of a radio button.

isEnabled	*boolean*	*Read*
isHidden	*boolean*	*Read*

Methods:

addRadioButton*(int id, String label, bool defaultValue)*	*RadioButton*
addRadioButton*(String label, bool defaultValue)*	*RadioButton*

The methods add a radiobutton item into a groupbox.

hide*()*	*void*
show*()*	*void*
enable*()*	*void*
disable*()*	*void*

Class: RadioButton

The class represents a item of a user dialog.

Properties:

id	*int*	*Read/Write*
text	*String*	*Read/Write*
isEnabled	*boolean*	*Read*
isHidden	*boolean*	*Read*
value	*boolean*	*Read/Write*
It returns TRUE if the radiobutton is selected or FALSE if it is not selected.		
clickedMethod	*String*	*Read/Write*
A callback method has one parameter: ID of the radio button. The 'clickedMethod' takes precedence over the ''clickedMethod' defined by a GroupBox object.		

Methods:

hide*()*	*void*
show*()*	*void*
enable*()*	*void*
disable()	*void*

Class: LineEdit

The class represents a item of a user dialog.

Properties:

id	*int*	*Read/Write*
value	*String*	*Read/Write*
It sets and returns a text from a line.		
isEnabled	*boolean*	*Read*
isHidden	*boolean*	*Read*
cursorPositionChangedMethod	*String*	*Read/Write*
A previous callback method has three parameter: ID, old position, new position.		
editingFinishedMethod	*String*	*Read/Write*
inputRejectedMethod	*String*	*Read/Write*
returnPressedMethod	*String*	*Read/Write*
selectionChangedMethod	*String*	*Read/Write*

The previous four callback methods have one parameter: *ID*. *inputRejectedMethod* is called when the typed text does not fit the *LineEditType*, for example a letter in a line of the type *LineEditType.Integer*.

textChangedMethod	*String*	*Read/Write*
textEditedMethod	*String*	*Read/Write*

The previous two callback methods have two parameters: *ID*,*TEXT*.

width	*int*	*Read*
fixedWidth	*int*	*Write*

Returns the width of the line in pixels / sets a fixed width.

Methods:

hide()	*void*
show()	*void*
enable()	*void*
disable()	*void*

Class: TextEdit

The class represents a item of a user dialog.

Properties:

id	*int*	*Read/Write*
value	*String*	*Read/Write*

It sets and returns a text from a textbox.

cursorPositionChangedMethod	*String*	*Read/Write*

A previous callback method has three parameter: ID, old position, new position

selectionChangedMethod	*String*	*Read/Write*

A previous callback method has one parameter: *ID*.

textChangedMethod	*String*	*Read/Write*

A previous two callback methods has two parameters: *ID*,*TEXT*.

width	*int*	*Read*
fixedWidth	*int*	*Write*
height	*int*	*Read*
fixedHeight	*int*	*Write*

Return the size of the text box in pixels / set a fixed width or height.

isEnabled	*boolean*	*Read*
isHidden	*boolean*	*Read*

Methods:

`hide()`	*void*
`show()`	*void*
`enable()`	*void*
`disable()`	*void*

Class: Button

The class represents a item of a user dialog.

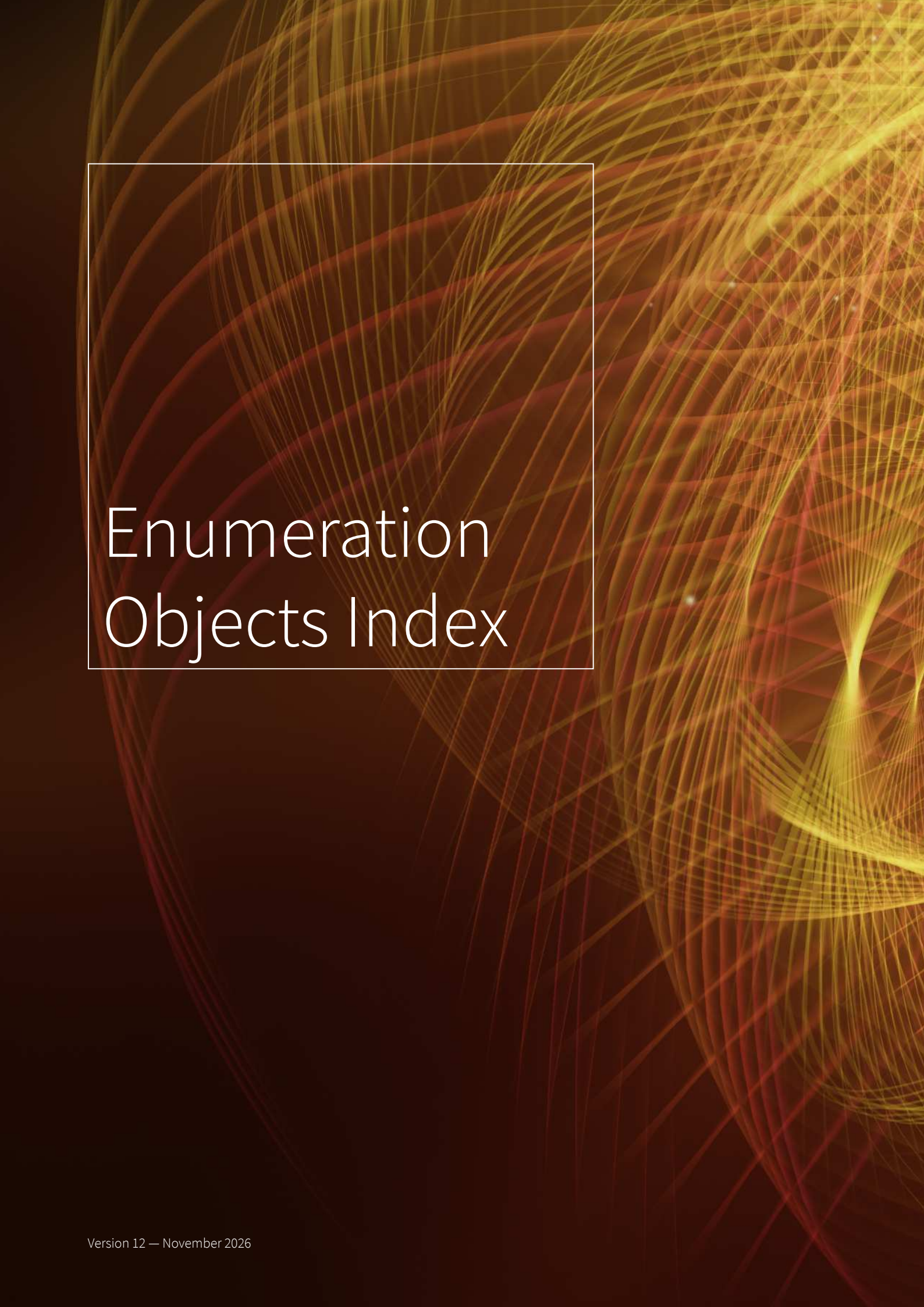
Properties:

`id`	*int*	*Read/Write*
`isEnabled`	*boolean*	*Read*
`isHidden`	*boolean*	*Read*
`text`	*String*	*Read/Write*
`clickedMethod`	*String*	*Read/Write*

A callback method has one parameter: *ID*.

Methods:

`hide()`	*void*
`show()`	*void*
`enable()`	*void*
`disable()`	*void*



Enumeration Objects Index

Version 12 — November 2026

Enumeration Objects Index

AutoLineSpacingMethod:	*AscentDescent*, *FontSize*
AttributeGroup:	*Character, Paragraph, Layout, Story*
BaselineGridNumbering:	*NoneNumbering*, *ShowAllNumbers*, *ShowAllEvenNumbers*, *ShowFirstAndLastNumbers*
BaselineGridDirection:	*LeftToRight*, *RightToLeft*, *TopToBottom*
BaselineGridType:	*None*, *FirstLine*, *AllLines*
BaselineType:	*LargestCharacter*, *AscentHeight*, *CapHeight*, *XHeight*, *Baseline*, *LineSpacing*
BaselineOffset:	*Off*, *Fixed*, *Minimum*
BlendType:	*Linear*, *Rectangle*, *Focus*, *Radial*
BulletCharacters:	*Custom*, *Bullet*, *MiddleDot*, *Dash*, *Plus*, *Triangle*, *Circle*
Color:	*AlwaysPrintingColor*, *Black*, *Blue*, *Bronze*, *Brown*, *Cyan*, *Gold*, *Green*, *Magenta*, *NeverPrintingColor*, *NoColor*, *Orange*, *Pink*, *Red*, *Salmon*, *Silver*, *Turquoise*, *Violet*, *White*, *Yellow*

NoColor, *AlwaysPrintingColor* and *NeverPrintingColor* do not stand for a brush: a *ColorBrush* made from them is not valid.

ColorDepth:	*Depth8bit*, *Depth24bit*
ColorMode:	*Normal*, *Multiply*, *Screen*, *Darken*, *Lighten*, *HardLight*
ColorModel:	*Rgb*, *Cmyk*, *Hsv*, *Lab*
ColorSeparation:	*CMYK*, *SpotColor*
ColorSeparationState:	*Yes*, *No*, *Prepare*
ColorSpace:	*Unknown*, *Mono*, *Gray*, *RGB*, *CMYK*, *LAB*
ContentRotation:	*Angle0*, *Angle90*, *Angle180*, *Angle270*
ContentType:	*Custom*, *Text*, *Picture*, *Table*

Custom: a content that is no text, picture or table.

DialogItemType:	*Unknown, LineEdit, TextEdit, CheckBox, Label, GroupBox, ComboBox, Button*
Digits:	*Default*, *Western*, *ArabicIndic*, *Persian*, *PersianFull*, *Native*, *Devanagari*, *Bengali*, *Gurmukhi*, *Gujarati*, *Oriya*, *Tamil*, *Telugu*, *Kannada*, *Malayalam*, *Thai*, *Lao*, *Tibetan*, *Burmese*, *Khmer*

The digits of numbers (character attribute *characterDigits*). *Default*: the digits of the language of the text; *Western*: 0 1 2 ...; *ArabicIndic*, *Persian*, *PersianFull*: Arabic-Indic and Persian digits; *Native*: the native digits of the language; the other values: the digits of the script of the same name.

DirectoryFilter:	*NoFilter*

Dirs: List directories that match the filters.

AllDirs: List all directories. Don't apply the filters to directory names.

Files: List files.

Drives: List disk drives(ignored under Unix).

NoSymLinks: Do not list symbolic links. (Ignored by operating systems that don't support symbolic links).

NoDot: Do not list the special entry ".".

NoDotDot: Do not list the special entry "..".

NoDotAndDotDot: Do not list the special entries "." and "..".

AllEntries: List directories, files, drives and symlinks. (This does not list broken symlnks unless you specify System).

Readable: List files for which the application has read access.The Readable value needs to be combined with Dirs or Files.

Writable: List files for which the application has write access. The Writable value needs to be combined with Dirs or Files.

Executable: List files for which the application has execute access.The Executable value needs to be combined with Dirs or Files.

Modified: Only list files that have been modified. (ignored on Unix).

Hidden: List hidden files(on Unix, files starting with a ".").

System: List system files(on Unix, FIFOs, sockets and device files are included; on Windows, .lnk files are included)

CaseSensitive: The filter should be case sensitive.

DirectorySortFlag:	*NoSort, Name, Time, Size, Unsorted, DirsFirst, Reversed, IgnoreCase, DirsLast, LocaleAware, Type*
DoubleQuotesType:	*InvalidDoubleQuotes*, *DoubleQuotes1*, *DoubleQuotes2*, *DoubleQuotes3*, *DoubleQuotes4*, *DoubleQuotes5*, *DoubleQuotes6*, *DoubleQuotes7*, *DoubleQuotes8*, *DoubleQuotes9*, *DoubleQuotes10*, *DoubleQuotes11*

The pairs: *DoubleQuotes1* " ", *DoubleQuotes2* „ ", *DoubleQuotes3* " ", *DoubleQuotes4* „ ", *DoubleQuotes5* « », *DoubleQuotes6* » «, *DoubleQuotes7* " " (default), *DoubleQuotes8* corner brackets, *DoubleQuotes9* " ", *DoubleQuotes10* ' ', *DoubleQuotes11* white corner brackets. The values have the numbers 200 to 210.

DropCapsType:	*None*, *WholeWord*, *Sequence*
DropCapsLinesType:	*FixedLineHeight*, *DynamicLineHeight*
DropCapsRunaroundType:	*RunaroundBox*, *RunaroundShape*
EndnotesBreakType:	*PlaceInText*, *PlaceInNewTextColumn*, *PlaceInNewTextObject*
EndnotesPositionType:	*ChapterEnd*, *TextEnd*
EnumerationsType:	*Off*, *Numbering*, *Bullets*
EnumerationsAlignment:	*Left*, *Center*, *Right*
EnumerationsNumberingMode:	*Continue*, *BeginNew*

EnumerationsNumberFormat:	*Format_1_2_3*, *Format_a_b_c*, *Format_A_B_C*, *Format_i_ii_iii*, *Format_I_II_III*, *Format_01_02_03*, *Format_001_002_003*, *Format_0001_0002_0003*, *Format_None*, *Format_Abjad*, *Format_AlifBaTa*, *Format_HebrewBiblical*, *Format_HebrewNonStandard*, *Format_Asterisks*, *Format_Symbols*

Format_01_02_03, *Format_001_002_003*, *Format_0001_0002_0003*: numbers with at least two, three or four digits; *Format_None*: no number (lists); *Format_Abjad*, *Format_AlifBaTa*: Arabic letters in abjad or alphabetical order; *Format_HebrewBiblical*, *Format_HebrewNonStandard*: Hebrew numerals (15 and 16 as tet-vav and tet-zayin, or as yod-he and yod-vav); *Format_Asterisks* (** ** ****), *Format_Symbols* (** …*): for footnotes and endnotes.

ErrorStatus:	*Ok*, *IsNotProcessed*, *IsNotValid*, *IsNotSet*, *IsNotUsable*, *IsNotAccessToFileSystem*

The values of the property *errorStatus* of every object. See the section "Error Status" in the chapter "Scripting Features".

ExposureType:	*Positive*, *Negative*
ExtendToNextParagraphType:	*No*, *Auto*
FileDialogAcceptMode:	*Open*, *Save*
FileDialogFileMode:	*AnyFile*, *ExistingFile*, *Directory*, *ExistingFiles*
FileDialogOption:	*ShowDirsOnly*, *DontResolveSymlinks*, *DontConfirmOverwrite*, *DontUseNativeDialog*, *ReadOnly*, *HideNameFilterDetails*, *DontUseSheet*, *DontUseCustomDirectoryIcons*
FileDialogViewMode:	*Detail*, *List*
FontReferenceSizeType:	*AscentHeight*, *CapHeight*
FontStyleLineRangeType:	*All*, *Words*
FontStyleLineType:	*None*, *Automatic*, *Manual*
FootnotesPosition:	*Layout*, *TextFrame*, *EveryTextColumn*
FrameForm:	*Rectangle*, *Ellipse*, *Polygon*
HeightType:	*Exactly*, *Minimum*
HorizontalAlignment:	*Left*, *Center*, *Right*
HyphenationQuality:	*StandardQuality*, *GoodQuality*, *BestQuality*, *BestAvailableQuality*
ChangeOrderType:	*ToFront*, *Forward*, *Backward*, *ToBack*
CharacterBaselineOrientation:	*Auto*, *Horizontal*, *Vertical*

CharacterFunction:	*NormalCharacter*, *Tab*, *BorderTab*, *Indent*, *EmSpace*, *EnSpacer*, *ThreePerEmSpace*, *FourPerEmSpace*, *SixPerEmSpace*, *ThinSpace*, *HairSpace*, *PunctuationSpace*, *NoBreakingEmSpace*, *EnSpaceNonBreaking*, *ThreePerEmSpaceNonBreaking*, *FourPerEmSpaceNonBreaking*, *SixPerEmSpaceNonBreaking*, *ThinSpaceNonBreaking*, *HairSpaceNonBreaking*, *PunctuationSpaceNonBreaking*, *SpaceNonBreaking*, *FixedSpaceNonBreaking*, *FigureSpace*, *ZeroWidthSpace*, *ZeroWidthNonJoiner*, *ZeroWidthJoiner*, *EmDash*, *EnDash*, *Hyphen*, *DiscretionaryHyphen*, *EmDashNonBreaking*, *EnDashNonBreaking*, *HyphenNonBreaking*, *LineBreak*, *ParagraphBreak*, *LayoutColumnBreak*, *TextColumnBreak*, *TextObjectBreak*, *LayoutBreak*, *ChapterBreak*

Indent: indent to here; *EnSpacer*: en space; *ZeroWidthSpace*, *ZeroWidthNonJoiner*, *ZeroWidthJoiner*: the Unicode characters of the same name (the non-joiner prevents a ligature or a joined letter form, the joiner forces it).

CharacterOutlineJoinStyle:	*MiterJoin*, *BevelJoin*, *RoundJoin*
CharacterPosition:	*Normal*, *Subscript*, *Superscript*, *Superior*
CharacterRotation:	*Auto*, *Angle0*, *Angle90*, *Angle180*, *Angle270*
CharacterUpperLowerCase:	*Normal*, *SmallCaps*, *Caps*, *LowerCase*, *TitleCase*
ImageFormat:	*Format_Invalid*, *Format_Mono*, *Format_MonoLSB*, *Format_Indexed8*, *Format_RGB32*, *Format_ARGB32*, *Format_ARGB32_Premultiplied*, *Format_RGB16*, *Format_ARGB8565_Premultiplied*, *Format_RGB666*, *Format_ARGB6666_Premultiplied*, *Format_RGB555*, *Format_ARGB8555_Premultiplied*, *Format_RGB888*, *Format_RGB444*, *Format_ARGB4444_Premultiplied*, *Format_RGBX8888*, *Format_RGBA8888*, *Format_RGBA8888_Premultiplied*, *Format_BGR30*, *Format_A2BGR30_Premultiplied*, *Format_RGB30*, *Format_A2RGB30_Premultiplied*, *Format_Alpha8*, *Format_Grayscale8*, *Format_RGBX64*, *Format_RGBA64*, *Format_RGBA64_Premultiplied*, *Format_Grayscale16*, *Format_BGR888*
IndexModel:	*InvalidIndex*, *Pantone*, *Focoltone*, *Trumatch*, *HKS_K*, *HKS_E*, *HKS_Z*, *HKS_N*, *Toyo*, *Dic*, *Web*, *FirstCustomIndexModel*
KashidaLevel:	*None*, *Short*, *Medium*, *Long*, *Stylistic*

The kashida level of a paragraph (justification of Arabic text by longer connections between the letters): *None* switches kashidas off; *Short*, *Medium* and *Long* stretch a word by up to one, two or three tatweel widths; *Stylistic* stretches it by the rank of the connection, as in calligraphy.

KerningType:	*Automatic*, *Manual*

Language:	*NoLang*, *Afrikaans_South_Africa*, *Albanian*, *Amharic*, *AncientGreek*, *Arabic*, *Aragonese_ES*, *Armenian*, *Assamese*, *Azerbaijani*, *Basque*, *Belarusian*, *Bengali*, *Bosnian*, *Breton*, *Bulgarian*, *Burmese*, *Catalan*, *CentralKurdish*, *Chichewa*, *Chinese*, *Chinese_HK*, *Chinese_TW*, *ChineseTraditional*, *Croatian*, *Czech*, *Danish*, *Dhivehi*, *Dutch*, *Dutch_Secondary*, *English*, *English_AU*, *English_CA*, *English_GB*, *English_NZ*, *English_US*, *English_UK_ise*, *English_UK_ize*, *Esperanto*, *Estonian*, *Faroese*, *Finnish*, *French*, *French_BE*, *French_UppercaseUnaccented*, *Galician*, *Georgian*, *German_AT*, *German_CH*, *German_CH_Reformed*, *German_DE*, *German_DE_Reformed*, *German_DE_Combined*, *Greek*, *Guarani*, *Gujarati*, *Hausa*, *Hebrew*, *Hindi*, *Hungarian*, *Icelandic*, *Igbo*, *Indonesian*, *Irish*, *Italian*, *Japanese*, *Kannada*, *Kazakh*, *Khmer*, *Kinyarwanda*, *Kiswahili*, *Korean*, *Kurdish*, *Kyrgyz*, *Lao*, *Latin*, *Lengadocian*, *Lithuanian*, *Latvian*, *Luxembourgish*, *Macedonian*, *Malagasy*, *Malay*, *Malayalam*, *Maltese*, *Maori*, *Marathi*, *MongolianCyrillic*, *Nepali*, *Norwegian*, *Norwegian_Nynorsk*, *Odia*, *Pashto*, *Persian*, *Polish*, *Portuguese*, *Portuguese_Brazilian*, *Punjabi*, *Romanian*, *Romansh*, *Russian*, *Russian_Je*, *Russian_Jo*, *Sanskrit*, *SerbianCyrillic*, *SerbianLatin*, *Setswana*, *ScottishGaelic*, *Sindhi*, *Sinhala*, *Slovak*, *Slovenian*, *Somali*, *Spanish*, *Spanish_MX*, *Spanish_UppercaseAccented*, *Swahili_TZ*, *Swedish*, *Swedish_FI*, *Tagalog*, *Tajik*, *Tamil*, *Telugu*, *Tetum*, *Thai*, *Tibetan*, *Tigrinya*, *Turkish*, *Ukrainian*, *Urdu*, *Uyghur*, *Uzbek*, *Vietnamese*, *Welsh*, *WesternFrisian*, *Yiddish*, *Yoruba*, *Zulu*, *Universal*

The languages have fixed numbers: *NoLang* = 0 to *Turkish* = 93, *Universal* = 94, and *Tamil* = 200 to *Chinese_HK* = 243. Use the names in scripts.

LayoutColumnsType:	*None*, *Automatic*, *Manual*
LayoutFootnotesMode:	*None*, *ColumnDependent*, *LayoutDependent*
LayoutLineDirection:	*None*, *TopToBottom*, *LeftToRight*, *RightToLeft*
LayoutLineJustification:	*None*, *Beginning*, *Middle*, *End*
LineCap:	*Normal*, *ArrowHead*, *Bullet*, *ArrowEnd*
LineEditType:	*Double, Integer, String*
LineForm:	*FreeLine* (simple line), *ConstrainedLine*: (line with angles 0°, 45°, 90°, 135°, 180°), *PolyLine* (line composed of multiple lines (path))
LineNumbersMode:	*None*, *Left*, *Right*
LineNumbersRestartType:	*Continue*, *RestartEveryLayoutColumn*, *RestartEveryLayout*, *RestartEveryTextColumn*, *RestartEveryTextObject*, *RestartEveryChapter*
LinkAction:	*GotoPageNumber*, *GotoFirstPage*, *GotoPreviousPage*, *GotoNextPage*, *GotoLastPage*
LinkDestination:	*None, Page, Url*
MoveReferencePoint:	*OriginTopLeft, BBoxTopLeft*
NoteType:	*Footnote*, *Endnote*

ObjectType:	*All* (for getobject list only), *Text*, *Picture*, *Table*, *Graphic*, *Line*, *Group*, *Alias*, *TextAlias*, *PictureAlias*
OpenFileMode:	*ReadOnly*, *WriteOnly*, *ReadWrite*, *Append*
OpenTypePlacement:	*Undefined*, *None*, *Denominators*, *Numerators*, *Superscript*, *Subscript*
OpenTypeNumerals:	*Undefined*, *None*, *TabularFigures*, *OldstyleFigures*, *LiningFigures*, *ProportionalFigures*
OpenTypePositionalForms:	*Undefined*, *None*, *TerminalForms*, *InitialForms*, *MedialForms*, *IsolatedForms*
OpticalAlignment:	*None*, *Font*, *Auto*
PageInsertPosition:	*BeforeCurrentPage*, *AfterCurrentPage*, *AtStartOfDocument*, *AtEndOfDocument*, *AfterPage*
PageMode:	*AliasPages*, *SinglePages*, *DoublePages*, *Spreads*
PageNumberingFormat:	*None*, *Arabic*, *UpperLetter*, *LowerLetter*, *UpperRoman*, *LowerRoman*, *LeadingZeros2*, *LeadingZeros3*, *LeadingZeros4*, *Abjad*, *AlifBaTa*, *HebrewBiblical*, *HebrewNonStandard*
Examples: *Arabic* 1 2 3 4, *UpperLetter* A B C D, *LowerLetter* a b c d, *UpperRoman* I II III IV, *LowerRoman* i ii iii iv, *LeadingZeros2* 01 02 03 (*LeadingZeros3* and *LeadingZeros4* with three or four digits). *Abjad* and *AlifBaTa*: Arabic letters in abjad or alphabetical order; *HebrewBiblical* and *HebrewNonStandard*: Hebrew numerals.	
PageOrder:	*LeftRight*, *RightLeft*
PageOrientation:	*Portrait*, *Landscape*
PageType:	*SinglePage*, *FacingPages*
PageSize:	*A0*, *A1*, *A2*, *A3*, *A4*, *A5*, *A6*, *A7*, *B5*, *USLetter*, *USLegal*, *PF_11_17*, *CompactDisk*, *LetterHalf*, *Custom*
ParagraphAlignment:	*Left*, *Right*, *Center*, *Justified*, *ForcedJustified*
ParagraphRuleLengthType:	*Text*, *Column*, *Space*
PDFCompatibility:	*PDF_13*, *PDF_14*, *PDF_15*, *PDF_16*, *PDF_17*, *PDF_17ext3*, *PDF_17ext8*, *PDF_20*, *PDFX1_2001*, *PDFX1a_2001*, *PDFX1a_2003*, *PDFX2_2003*, *PDFX3_2002*, *PDFX3_2003*, *PDFX4*, *PDFX4p*, *PDFX5g*, *PDFX5pg*
PDFCompressionMethod:	*None*, *ZIP*, *JPEG*
PDFColorConversion:	*None*, *Always*, *PreserverNumbers*
None: no color conversion; *Always*: conversion to the target profile; *PreserverNumbers*: conversion to the target profile with "Preserve Numbers", as in the PDF export dialog.	
PictureFormat:	*TIFF*, *JPEG*, *PNG*
PictureHorizontalPosition:	*Ignore*, *Left*, *Center*, *Right*
PictureVerticalPosition:	*Ignore*, *Top*, *Center*, *Bottom*
PolygonPointFlag:	*After*, *Default*, *Control*, *Before*, *Constraint*, *First*, *Last*, *Align*

The values are flags and can be combined: ***After*** = ***Default*** = 0, ***Control*** = 1, ***Before*** = 2, ***Constraint*** = 4, ***First*** = 8, ***Last*** = 16, ***Align*** = 32.

PrintDeviceType:	*LaserPrinter*, *ImageSetter*, *FilmRecorder*, *ColorPrinter*, *ColorProof*, *Plotter*, *ComputerToPlate*
PrintImageType:	*NoMirroring*, *Mirroring*
PrintMaterialType:	*Paper*, *Film*, *Foil*
PrintResolutionType:	*Low*, *High*, *R300dpi*, *R600dpi*, *R1280dpi*, *R2400dpi*
PrintPagesRangeType:	*CurrentPage, PagesRange, PageSelection*
PrintLayersRange:	*AllLayers*, *VisibleLayers*, *PrintableLayers*, *VisibleAndPrintableLayers*
PrintPageNumbering:	*Physical*, *Logical*, *PhysicalAndLogical*
PrintColorSeparation:	*Off*, *App*, *InRIP*

Off: composite output; ***App***: color separations made by VivaDesigner; ***InRIP***: color separations in the RIP.

PrintHorizontalAlignment:	*Left*, *Center*, *Right*
PrintVerticalAlignment:	*Top*, *Center*, *Bottom*
PrintFontEmbedding:	*None*, *AllButBase14*, *OnlyType1*, *All*
PrintImageQuality:	*None*, *LoRes*, *HiRes*
PrintOPI:	*None*, *TIFF*, *EPS*, *Both*, *Generic*

The values have no effect: VivaDesigner does not create OPI information.

PrintNameChangePolicy:	*AppendPageNumber*, *ReplaceWithPageNumber*, *ReplaceWithPageName*
ProgressReporterType:	*NoItems*, *TextItem*, *TwoTextItems*, *CancelButton*

The values are flags and can be added: ***NoItems*** = 0, ***TextItem*** = 1, ***TwoTextItems*** = 3, ***CancelButton*** = 4 (a Cancel button in the progress window).

ReferenceAnchorNumberingType:	*ContinuousNumbering*, *ChapterWiseNumbering*, *PageRestartNumbering*, *SpreadRestartNumbering*, *SectionRestartNumbering*

The numbering of notes starts anew: never, in every chapter, on every page, on every spread or in every section of the page numbering.

RubyPosition:	*Above*, *Before*, *Below*, *After*, *Hidden*
RunaroundMode:	*None*, *Left*, *Right*, *All*
RunaroundShapeType:	*ObjectContours*, *Block*, *ImageBoundaries*, *ClippingPath*
ScalePictureType:	*Proportional*, *FitToFrame*, *FillFrameProportionally*, *FitToPicture*, *OriginalSize*
ScreenAngle:	*Cyan*, *Magenta*, *Yellow*, *Black*

SingleQuotesType:	*InvalidSingleQuotes*, *SingleQuotes1*, *SingleQuotes2*, *SingleQuotes3*, *SingleQuotes4*, *SingleQuotes5*, *SingleQuotes6*, *SingleQuotes7*, *SingleQuotes8*, *SingleQuotes9*, *SingleQuotes10*, *SingleQuotes11*, *SingleQuotes12*, *SingleQuotes13*, *SingleQuotes14*, *SingleQuotes15*

The pairs: *SingleQuotes1* ‘ ’, *SingleQuotes2* , ‘, *SingleQuotes3* ’ ’, *SingleQuotes4* , ’, *SingleQuotes5* ‹ ›, *SingleQuotes6* › ‹, *SingleQuotes7* ' ' (default), *SingleQuotes8* white corner brackets, *SingleQuotes9* ’ ‘, *SingleQuotes10* “ ”, *SingleQuotes11* « », *SingleQuotes12* » «, *SingleQuotes13* „ “, *SingleQuotes14* ‘ ‘, *SingleQuotes15* corner brackets. The values have the numbers 100 to 114.

SpecialCharacters:	*SingleQuote1*, *SingleQuote2*, *SingleQuote3*, *SingleQuote4*, *SingleQuote5*, *SingleQuote6*, *DoubleQuote1*, *DoubleQuote2*, *DoubleQuote3*, *DoubleQuote4*, *DoubleQuote5*, *DoubleQuote6*, *Bullet*, *ATSign*, *SectionSign*, *BaskslashSign*, *VerticalLine*, *PerMilleSign*, *HorizontalEllipsis*, *EuroSign*, *DollarSign*, *PoundSign*, *YenSign*, *CopyrightSymbol*, *RegisteredSymbol*, *TrademarkSymbol*
StyleSheetFamily:	*Character*, *Paragraph*, *Layout*, *Graphic*, *Picture*, *Table*, *TableRow*, *TableColumn*, *TableCell*
TableLineSeparator:	*None*, *Vertical*, *Horizontal*
TableSeparatorPosition:	*Left*, *Top*, *Right*, *Bottom*
TabsType:	*None*, *Automatic*, *Manual*
TabulatorAlignment:	*Begin*, *Center*, *End*, *Character*
TaggedPdfType:	*Automatic*, *Artifact*, *Figure*

How an object is exported into a tagged PDF file: *Automatic* (default), *Artifact* (decoration) or *Figure* (with alternative text). See the class *TaggedPdfOptions*.

TextColumnLineAnchor:	*ObjectFrame*, *Baseline*
TransparencyType:	*Linear*, *Radial*
TypographicQuotesMode:	*TypographicQuotesOff*, *TypographicQuotesOn*, *TypographicQuotesPref*
VariableType:	*Date*, *Note*, *NoteNumber*, *DocumentCreationDate*, *DocumentModificationDate*, *RunningTitle*
VerticalAlignment:	*Top*, *Center*, *Bottom* The values define the vertical alignment of an object in the table cell. Of course the top and bottom indents are also included in the calculation.
VerticalGuideZone:	*BeforePages, AfterPages*
WhichObjects:	*Selected*, *TopLevel*, *All* Defines a filter for the list of objects. Available values:
WhichSpreads:	*Document*, *Alias*, *All* Defines a filter for the list of objects. Available values:
WidowsOrphansMode:	*None*, *WholeParagraph*, *FirstAndLastLines*

`WidthType`:	*`Auto`* (The width of the column is dynamic and adjusted by the real content of table cells.) *`Manual`* (Fixed width of the table column. If the content of column cells exceeds the width of the column, it is cropped.) *`Percent`* (The value of the property *`width`* is interpreted as a percentage value and means that the width of the column is related to the width of the table.)
`WritingDirection`:	*`LeftToRight`*, *`RightToLeft`*

Index

Version 11.5 — November 2026

Index

A

absolutePath 12
accept 217
acceptLabel 215
acceptMode 215
activatedMethod 220
active148, 171, 172, 173, 174, 178, 179, 180, 181, 182, 184, 186, 187, 190, 192, 193, 194, 196, 197, 200, 201, 202, 205
add 23, 28, 79
addAcceptButton 217
addAliasPage 18
addBlend 128
addButton 217
addButtons 217
addCheckBox 217
addColor 128
addComboBox 217
addDocumentPage 18
addGroupBox 217
addHorizontalGuide 29
addItem 220, 221
addLabel 217
addLayer() 19
addLineEdit 217
addOption 215
addPath 158
addRadioButton 221
addRejectButton 217
addSeparator 221
addTabulator 202
addTextEdit 217
addVerticalGuide 29
adjustments 59, 144
afterPageNumber 156
aliasObjectsSelectable 163
aliasPage 25
aliasPageAt 18
aliasPageByName 18
aliasPageCount 15
aliasSp 15
aliasSpreadAt 19
alignCharacter 202
alignment 176, 202
alternatingColumns 147
alternatingColumnsData 147
alternatingRows 147
alternatingRowsData 147
alternativeText 200, 201, 202, 203
anchorText 188
angle 198, 205
annotation 88
annotationTextRange 170
appendPageNumber 201
appendPlainText 86
appendPoint 124
appendStop 130, 205
applyAsIndent 202
applyElementOpacity 198
applyStandardTabs 202
applyValue 172
appScriptsPath 8
areaAt 65
areaCount 65
areaIndex 65
asChart() 79
askForImages 163
at 39, 64, 119, 128, 136
AttributeGroup 226
Attributes() 171
author 157, 174
AutoLineSpacingMethod 226
automaticCharacterSpacing 106, 107, 141
AutomaticCharacterSpacing() 171
automaticCharacterWidth 107, 141
AutomaticCharacterWidth() 171
automaticGrammarCheck 166
automaticSpellCheck 166
automaticTextObject 154
automaticWordSpacing 106, 141
AutomaticWordSpacing() 172

B

backgroundMaskVisible 62
baselineDescenders 54, 146
baselineGrid 54, 146, 167
BaselineGrid 172
BaselineGrid() 172
BaselineGridDirection 226
BaselineGridNumbering 226
baselineGridType 106, 142
BaselineGridType 226
BaselineOffset 226
baselinePosition 54, 145
BaselinePosition 172
BaselineType 226
baseOpacity 145
baseTextRange 170
bcp47 198
begin 120
bevelEmboss 46, 146
bleed 190
bleedBottom 195
bleedLeft 195
bleedRight 195
bleedTop 195
blendAngle 70, 71, 72
BlendType 226
blue 132
blur 184, 190, 198
bottom 197
bottomAnchor 174
bottomBorder 193
bottomIndent 54, 73, 150, 174, 180, 181, 192, 193
bottomMargin 154
bottomSeparator 150
bottomSeparatorData 150
bottomTextIndent 145
brush . . 32, 34, 70, 72, 73, 76, 93, 138, 140, 180, 181, 183, 190, 192, 193, 194, 199, 206
brushes 15
bullet 176
BulletCharacters 226
bwImageResolution 190
bwImagesMinimumResolution 164

C

caption 215, 216
cell 66
changeColorSpace 63
changeDir 210
changeOrder 43
ChangeOrderType 228
ChangePictureColorSpaceSettings(); 179
changeTracking 167
chapterInfo 25
ChapterInfo 156
ChapterInfo() 156
chapterName 114
chapters 115
characterBackground 103, 139, 141
CharacterBackground 180
characterBaselineOrientation 99, 139, 140
CharacterBaselineOrientation 228
characterBaselineShift 99, 138, 140
characterCount 175
characterDigits 101, 139, 142
characterFrame 103, 139, 141
CharacterFrame 181
CharacterFrame() 181
characterFunction 89
CharacterFunction 229
characterHeightScale 97, 138, 140
characterKerning 98, 138, 140
CharacterKerning() 182
characterLigatures 100, 139, 140
characterNoKashida 100, 101, 139, 142
CharacterOutlineJoinStyle 229
characterPosition 96, 138, 140
CharacterPosition 229
characterPosition(int begin, int end) 96
characterPosition(int pos) 96
CharacterPositionCorrection 182
CharacterPositionCorrection() 182
characterRotation 100, 139, 140
CharacterRotation 229
characterRule 104, 139, 141
CharacterRule 182
CharacterRule() 182
characterSpacing 98, 138, 140
characterStyleSheet 104, 140, 167, 176
characterStyleSheetByName 105, 167, 176
characterUpperLowerCase 96, 138, 140
CharacterUpperLowerCase 229
characterWidthScale 97, 138, 140
charAsCode 87
charAsString 87
cleanupFolder 191
clear() 90, 137
clearGuides() 30
clickedMethod 221, 222, 224
clippingEnabled 59
clippingTight 59
clippingTolerance 59
close 19, 212
close() 216
closeAndSave() 19
cmyk 129
collectionAt 13
collectionCount 12
color 173, 184, 190, 197
Color 226
colorBrush 131
ColorBrush 129
colorDepth 163
ColorDepth 226
colorImageResolution 190
colorImagesMinimumResolution 164
colorMode 59
ColorMode 226
colorModel 129
ColorModel 226
colorOverlay 45, 146
ColorOverlay() 173
colorProfile 179
colorRgb 32
colorSeparation 129, 157, 195
ColorSeparation 226

ColorSeparationState 226
colorSpace 59, 179
ColorSpace
column 66, 74
columnCount 54, 66, 146, 178, 185, 214
columnDistance 178
columnGutter 146, 155
columnIndex 214
columnLines 54, 146
ColumnLines 173
ColumnLines() 173
columnOffset 54
columns 143
columnSpan 73
commands 14
comment 88
Comment() 174
connectTo 52
containsUserProperty 20, 24, 29, 43
content 52, 57, 83
contentExpandableObject 73
contentObject 73
contentOpacity 52, 57, 145
contentRotation 73
ContentRotation 226
contentType 73
ContentType 226
convertUnitValue 8
copy 211
copy() 43, 90
copyAsAlias 156
copyAttributes 44, 55, 63, 71, 72, 74, 77
copyContentFrom 63
copyObjectPreferences 20
copyObjects 156
copyOnlySelectedObjects 156
copyUserProperties 42
count 148, 184, 202, 210, 220
Count 15
Courier Bold 2
coverShadow 198
create 39, 136
createAlias() 43
createChart 26
creationDate 174
crop 189
cropMarkDistance 195
cropMarkLength 195
cropmarks 195
cropsHeight 34
cropsWidth 34
cropsX 34
cropsY 34
currentCollection 12
currentDocument 12
currentIndex 220
currentIndexChangedMethod 220
currentLayer 17
currentPage 16
currentPictureObject 17
currentSpread 16
currentText 17
cursorPositionChangedMethod 222, 223
customHeight 154
customProperty 43
customPropertyBoolean 43
customWidth 154
cyan 132

D

DataTable 214
deadline 158
deleteObjects() 23
deletePoint 124
deleteSelectedObjects() 23
description 21, 155
destination 187
Dialog 216
DialogItemType 226
Digits 226
direction 172
directory 215
Directory 210
directoryFilter 215
DirectoryFilter 226
DirectorySortFlag
disable 219, 221, 224
disable() 222
disableSystemFonts 162
displayPosition 203
distance 197
distanceBetweenStandardTabs 166
distanceToText 175
documentAt 13
documentCount 12, 21
documentExportFormats 13
documentImportFormats 12
documentName 187
documentOpenFormats 12
documentPageAt 18
documentPageCount 15
documentPathAt 21
documentSaveFormats 13
documentSettings 17
DocumentSettings 154
documentSp 15
documentSpreadAt 18
documentStatistics 16
DocumentStatistics 157
done 218
doubleQuotesType 162
DoubleQuotesType 227
dropCaps 108, 141
DropCaps 174
DropCaps() 174
DropCapsLinesType 227
DropCapsRunaroundType 227
DropCapsType 227
dropShadow 45, 146
dump 8
dx 126
dy 126

E

editable 80
editingFinishedMethod 222
embedAllImages 163
embedAllUsedFonts 162
embeddedFontsPreferences 16
enable 219, 221, 222, 224
end 120
endnoteLabelStyleSheet 167
endnoteLabelStyleSheetByName 167
endnoteReferenceStyleSheet 167
endnoteReferenceStyleSheetByName 167
endnotes 167
EndnotesBreakType 227
EndnotesPositionType 227
endnotesRanges 116
endnoteTextStyleSheet 167
endnoteTextStyleSheetByName 167
enumerations 108, 141
Enumerations 176
Enumerations() 176
EnumerationsAlignment 227
EnumerationsNumberFormat 228
EnumerationsNumberingMode 227
EnumerationsType 227
eof 212
equals 65
ErrorStatus 228
exec 217
execute 8
exists 158, 210, 211
exportRange 68
exportToPDF 20, 22
exportToXML 19
exposure 157
ExposureType 228
extendToColumnBorder 192, 193
extendToNextParagraph 192, 193
ExtendToNextParagraphType 228

F

fax 158
featuresString 189
File 211
FileDialog 215
FileDialogAcceptMode 228
FileDialogFileMode 228
FileDialogOption 228
FileDialogViewMode 228
fileInfo 68
fileMode 215
fileName 211
fileNameLabel 215
filePath 12, 16, 21, 58
fileTypeLabel 215
fillBlendAngle 50, 145, 147, 149, 150
fillBrush 145, 147, 149, 150
fillCharacter 202
fillColor 50
fillOpacity 50, 145
fillShade 50, 145, 147, 149, 150
find 90, 128, 136
findAndReplace 90
findAndReplaceAll 90
findAndReplaceAllRegEx 90
findAndReplaceRegEx 90
findRegEx 90
firstBaseline 167, 172
firstCount 148
firstStyleSheet 148
firstStyleSheetByName 148
fixedHeight 223
fixedWidth 220, 223
flagAfter 123
flagAfterConstraint 123
flagBefore 123
flagBeforeConstraint 123
flagFirst 123
flagLast 123
flags 122
flush 212
focusPosition 131, 205
font 91, 138, 140
fontEmbedding 195
FontReferenceSizeType 228
fontSize 92, 138, 140
FontSize 178
fontsSubfolder 191
FontStyleLineRangeType 228
FontStyleLineType 228
footer 55, 146
footerFormattedText 55
footerRows 147
footerRowsData 147
footnoteLabelStyleSheet 167
footnoteLabelStyleSheetByName 167
footnoteReferenceStyleSheet 167
footnoteReferenceStyleSheetByName 167
footnotes 55, 143, 146, 167
FootnotesPosition 228
footnoteTextStyleSheet 167
footnoteTextStyleSheetByName 167

forceLineBreak 166
forceRedraw 216
form 47, 49
format 179, 190
formattedText 54, 87
frameColor 50
FrameForm 228
frameOpacity 50
frameRadius 50
frameShade 50
frameStyle 50
frameWidth 50, 181, 193
fromPage 195
fullCopy() 43
fullPath 210

G

getAttributes 44, 55, 63, 71, 72, 74, 77, 91
getExistingDirectory 215
getFileList 211
getObject 17, 23, 28, 32, 78
getOpenFileName 215
getPoint 124
getSaveFileName 216
getSaveFileName() 216
getTabulator 202
gotoPage 18
gradientOverlay 46, 146
GraphicFootnotes() 178
graphicStyleSheet 163
grayImageResolution 190
grayImagesMinimumResolution 164
greekBelow 166
green 132
groupObject 42
guide 24, 30, 41
guideCount 23, 28
guideObject 145
guideObjectsMagnetic 163
guideObjectsSelectable 163

H

halftoneAngle 59, 144
halftoneScreen 59, 144
hasBackgroundMask 62
header 55, 146
HeaderFooter() 178
headerFormattedText 55
headerRows 147
headerRowsData 147
height 23, 49, 58, 70, 126, 145, 178, 223
heightType 70
HeightType 228
hidden 31, 129
hide 219, 221, 222, 224
highlightedMethod 220
horizontalAlignment 33, 73, 150, 195
HorizontalAlignment 228
horizontalCorrection 182
horizontalMirrored 59
horizontalOffset 58
horizontalPasteOffset 163
horizontalScale 58, 144
horizontalSeparator 149
horizontalSeparatorAbove 66
horizontalSeparatorData 149
hsv 129
Hsv 132
hue 132
hyphenation 108, 141
Hyphenation 178
Hyphenation() 178
hyphenationModuleIdent 162
HyphenationQuality 228

I

iconFrame 220
iconHeight 220
iconWidth 220
id 42, 219, 221, 222, 223, 224
identifier 12
ignoreTrapping 195
image 62
Image() 183
Image(String data) 183
imageData 63
ImageFormat 229
imageQuality 195
imagesSubfolder 191
importPicture 62
importRange 67
include 7
includeDocument 191
includeFonts 191
includeImages 191
includeProtocol 191
indexKey 129
indexModel 129
IndexModel 229
individualAliasContent 52, 57, 145
info() 80
information (. 210
innerGlow 45, 146
InnerGlow() 184
innerMargin 154
innerShadow 45, 146
inputRejectedMethod 222
insertAfter 78
insertAnnotation 88
insertAt 78
insertCharacter 87
insertComment 88
insertFormattedText 88
insertHtml 87
insertNote 89
insertPlainText 55, 86
insertPoint 124
insertPosition 156
insertSeparator 221
insertVariable 89
isAbsolute 210
isAliasPage 22
isChart 80
isEmbedded 59
isEmpty 59
isEnabled 219, 220, 221, 222, 223, 224
isGroup 124
isHidden 219, 220, 221, 222, 223, 224
isNull 33
isOnline 38
isPlacedDocument 62
isReadable 210
isRelative 210
isRoot 210
itemKey 221

J

joinStyle 191
justifiedSpaceBetweenLinesMaximum 112, 143

K

kashidaLevel 109, 142
KashidaLevel 229
kashidaRules 109, 142
keepParagraphTogether 111, 142
keepRunaround 31
KerningType 229
key 132
keywords 21, 155

L

lab 129
Lab 133
label 25
landscape 23
language 21, 93, 138, 140, 155, 162, 166
Language 230
languageByName 92
languageInfo 162
languagePreferences 16
lastResult 80
layer 42
layerAt 19
layerCount 15
layerOverrides 59
layersRange 195
layoutColumns 113
LayoutColumnsInfo() 184
LayoutColumnsType 230
layoutFootnotes 113
LayoutFootnotes() 185
LayoutFootnotesMode 230
layoutGrid 154
LayoutLineDirection 230
layoutLineJustification 112
LayoutLineJustification 230
layoutLineOrientation 112
layoutPositionColumn 22
layoutPositionRow 22
layoutPreferences 16
layouts 115
layoutStyleSheet 113, 114, 167
layoutStyleSheetByName 114, 167
left 197
leftBorder 193
leftColumns 147
leftColumnsData 147
leftIndent 54, 73, 150, 180, 181, 183, 192, 193, 194
leftMargin 155
leftSeparator 150
leftSeparatorData 150
leftTextIndent 145
length 38, 64, 78, 86, 119, 120, 124, 128, 136, 158
lengthType 194
level 176, 200, 202
lightness 133
lineAngle 47
lineBrush 147, 186
LineCap 230
lineColor 47, 145, 173
lineCornerRadius 145
LineEditType 230
lineEnd 48, 145
LineForm 230
lineJustification 143
lineLength 47
LineNumbers 186
LineNumbers() 186
lineNumbersColor 166
lineNumbersData 166
lineNumbersFont 166
lineNumbersFontSize 166
lineNumbersMode 166
LineNumbersMode 230
LineNumbersRestartType 230
lineOpacity 47, 145, 174
lineOrientation 143
lineShade 47, 145, 147, 173, 186
lineStart 48, 145
lineStyle . . 47, 145, 147, 174, 181, 183, 186, 191, 193, 194, 199, 206
LineStyle 186
LineStyle() 186
linesType 174
lineWidth . .47, 145, 147, 174, 183, 186, 191, 194, 199, 206

link 103, 139, 141
Link 187
Link() 187
linkableWithOriginal 54, 145
linkAction 187
LinkAction 230
LinkDestination 230
linkToTextChain 156
loadExcelSheet 214
localizedName 129
locked 31, 41
logicalNumber 25
lookInLabel 215

M

m11 126
m12 126
m21 126
m22 126
magenta 132
mainText 170, 188
makeDir 210
margin 185
marginAt 184
matrix 41
Matrix 126
Matrix() 126
maxHyphensInARow 179
maximum 171, 172
maximumLines 175
maximumPreviews 164
maximumWidth 217
message) 210
messageBox.information("Msg
", "Point
[" + p.x + "," + p.y + "," + p.flags + ...
. . . . . ."]" + "\nFlagAfter
" + p.flagAfter + ... 125
mimeTypeFilter 215
minimum 171, 172
minimumLines 175
minimumWidth 216
minPrefix 179
minSuffix 179
mirrored 41, 145
mirroredHorizontally 144
mirroredVertically 144
mode 173, 176, 184, 185, 186, 190, 197, 198
model 80
modificationDate 174
move 44
moveLayer 19
moveOnPageTo 43
moveOnSpreadTo 44
movePicture 63
MoveReferencePoint 230

N

name 28, 30, 31, 40, 137, 198, 210
nameChangePolicy 195
nameFilter 215
new BoundingBox(Array values) 125
new Cmyk 132
new PDFExportSettings() 194
new Rgb 131
new TextEndnotes 203
new TextFootnotes 204
newDocument 13
newDocument() 13
newName) 211
newPageNumber 156
"\nFlagBeforeConstraint
" + p.flagBeforeConstraint + ... 125
note 41, 88
notesVisible 86
NoteType 230
numberFormat 176, 203, 204
numbering 172
numberingFormat 156
numberingType 203, 204
numberOfColumns 155
numberOfDigits 195
numberOfSizingPoints 124
numberOfSubPolygons 124

O

object 87
objectCount 17, 23, 28, 32, 78
objectOpacity 50
objectPage 29
objectPreferences 16
objects 17, 23, 28, 78
objects(ObjectType objectType = All) 32
objectType 78
ObjectType 231
offset 183, 186, 194, 196, 198, 199, 206
offsetOnSpreadX 22
offsetOnSpreadY 22
offsetX 144
offsetY 144
oldName, ... 211
on 176, 186
opacity . . . 94, 131, 138, 140, 173, 180, 181, 183, 184, 190,
191, 192, 193, 194, 196, 198, 199, 205, 206
open 212, 215
openCollection 13
openDocument 13
OpenFileMode 231
openType 102, 103, 139, 141
OpenType() 188
OpenTypeNumerals 231
OpenTypePlacement 231
OpenTypePositionalForms 231
opi 195
opticalAlignment 108, 141
OpticalAlignment 231
optimizePicture 63
OptimizePictureSettings() 189
optimum 171, 172
orderedBy 157
orderedObjects() 32
orderNumber 157
outerGlow 45, 146
OuterGlow() 190
outerMargin 155
outFilePath 180, 190
outlineOptions 94, 95, 139
OutlineOptions 190
OutlineOptions() 190
outlineOptionsData 141
outputPath 195
overwriteExistingFiles 191
owner 82

P

package 20
PackageSettings() 191
page 41
pageAt 29
pageBoundingBox 41
pageBox 62
pageCount 28
pageFormat 23
pageGeometryPoints 47, 49
pageHeight 17
PageInsertPosition 231
pageMode 15
PageMode 231
pageNumber 187
pageNumbering 195
PageNumberingFormat 231
pageOrder 154
PageOrder 231
pageOrientation 154
PageOrientation 231
pagePreferences 16
pageSelection 195
PageSettings() 156
pageSize 154
PageSize 231
pagesPerJob 195
pagesRangeType 195
pageType 28, 154
PageType 231
pageWidth 17
pagex 40
pagey 40
ParagraphAlignment 231
paragraphAtCharacterPosition 115
paragraphBackground 110, 142
ParagraphBackground 191
ParagraphBackground() 191
paragraphFrame 110, 142
ParagraphFrame 192
ParagraphFrame() 192
paragraphIndent 107, 141
paragraphLeftIndent 107, 141
paragraphLineSpacing 106, 141
paragraphRightIndent 107, 141
ParagraphRule 194
ParagraphRule() 194
paragraphRuleAbove 110, 142
paragraphRuleBelow 111, 142
ParagraphRuleLengthType 231
paragraphs 115
paragraphSpaceMaximum 113, 143
paragraphStyleSheet 111, 167
paragraphStyleSheetByName 112, 167
paragraphTabs 107, 108, 142
parentId 220
path 210
pathAt 158
pathClipping 61
pdfColorConversion 196
PDFColorConversion
pdfCompatibility 196
PDFCompatibility 231
pdfCompressionMethod 196
PDFCompressionMethod 231
pdfCompressionQuality 196
pdfCompressText 196
pdfCreateBookmarks 196
pdfCreateLayers 196
pdfExportNotes 196
pdfOptimize 196
pdfPageNo 62
pdfTargetProfile 196
pdfWarningScale 196
pdfWarningScaleAbove 196
pdfWarningScaleBelow 196
pdfWarningTextOverflow 196
pdfxAllowEmbeddedICCProfiles 196
pdfxGrayProfile 196
pdfxOutputIntent 196
phone 158
physicalNumber 22
PictureFormat 231
PictureHorizontalPosition 231
pictureIndividualAliasContent 145
pictureStyleSheet 163
PictureVerticalPosition 231
placeMaximumColumnCount 185
placeStartColumn 185
plainText 55, 86
Point 122

PointList 124
PointList() 124
PolygonPointFlag 231
position 131, 170, 178, 188, 202, 205
positionCorrection 89
postfix 176, 180, 189, 203, 204
prefix 156, 176, 179, 189, 203, 204
preflightDocument 164
preflightPreferences 16
PreflightPreferences 164
PreflightPreferences() 164
preview 24, 25, 44, 62
previewQuality 164
printable 31, 40, 145
PrintColorSeparation 232
printDeviceType 157
PrintDeviceType 232
printEmptyPages 195
PrintFontEmbedding 232
PrintHorizontalAlignment 232
printImage 157
PrintImageQuality 232
PrintImageType 232
PrintLayersRange 232
printMaterial 157
PrintMaterialType 232
PrintNameChangePolicy 232
PrintOPI 232
PrintPageNumbering 232
PrintPagesRangeType 232
printResolution 157
PrintResolutionType 232
printSeparately 195
printSpellCorrection 195
printSpreads 195
PrintVerticalAlignment 232
progressBarFinish 216
progressBarLength 216
progressBarPosition 216
progressBarReset 216
progressBarSubText 216
progressBarText 216
ProgressReporter 216
ProgressReporterType
protected 41

Q

quality 179
quickPictureView 163
quickTextView 163

R

range 170
rangeType 199, 206
readLine 212
red 131
ReferenceAnchorNumberingType 232
reflection 45, 146
Reflection() 196
refresh 39
registrationMarks 195
reject 218
rejectLabel 215
remove 139, 142, 143, 144, 146, 148, 149, 150, 211
remove() 25, 28, 31, 52, 57, 67, 79, 83, 129, 137
removeAll() 158
removeAllTabulators 202
removeBackgroundMask() 63
removeContent() 74
removeDir 210
removeDocumentPages 18
removeGuide 30
removeOption 215
removePath 158
removeRange 90
removeStopAt 130, 205
removeStyleSheets 90
removeTabulator 202
rename 211
renameDir 211
resizable 217
resolutionX 58
resolutionY 58
responsible 158
restartType 186
returnPressedMethod 222
rgb 129
right 197
rightBorder 193
rightColumns 147
rightColumnsData 147
rightIndent 54, 73, 150, 180, 181, 183, 192, 193, 194
rightMargin 155
rightSeparator 150
rightSeparatorData 150
rightTextIndent 145
root 55
rotation 41, 58, 144, 145, 150
row 66, 74
rowAt 214
rowByFieldContent 214
RowColumnStyle 148
RowColumnStyle() 148
RowColumnStyleAlt 148
rowCount 65, 214
rowSpan 73
RubyPosition 232
runaround 42, 145
Runaround 197
Runaround() 197
RunaroundMode 232
RunaroundShapeType 232
runaroundType 175
runCommand 14

S

saturation 132
save 19, 22
saveLargePreview 164
saveSmallPreview 164
saveTo 203
saveToFile 63
scale 33, 157, 175, 198
scalePicture 63
ScalePictureType 232
screenAngle 129
ScreenAngle 232
searchPaths 16
secondCount 148
secondStyleSheet 149
secondStyleSheetByName 149
selected 41
selectedFiles 215
selection 120
selectionChangedMethod 222, 223
selections 120
separator 66, 147
separatorHorizontalOffset 204
separatorLineBrush 204, 205
separatorLineLength 204, 205
separatorLineStyle 204, 205
separatorLineWidth 204, 205
separatorVerticalOffset 204
separatorVisible 204
set 82
setAction 187
setAliasPage 25
setAlternatingColumns 148
setAlternatingRows 148
setAttributes 44, 55, 63, 71, 72, 74, 77, 91
setAutomaticCharacterSpacing 107
setAutomaticCharacterWidth 107
setAutomaticWordSpacing 106
setBackgroundMask 63
setBaselineGridType 106
setBlock 197
setBottomSeparator 150
setBrush 93
setBullet 177
setChapterName 114
setCharacterBackground 103
setCharacterBaselineOrientation 99
setCharacterBaselineShift 99
setCharacterDigits 101
setCharacterFrame 104
setCharacterFunction 89
setCharacterHeightScale 97
setCharacterKerning 98
setCharacterLigatures 100
setCharacterNoKashida 101
setCharacterPosition 96
setCharacterRotation 100
setCharacterRule 104
setCharacterSpacing 98
setCharacterStyleSheet 104
setCharacterStyleSheetByName 105
setCharacterUpperLowerCase 97
setCharacterWidthScale 97
setClipping 63
setCount 184
setCustomProperty 43
setData 81, 184
setDropCaps 108
setDynamicLines 175
setEnumerations 108
setFillStyle 67, 71, 72, 74
setFixLines 175
setFont 91, 92
setFontSize 92
setFooterRows 147
setFrameStyle 77
setHeaderRows 147
setHorizontalSeparator 149
setHyphenation 108
setImageData 63
setIndents 74
setItemKey 221
setJustifiedSpaceBetweenLinesMaximum 112, 113
setKashidaLevel 109
setKashidaRules 109
setKeepParagraphTogether 111
setLanguage 93
setLanguageByName 92
setLanguageInfo 162
setLayoutColumns 113
setLayoutFootnotes 113
setLayoutLineJustification 112
setLayoutLineOrientation 112
setLayoutStyleSheet 114
setLayoutStyleSheetByName 114
setLeftColumns 148
setLeftSeparator 150
setLink 103
setMarginAt 184
setObject 87
setOpacity 94
setOpenType 103
setOpticalAlignment 109
setOptions 81
setOutlineOptions 95
setParagraphBackground 110
setParagraphFrame 110
setParagraphIndent 107
setParagraphLeftIndent 107
setParagraphLineSpacing 106
setParagraphRightIndent 107

setParagraphRuleAbove 110
setParagraphRuleBelow 111
setParagraphSpaceMaximum 113
setParagraphStyleSheet 111
setParagraphStyleSheetByName 112
setParagraphTabs 108
setPoint 124
setPositionCorrection 89
setQrCodeToImage 13
setRange 120
setRightColumns 148
setRightSeparator 150
setShade 94
setSize 25
setSpaceToNextLayout 113
setSpaceToNextParagraph 106
setSpaceToPreviousLayout 113
setSpaceToPreviousParagraph 106
setStopAt 130, 205
setStoryEndnotes 115
setStoryFootnotes 115
setStoryLineCounterData 115
setStrikethrough 96
setSupressLineNumbering 110
setTablesAbbreviationEntry 102
setTablesBibliographyEntry 102
setTablesContentsEntry 102
setTablesIndexEntry 101, 102
setTablesPictureEntry 102
setTablesRunningTitleEntry 102
setTabulator 202
setTextAlignment 105
setTopSeparator 150
setUnderline 95
setUserProperty 20, 23, 29, 42
setUserXMLProperty 20, 24, 29, 42
setVerticalSeparator 149, 150
setWidowsAndOrphans 142
setWidowsOrphans 111
setWidthAt 184
setWritingDirection 108
shade .70, 71, 72, 76, 94, 131, 138, 140, 173, 180, 181, 183, 184, 190, 192, 193, 194, 197, 199, 206
Shadow() 197
shapeType 197
show 219, 221, 222, 224
showAliasObjects 163
showBaselineGrid 163
showGuideObjects 163
showGuides 163
showInvisibles 166
showMarginsAndColumns 163
showRuler 163
showSmartGuides 163
showTextRuler 163, 166
singleQuotesType 162
SingleQuotesType 233
size 178, 196, 211
skew 41, 59, 144, 145
skipFirstCount 149
skipLastCount 149
smallCapsCharacterHeight 166
smallCapsCharacterWidth 166
smallestWord 179
smallPreviewsMaximumSize 164
snapDistance 163
sourcePageCount 62
sourcePath 62
spaceToNextLayout 113, 143
spaceToNextParagraph 106, 141
spaceToPreviousLayout 113, 143
spaceToPreviousParagraph 106, 141
spacing 172
spans 115, 116
SpecialCharacters 233
SpellLang 198
spellModuleIdent 162
spread 22, 41
spreadBoundingBox 41
spreadGeometryPoints 47, 50
spreadx 40
spready 40
start 172
startColumn 178
startNumber 204
startsWith 186
startWith 176
stateChangedMethod 219
step 186
Stop 131, 205
Stop(. 205
stopAt 130, 205
stopCount 130, 205
story 116
storyEndnotes 114, 115
storyFootnotes 115
storyLineCounterData 115
strikethrough 95, 96, 138, 140
strikethroughLineWidth 165
strikethroughOffsetHorizontalLayout 165
strikethroughOffsetVerticalLayout 165
StrikethroughOptions 199
String fillCharacter) 202
stroke 45, 146
styleSheet 45, 59, 66, 70, 72, 74, 148
styleSheetByName 45, 59, 66, 70, 72, 74, 148
StyleSheetFamily 233
styleSheets 15
subdirFlagAt 158
subscriptCharacterHeight 165
subscriptCharacterWidth 166
subscriptOffset 165
superiorCharacterHeight 166
superiorCharacterWidth 166
superscriptCharacterHeight 165
superscriptCharacterWidth 165
superscriptOffset 165
supressEmptyParagraphs 186
supressLineNumbering 110, 141

T

tableHeight 65
TableLineSeparator 233
TablesAbbreviation 200
tablesAbbreviationEntry 102, 139, 141
TablesBibliography 200
tablesBibliographyEntry 102, 139, 141
TablesContents 200
tablesContentsEntry 102, 139, 141
TableSeparatorPosition 233
TablesIndex 201
tablesIndexEntry 101, 139, 141
TablesPicture(bool active, String alternativeText) . . . 201
tablesPictureEntry 102, 139, 141
TablesRunningTitle(bool active, String alternativeText, int level) 201
tablesRunningTitleEntry 102, 139, 141
tableWidth 65
tableWidthType 65
TabsType 233
tabulator 176
Tabulator 202
Tabulator (String position, TabulatorAlignment alignment, ...202
TabulatorAlignment 233
Tabulators 202
taggedPdfOptions 42
TaggedPdfOptions 203
TaggedPdfType 233
tagType 203
templateStyleSheet 137
testDirectoryFilter 215
testOption 215
text 65, 120, 174, 188, 207, 219, 220, 222, 224
textAlignment 105, 141
textChain 52, 83, 120
textChains 17
textChangedMethod 223
TextColumnLineAnchor 233
textCursor 16
textEditedMethod 223
TextFile 212
textLinesAutomaticSpacing 166
textLinesMethod 166
textOffset 203, 204
textPreferences 16
textRange 54
TextRange 119
thesaurusModuleIdent 162
title 21, 155
title, ...210
toggleColumn 17
toGroup() 82
tolerance 165
top 197
toPage 195
topAnchor 174
topBorder 193
topIndent 54, 73, 150, 174, 180, 181, 192, 193
topLevel 41
topMargin 154
topSeparator 150
topSeparatorData 150
topTextIndent 145
transparency 45, 146
Transparency 205
TransparencyType 233
type . .40, 130, 174, 176, 178, 182, 187, 198, 199, 202, 205, 206, 207
typeString 42
typographicQuotationMarks 166
TypographicQuotesMode 233

U

underline 95, 138, 140
underlineLineWidth 165
underlineOffsetHorizontalLayout 165
underlineOffsetVerticalLayout 165
UnderlineOptions 206
ungroup() 78
uniqueName 128
unzip 212
updateCustomPropertis 20
updateView() 20
useBleed 195
userProperty 20, 23, 29, 42
userScriptsPath 9
userXMLProperty 20, 24, 29, 42
useSubfolders 191

V

valid 74, 174
value 132, 172, 182, 219, 220, 222, 223
var myObjGroup 4
var myRgb 4
var obj = doc.objects().create(ObjectType.Text); 125
var variableName1 4
var variableName2 4
var variableName3 4
var variableName4 4
variable 89
Variable 207
VariableType 233
verticalAlignment 34, 73, 150, 195
VerticalAlignment 233
verticalCorrection 182

VerticalGuideZone 233
verticalMirrored 59
verticalOffset 58
verticalPasteOffset 163
verticalScale 58, 144
verticalSeparator 149
verticalSeparatorData 149
viewMode 215
viewZoomFactor 163
visible 41, 76
visualizeStyleSheets 163, 166

W

warning (. 210
warningForBwColorMode 165
warningForEffects 165
warningForGrayColorMode 165
warningForRgbColorMode 164
warningForTransparency 165
WhichObjects 233
WhichSpreads 233
widowsOrphans 111, 142
WidowsOrphans 207
WidowsOrphansMode 233
width 23, 49, 58, 71, 126, 145, 178, 220, 223
widthAt 184
widthType 71
WidthType 234
withFrame 34
workingTime 158
writeLine 212
writingDirection 108, 141
WritingDirection 234

Y

yellow 132

Z

zip 191, 211, 212