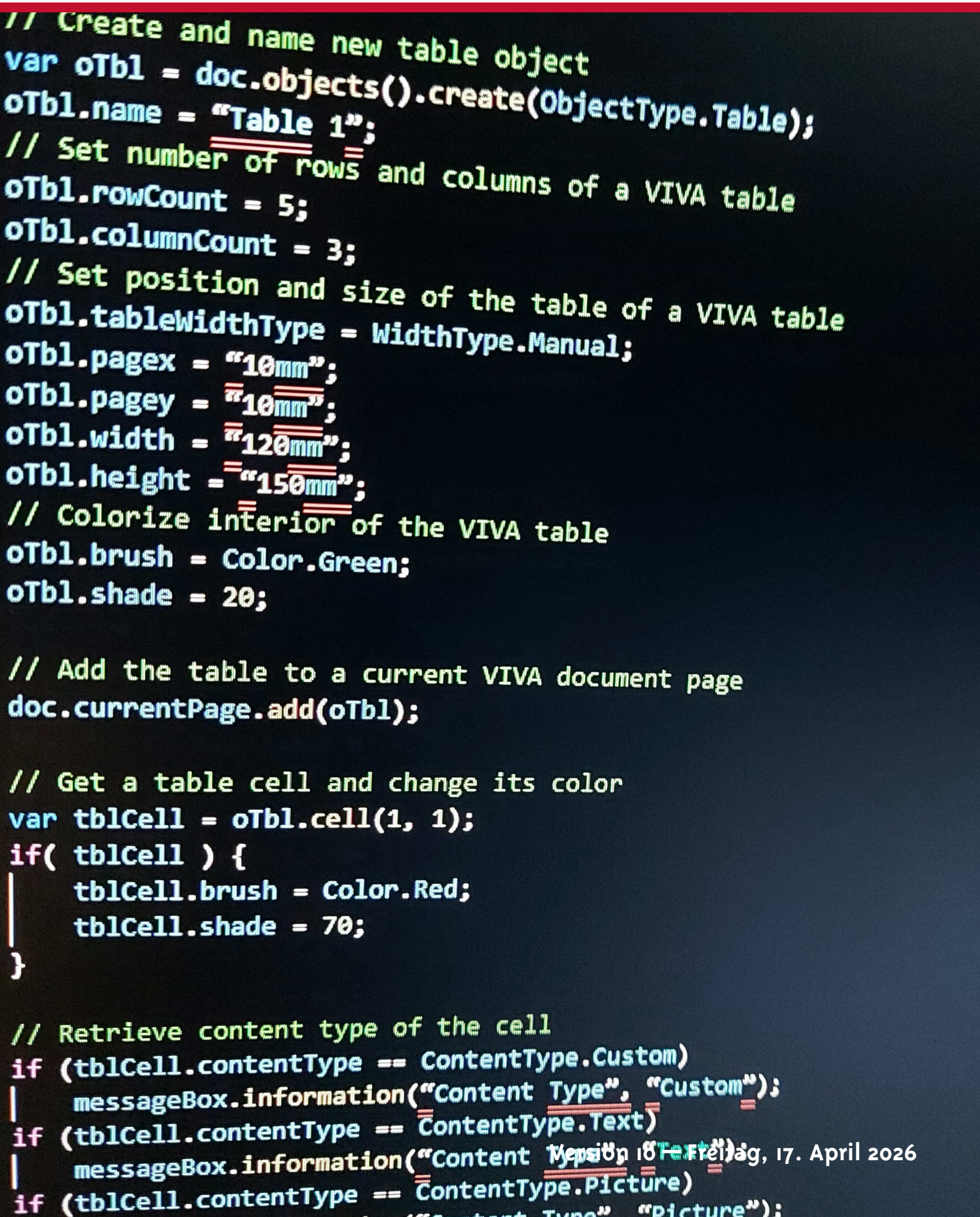
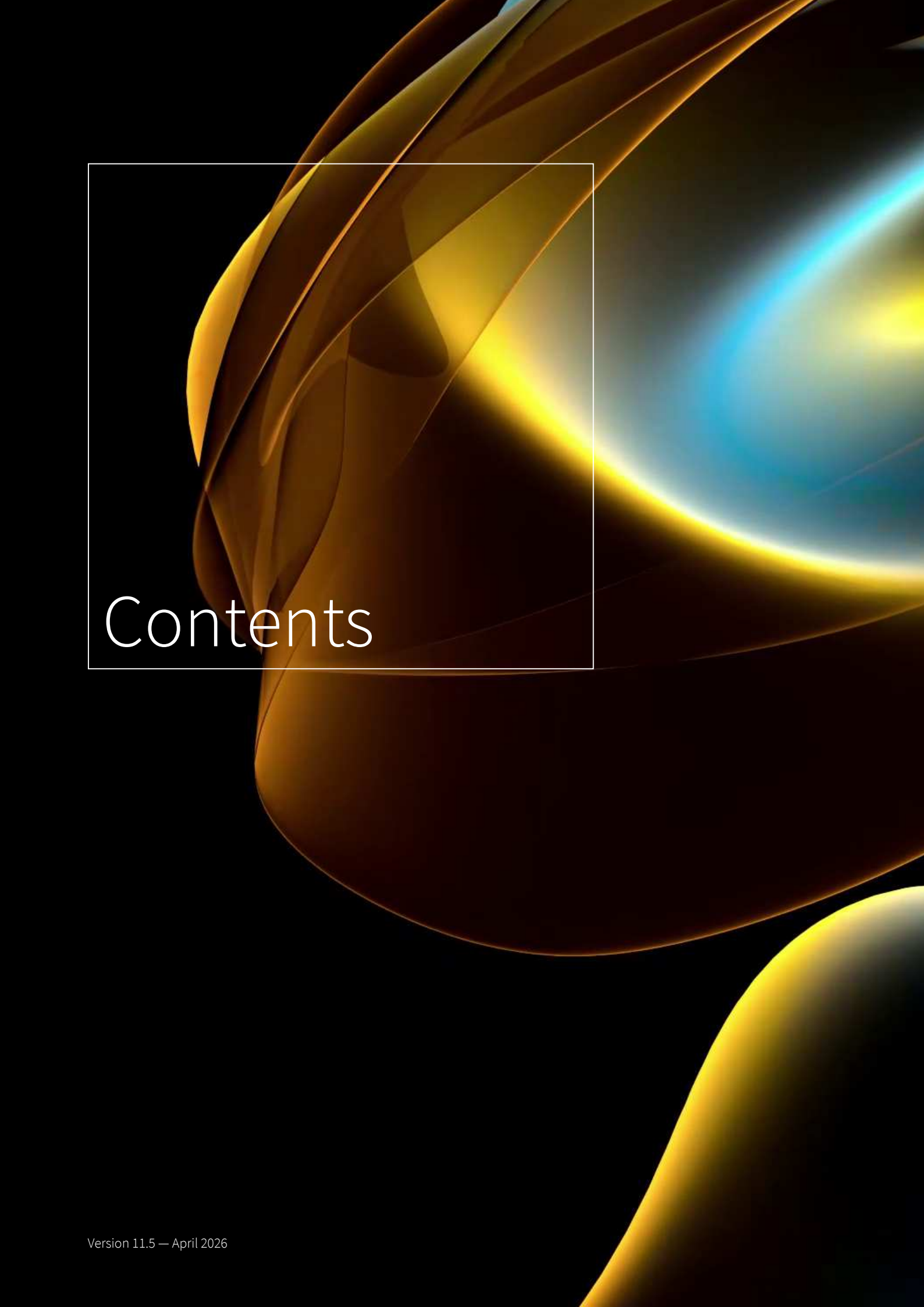
VivaDesigner 11 (Build 9224)

JSkripter 1.5

WE PUBLISH THE WORLD





Contents

Version 11.5 — April 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....7
Command: execute....7
Command: convertUnitValue....8

Application, Document Structure Objects

Main object “application”....10
Class: Document....11
Class: Page....16
Class: DocumentPage....18
Class: AliasPage....19
Class: Spread....19
Class: AliasSpread....21
Class: Layer....22

Object Types and Structure

Class: Objects....25
Class: Object....27

Class: GraphicObject....30
Class: LineObject....31
Class: FrameObject....33
Class: TextObject....36
Class: TextContent....38
Class: PictureObject....40
Class: PictureContent....42
Class: TextChains....44
Class: TextChain....44
Class: TableObject....45
Class: TableRow....47
Class: TableColumn....48
Class: TableCell....50
Class: TableSeparator....53
Class: GroupObject....55
Class: ObjectAlias....56
Class: TextObjectAlias....57

FormattedText Objects

Class: FormattedText....60
Class: TextRanges....90
Class: TextRange....90
Class: TextCursor....91

Object shapes

Class: Point....94
Class: PointList....95
Class: BoundingBox....97
Class: Matrix....97

Brushes and Colors

Introduction....100

Class: Brushes....100
Class: Brush....100
Class: ColorBrush....101
Class: BlendBrush....101
Class: Stop....103
Class: Rgb....103
Class: Cmyk....104
Class: Hsv....104
Class: Lab....104
Examples....105

StyleSheets

Class: StyleSheets....109
Class: StyleSheet....110
Class: CharacterStyleSheet....111
Class: ParagraphStyleSheet....112
Class: LayoutStyleSheet....115
Class: PictureStyleSheet....116
Class: GraphicStyleSheet....116
Class: TableStyleSheet....118
Class: RowColumnStyle....119
Class: RowColumnStyleAlt....120
Class: TableRowStyleSheet....120
Class: TableColumnStyleSheet....121
Class: TableCellStyleSheet....121

Document and Page Settings

Class: DocumentSettings....124
Class: PageSettings....125
Class: ChapterInfo....126
Class: DocumentStatistics....126
Class: SeachPaths....127

Preferences

Class: EmbeddedFontsPreferences....132
Class: LanguageInfo....132
Class: LanguagePreferences....132
Class: LayoutPreferences....133
Class: ObjectPreferences....133
Class: PagePreferences....134
Class: PreflightPreferences....134
Class: TextPreferences....135

Helped Data Objects

Class: Annotation....141
Class: Attributes....142
Class: AutomaticCharacterSpacing....142
Class: AutomaticCharacterWidth....142
Class: AutomaticWordSpacing....142
Class: BaselineGrid....143
Class: BaselinePosition....143
Class: ColorOverlay....144
Class: ColumnLines....144
Class: Comment....145
Class: DropCaps....145
Class: Enumerations....146
Class: FontSize....148
Class: GraphicFootnotes....148
Class: HeaderFooter....149
Class: Hyphenation....149
Class: ChangePictureColorSpaceSettings....150
Class: CharacterBackground....150
Class: CharacterFrame....151
Class: CharacterKerning....152
Class: CharacterPositionCorrection....152

Class: CharacterRule....153
Class: Image....154
Class: InnerGlow....154
Class: LayoutColumnsInfo....155
Class: LayoutFootnotes....155
Class: LineStyle....156
Class: LineNumbers....156
Class: Link....157
Class: Note....158
Class: OpenType....158
Class: OptimizePictureSettings....159
Class: OuterGlow....160
Class: OutlineOptions....160
Class: PackageSettings....161
Class: ParagraphBackground....161
Class: ParagraphFrame....162
Class: ParagraphRule....164
Class: PDFExportSettings....164
Class: Reflection....166
Class: Runaround....166
Class: Shadow....167
Class: SpellLang....168
Class: StrikethroughOptions....168
Class: TablesAbbreviation....169
Class: TablesBibliography....169
Class: TablesContents....169
Class: TablesIndex....170
Class: TablesPicture....170
Class: TablesRunningTitle....171
Class: Tabulator....171
Class: Tabulators....171
Class: TextEndnotes....172

Class: TextFootnotes....172
Class: TransparencyStop....173
Class: Transparency....173
Class: UnderlineOptions....174
Class: Variable....175
Class: WidowsOrphans....175

User Interface Objects

Class: MessageBox....178
Class: Directory....178
Class: File....179
Class: TextFile....180
Class: FileDialog....182
Class: ProgressReporter....183
Class: Dialog....183
Class: Label....185
Class: CheckBox....185
Class: ComboBox....186
Class: GroupBox....187
Class: RadioButton....187
Class: LineEdit....188
Class: TextEdit....189
Class: Button....189

Enumeration Objects Index

Index

Introduction

Version 11.5 — April 2026

Introduction

JScripter is a VivaDesigner tool that allows you to manipulate documents using a simplified version of the JavaScript programming language. 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

Scripting Features

Version 11.5 — April 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(Object.Group);
```

The parameter of the function *create()* is the name of the class of the created object.

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"
```

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 a string type from a defined list of values that are created from substrings: *Solid*, *Dash*, *Dot*, *Line*: *SolidLine*, *DashLine*, *DotLine*, *DashDotLine*, *DashDotDotLine*

Or the options: *'Pattern 0'* to '*Pattern 17'*.

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:

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:

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 attribut 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 accesible becase the file system is protected.

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 without its path

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 runs an external program. The first parameter is a program path, second and next are program parameters.

Using:

```
execute (String programPath, String arg1, String arg2, ...)
```

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 preferences *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 = he unit will be added to the result after value. Optional, the default value is false.

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:",
application.convertUnitValue(ds.columnDistance, "m", true, "pt"));

repaint();
quit();

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

Application, Document Structure Objects

Version 11.5 — April 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*

language *String* *Read*

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)* *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.

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

Examples:

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

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 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.

documentSp*read*Count *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.

aliasSp*read*Count *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 on object of the class PreflightPreferences. See the chapter “Class: PreflightPreferences” for more information about preflight preferences.

languagePreferences	*LanguagePreferences*	*Read/Write*

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

objectPreferences	*ObjectPreferences*	*Read/Write*
layoutPreferences	*LayoutPreferences*	*Read/Write*
pagePreferences	*PagePreferences*	*Read/Write*
embeddedFontsPreferences	*EmbeddedFontsPreferences*	*Read/Write*
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	*UnitValueString*	*Read*
pageHeight	*UnitValueString*	*Read*
documentSettings	*DocumentSettings*	*Read/Write*
toggleColumn	*int*	*Read/Write*

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*

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

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

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)* *void*

Moves the specified layer to a position which is given by the index parameter.

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)* *bool*

exportToPDF*(PDFExportSettings settings)* *bool*

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

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*) *String*

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

updateView() *void*

Updates the view of all documents.

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: 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*
layoutPositionColumn	*int*	*Read/Write*
guideCount	*int*	*Read*

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*

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

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)* *String*

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

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

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

addHorizontalGuide(String position, Layer layer) *int*

Returns ID of the guide

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

Returns ID of the guide

removeGuide(int id) *bool*

clearGuides() *void*

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, remove pages or move them to another position in the document.

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.

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();
}
```

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 spread.

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.

Methods:

add*(Object obj)* *void*

Add an object to the spread.

obj: Object which is added to the spread.

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*

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

pageAt*(int index)* *DocumentPage*

Returns a *DocumentPage* object at the position which is identified by the index.

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)* *DocumentPage*

Returns a *DocumentPage* object 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) String*

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

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() *void*

Removes the Alias spread (including nested Alias pages) from the appropriate document.

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*
brush	*Brush*	*Write*

Sets a color for the virtual frames of all objects (represented in the program interface via the option "Show Guides"). This allows you to distinguish objects on different layers.

colorRgb	*Rgb*	*Read/Write*

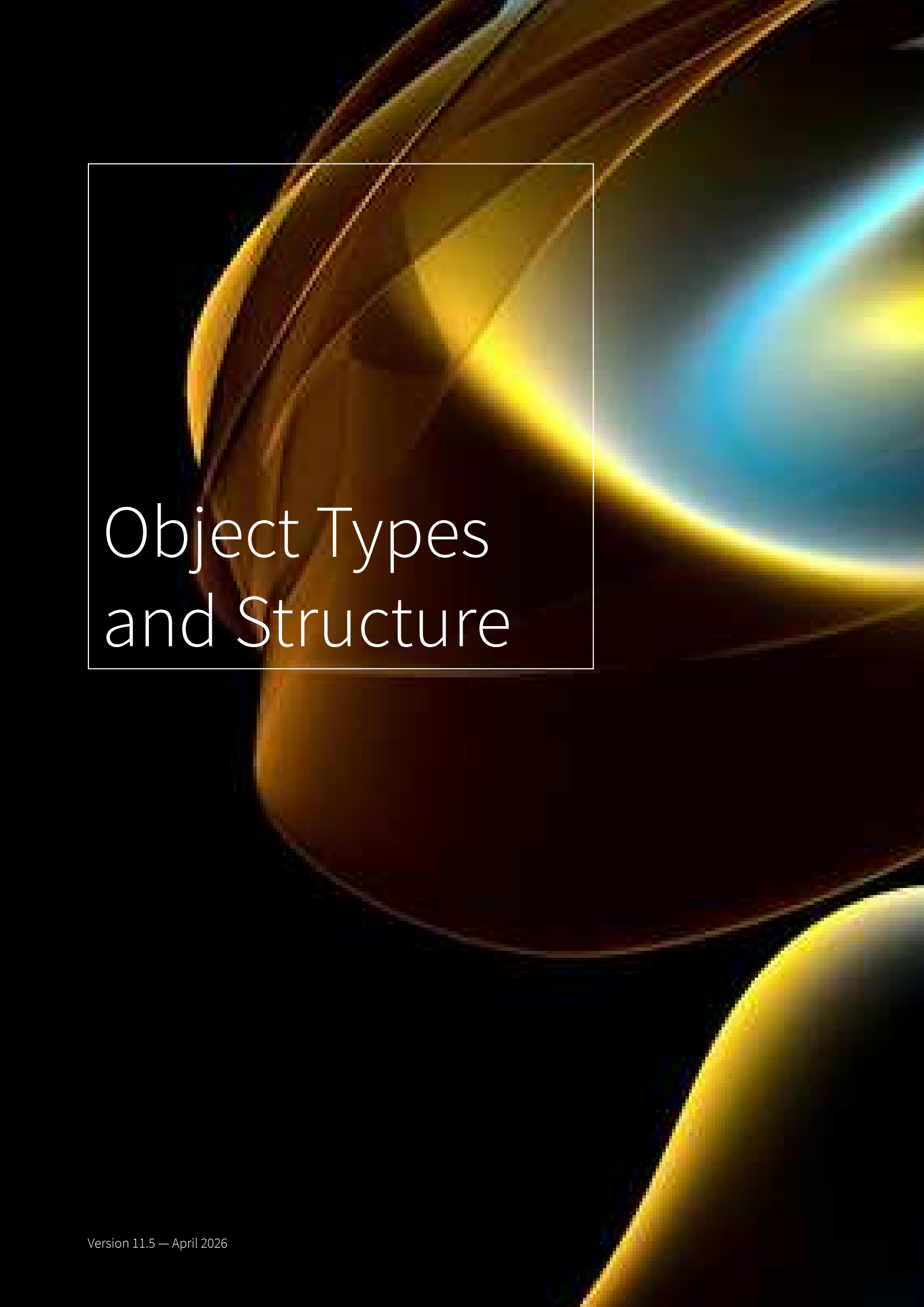
Set/get the color of a layer.

Methods:

objects(ObjectType objectType = All) objectType: See the enumeration ObjectType Returns a list of objects which are assigned to the layer.	*Objects*
objectCount*()*	*int*
getObject*(int index)*	*Object*
remove() Removes the layer from the appropriate document.	*void*

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.brush = Color.Blue;
  lay1.printable = false; // Don't print background layer.

  // Add the 2nd layer and configure it
  var lay2 = doc.addLayer();
  lay2.name = "Layer 1";
  lay2.brush = Color.Red;

  // 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.brush = Color.Green;

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



Object Types and Structure

Version 11.5 — April 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*** and ***GroupObject***. 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
|--- GraphicObject
   |--- LineObject
   |--- FrameObject
      |--- TextObject
      |--- PictureObject
      |--- TableObject
```

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 size and position cannot be changed.

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	*Object*	*Read*

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

runaround *Runaround* *Read/Write*

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

layer *Layer* *Read/Write*

Returns the layer object where the object is placed.

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.

remove() *bool*

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

createAlias() *AliasObject*

Creates the alias object for the current object.

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.

customProperty	*Boolean(String ident)*	*String*

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).

changeOrder *(ChangeOrderType type) bool*

Changes the order of object on the page.

moveOnPageTo *(Point p, MoveReferencePoint ref = BBoxTopLeft, Page pg = null)*	*bool*
moveOnSpreadTo *(Point p, MoveReferencePoint ref = BBoxTopLeft)*	*bool*
move *(Point point)*	*bool*
getAttributes *()*	*Attributes*
setAttributes *(Attributes attrs)*	*void*
copyAttributes *(Object fromObj)*	*boolean*

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*

Sefines 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*
reflection	*Reflection*	*Read/Write*
transparency	*Transparency*	*Read/Write*
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***.

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 = [[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 = [[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 text object width.

height *UnitValueString* *Read/Write*

Defines the text 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 *float* *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
      pageGeometryBounds = [50.15, 201.65, 200, 300];

      // Rounded corners (2.0 mm)
      cornerRadius = 2.0;

      // 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
      lineColor = brushBlue;
      lineWidth = "3pt";
      lineStyle = "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%
        baseOpacity = 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”. In addition to the property geometryBounds from the FrameObject class, this class also adds the further properties x, y, width and height to define position and size.

Properties:

content	*TextContent*	*Read*

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

individualAliasContent	*bool*	*Read/Write*
contentOpacity	*float*	*Read/Write*
textChain	*TextChain*	*Read*

The property returns the textchain of texts.

Methods:

connectTo *(TextObject object)*

Connect the current object to the story (chain)

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";
```

```
  // Position and size can also be set using the property
pageGeometryBounds
  //   pageGeometryBounds = ["25mm", "45.5pt", "80mm", "120mm"];
  // Create red border, yellow interior and set opacity
       lineWidth = "1.5mm";
       lineColor = Color.Red;
       fillColor = Color.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;
    columnDistance = "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(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.

columnDistance	*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*

The property sets the columnLines parameters.

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*
baselineGrid	*BaselineGrid*	*Read/Write*
header	*HeaderFooter*	*Read/Write*
footer	*HeaderFooter*	*Read/Write*

footnotes	*GraphicFootnotes*	*Read/Write*
headerFormattedText	*FormattedText*	*Read*
footerFormattedText	*FormattedText*	*Read*

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)* *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.

setHeader *(bool state, String value)*	*void*
setFooter *(bool state, String value)*	*void*
getAttributes *()*	*Attributes*
setAttributes *(Attributes attrs)*	*void*
copyAttributes *(Object fromObj)*	*boolean*

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;
      pageGeometryBounds = ["15mm", "35.5pt", "180mm", "120mm"];
      // Create red border, yellow interior and set opacity
      lineWidth = "1mm";
      lineColor = Color.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;
    columnDistance = "8mm";
    // Configure the vertical line separators between the columns
    columnLineColor = Color.Green;
    columnLineShade = 50;
    columnLineWidth = "1.5mm";
    columnLineTopIndent = "4mm";
    columnLineBottomIndent = "4mm";
    columnLineTopAnchor = TextColumnLineAnchor.Baseline;
    columnLineBottomAnchor = TextColumnLineAnchor.ObjectFrame;
    columnLineStyle = "Pattern 8";
    showColumnLines = true;

    // 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(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". In addition to the property geometryBounds from the FrameObject class, this class also adds the further properties x, y, width and height to define position and size.

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*

Returns flags about Alias 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

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
      pageGeometryBounds = ["35mm", "220.2pt", "130mm", "85mm"];
      // Rotate the object by 15 degrees
      rotation = 15.0;

      // Import a picture to the picture object
      content.importPicture("c:\\picture.pdf");
      // Read or manipulate the picture using property "content"
      with (content) {
        opacity = 80;
        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. (points per inch)		
resolutionY	*float*	*Read*
Get the Y resolution of the imported image in ppi. (points 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 imageimage 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*
Switches on clipping tightness for the imported image.		
clippingTolerance	*int*	*Read*

Set/get the clipping tolerance of the imported image.

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*
isEmbedded	*bool*	*Read*

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.

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, PictureVerticalPosition ver)* *void*

Set horizontal and vertical parameters for moving the image.

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

Set parameters ***type*** and ***center*** flag for moving the image.

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*

Class: TextChains

The class holds list of TextChain objects.

Properties:
length *int* *Read*

Methods:
at *(int index)* *TextChain*

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(0);
   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) TextObject*
Returns the text object from its index position in the text chain.

areaIndex *(TextObject area)* *int*

Returns an index of the TextObject 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”. In addition to the property ***geometryBounds*** from the ***FrameObject*** class, this class adds the properties x, y, width and height to define the position and size.

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.

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.brush = Color.Green;
    oTbl.shade = 20;

    // 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 = Color.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");
}
```

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 pf the table row is filled by a color blend instead of a solid color.

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

copyAttributes *(Object fromObj) boolean*

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*

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 colo.

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 colspanSpan = 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*

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 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`.

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*

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;
    lineWidth = 8;
    pagex = "0mm";
```

```
    pagey = "0mm";
    width = "30mm";
    height = "20mm";
    lineColor = Color.Blue;
    fillColor = Color.Yellow;
    lineShade = 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 ***Table***, 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*, *float lineWidth)*	*void*

This method implements the formatting of the cell separator in a complex way.

brush: Defines the 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 in millimeters (mm).

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

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 = Color.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(Color.Blue, 80, 50, "Pattern 1", 5);
}
```

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*

getObject(int index) *Object*

ungroup() *void*

Release all objects that are grouped in this group object.

insertAfter*(Object obj, VJObject afterObj)* *void*

Insert a new object in the group object.

obj: Newly inserted object.

afterObj: The object after which the new object is inserted.

insertAt*(Object obj, int pos)* *void*

Insert a new object in the group object.

obj: Newly inserted object.

pos: Position where the new object is inserted.

add*(Object obj)* *void*

Add a new object to the group object. The new object is appended to the end of the list of previously grouped objects.

obj: Newly inserted object.

remove() *bool*

Removes group object.

Examples:

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

  // Create several objects
  var obj1 = objs.create(ObjectType.Text);
  obj1.name = "Obj1";
  obj1.pageGeometryBounds = [20, 20, 100, 100];

  var obj2 = objs.create(ObjectType.Graphic);
  obj2.name = "Obj2";
  obj2.pageGeometryBounds = [60, 60, 150, 150];

  var obj3 = objs.create(ObjectType.Text);
  obj3.name = "Obj3";
  obj3.pageGeometryBounds = [210, 210, 50, 50];

  // 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: 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*

Methods:

`remove()`	*bool*

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*
`textChain`	*TextChain*	*Read*

FormattedText Objects

Version 11.5 — April 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 the method ***TextContent.formattedText()*** to access the text content of the text object.

Properties:

length	*int*	*Read*

The propety 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.

quotesMode: typographic quotes mode

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.

Returns a TextRange object which is related to the newly inserted text.

charAsString*(int pos=0)*	*String*
charAsCode *(int pos=0)*	*int*

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. If the parameter pos is higher that text length minus 1, the method returns value 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.

---

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.

---

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*

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 first occurrence of the text 'text' in the formatted text from a position 'from'. Returns TextRange object.

findRegEx*(int from, String pattern, bool ignoreCase=true)*	*TextRange*

Find first occurrence of the regular expresion 'pattern' in the formatted text from a position 'from'. Returns TextRange object.

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*

Replace all occurrences of the regular expresion 'pattern' by the string 'text'.

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.

Methods (character setting)

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 name of the language.

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: Name of selected language.

begin: Index of first letter of affected text fragment. The default value is 0.

end: Index of theposition 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 name of the 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 theposition 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*

Apply a new brush (color) to a specified text fragment.

brush: ColorBrush applied to the text fragment.

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*(qfloat 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(int pos=0)	*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.

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)*	*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(*int pos=0*) *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*
getSupressLineNumbering()	*bool*
getSupressLineNumbering(*int pos*) *bool*	

getSupressLineNumbering*(int begin, int end)*	*bool*
getSupressLineNumbering*(TextRange range)* *bool*	
setSupressLineNumbering*(bool on, TextRange range)*	*void*
setSupressLineNumbering*(bool on=true, int begin=0, int end=len)*	*void*
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*(int pos=0)* *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*
layoutLineDirection *(int pos=0)*	*LayoutLineDirection*
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=0)*	*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*(int pos = 0)*	*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 pos, 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*(int pos=0)*	*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=end)*	*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*(TextRange range)* *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*

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("20", trLine1.begin, trLine1.end - 20);
        setBrush(Color.Blue, trLine1.begin + 20, trLine1.end - 20);
        // Automatic underline: no style parameters are necessary
        setAutomaticUnderline(FontStyleLineRangeType.Words, trLine2.begin,
                trLine2.begin + 11);
        // Manual strikethrough: prepare parameters and apply them to a
text
        var optSt = new StrikethroughOptions();
        with (optSt) {
            brush = Color.Red;
            shade = 100;
            opacity = 60;
            lineWidth = "1.5pt";
            lineStyle = "Pattern 2";
            offset = "-3pt";
            rangeType = FontStyleLineRangeType.Words;
        }
        setManualStrikethrough(optSt, trLine2.begin + 12, trLine2.begin +
32);
        // Glyphs
        var trGl = appendPlainText("1 x 1012 ... No Glyph Superscript
Subscript\n");
        appendPlainText("\n");
        setFontSize("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("35", trOl);
        var optOl = new OutlineOptions();
        with (optOl) {
            brush = Color.Green;
            shade = 100;
            opacity = 100;
            lineWidth = "2pt";
            lineStyle = "Solid";
            joinStyle = CharacterOutlineJoinStyle.MiterJoin;
        }
        setOutlineOptions(optOl, new TextRange(trOl.begin, trOl.begin +
7));

        optOl.brush = Color.Blue;
        optOl.joinStyle = CharacterOutlineJoinStyle.BevelJoin;
    optOl.active = true;
        setOutlineOptions(optOl, new TextRange(trOl.begin + 10, trOl.begin
+ 17));

        optOl.brush = Color.Magenta;
        optOl.joinStyle = CharacterOutlineJoinStyle.RoundJoin;


        // 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("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("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("15", trSpc);

        var charFrame1 = new CharacterFrame(Color.Green, 100, 100, "-2mm",
"-2mm",            "-7pt", "-7pt", "2pt", "Pattern 8");
        setCharacterFrame(true, charFrame1, trFrm.end - 37, trFrm.end - 22)

        var charFrame2 = new CharacterFrame();
        charFrame2.brush = 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 = "Pattern 5";
        setCharacterFrame(true, 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");
        setParagraphAlignment(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");
        setParagraphAlignment(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");
        setParagraphAlignment(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"];
                lineColor = Color.Red;
                lineWidth = "2pt";
                fillColor = 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*	*Read*

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 an object with currently edited text with the cursor active.

Properties:

text	*FormattedText*	*Read*

Returns objects of the type FormattedText that hold all text.

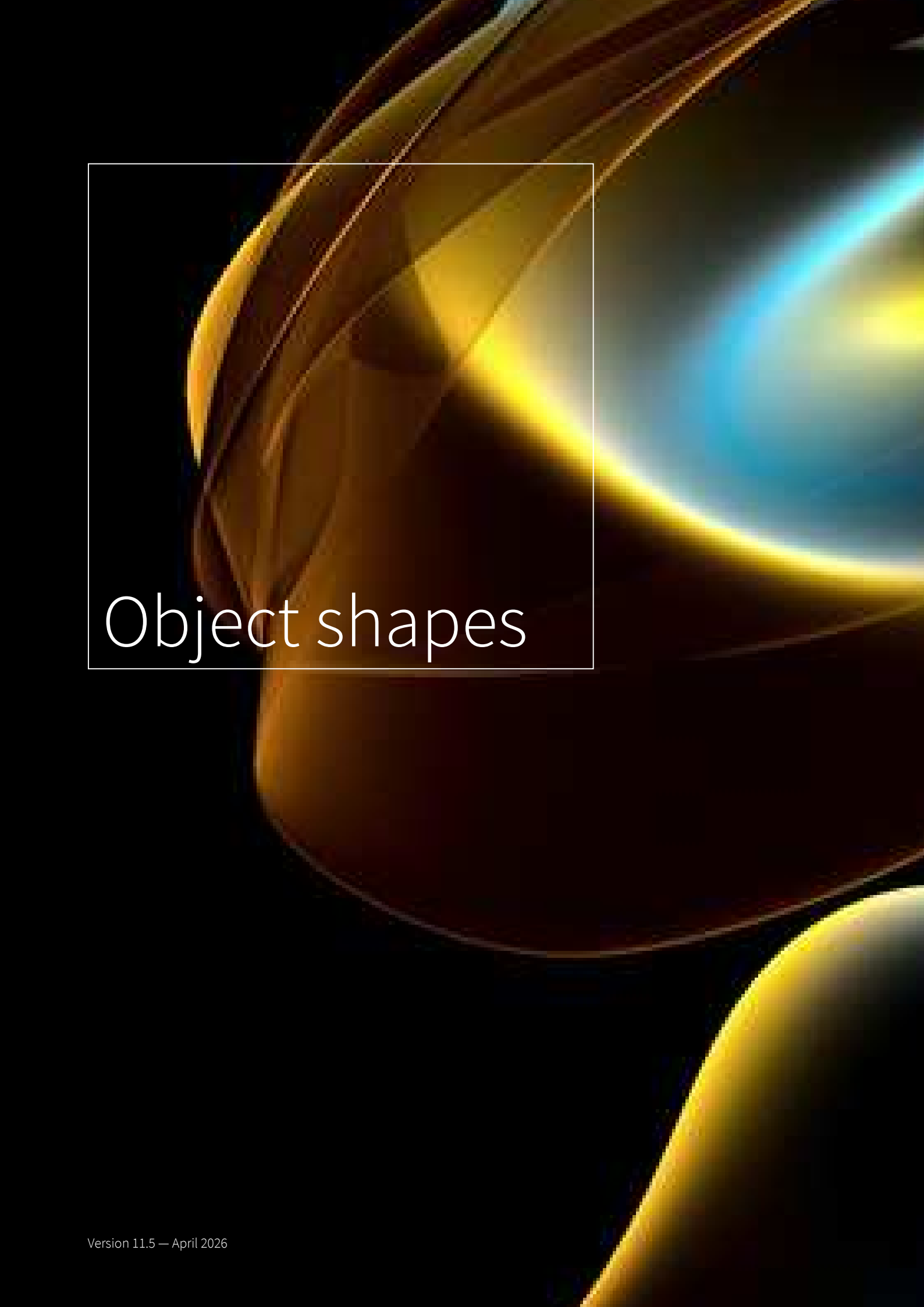
selection	*TextRange*	*Read/Write*

Set/get the range of text that is selected.

textChain	*TextChain*	*Read*

Returns the story of the text object that holds current text.

selections	*TextRanges*	*Read*



Object shapes

Version 11.5 — April 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

"***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

List of usable flags for the sizing point:

"***F***" - first point of the polygon point list; it must be only be defined when further polygons are defined in the polygon point list

"***L***" - last point of the polygon point list; it must only be defined when more polygons is defined in the polygon point list.

Constructor:

new `Point`***(x,y)***
new `Point`***(x,y,flags)***

Properties:

`x`	*UnitValueString*	*Read/Write*
Set/get x coordinate of the point.		
`y`	*UnitValueString*	*Read/Write*
Set/get y coordinate of the point.		
`flags`	*short*	*Read/Write*
Set/get flag information. Each flag is represented by one bit in the word.		
`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.

Examples:

```
var point = [335.56, 370.928, "F"];
var point = [304.554, 643.618];
var point = new Point(["15,3cm", "462pt", "BC"]);
```

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 more polygon point lists, the flags “first” and “last” sizing point must be used.

The point coordina tes are related to the page.

Constructor:

new `PointList()`
new `PointList(point list constant)`

Properties:

length	*int*	*R*

Returns the number of points in the polygon point list.

isGroup	*bool*	*R*

Returns true when point list contains more subpolygons, otherwise it returns false.

numberOfSizingPoints	*int*	*R*

Returns the number of sizing points.

numberOfSubPolygons	*int*	*R*

Returns the number of subpolygons in the polygon points list.

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(Object.TextFrame);
   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:

```
BoundingBox(UnitValueString x, UnitValueString y,
     UnitValueString width,  UnitValueString height)
```

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, float dx, float 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*

Brushes and Colors

Version 11.5 — April 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 ***brushes()*** method 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, the exception is caused.

addBlend*(name)*	*BlendBrush*

Adds a brush with “name” to the list of brushes and returns BlendBrush. If “name” is not unique, the exception is caused.

at*(index)*	*Brush*

Returns ***ColorBrush*** or ***BlendBrush*** from the index position in the repository.

find*(name)*	*Brush*

Returns the ***ColorBrush*** or ***BlendBrush*** found 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/Write*

Get/Set the localized name of brush object.

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.

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 *short* *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*

Inserts the Stop object on the index position in 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)

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*)*Properties*:

cyan	*int (0-100)*	*Read/Write*

Get/Set a cyan part of the color model.

magenta	*int (0-100)*	*Read/Write*

Get/Set a magenta part of the color model.

yellow	*int (0-100)*	*Read/Write*

Get/Set a yellow part of the color model.

key	*int (0-100)*	*Read/Write*

Get/Set a key for 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));
  }
}
```

StyleSheets

Version 11.5 — April 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 *?????* .

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 *(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 (*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);

    doc.styleSheets.at(1, StyleSheetFamily.Character).remove();
    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);

 doc.styleSheets.find("TemplateChar").remove();
}
```

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*
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*

Methods:

setCharacterFrame*(bool on, CharacterFrame chf = 0)*	*void*
setCharacterBackground*(bool on, CharacterBackground chb = 0)*	*void*
setCharacterRule*(bool on, CharacterRule chr = 0)*	*void*

removeProperty*(String propertyName)*

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', 'characterFrame', 'characterRule', 'characterBackground', 'tablesIndex', 'tablesContents', 'tablesRunningTitle', 'tablesBibliography', 'tablesPicture', 'tablesAbbreviation'

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 *float* *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*
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*
outlineOptions	*OutlineOptions*	*Read/Write*
characterFrame	*CharacterFrame*	*Read/Write*
characterBackground	*CharacterBackground*	*Read/Write*
characterRule	*CharacterRule*	*Read/Write*
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*
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*
paragraphAlignment	*ParagraphAlignment (enum)*	*Read/Write*
paragraphLineSpacing Set word “auto” to set automatically.	*UnitValueString*	*Read/Write*
spaceToPreviousParagraph	*UnitValueString*	*Read/Write*
spaceToNextParagraph	*UnitValueString*	*Read/Write*
widowsOrphans	*WidowsOrphans*	*Read*
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*

Methods:

setAutomaticUnderline *(FontStyleLineRangeType rangeType = All)*	*void*
setManualUnderline *(UnderlineOptions options)*	*void*
resetUnderline() *void*	

setAutomaticStrikethrough *(FontStyleLineRangeType type = All)*	*void*
setManualStrikethrough *(StrikethroughOptions options)*	*void*
resetStrikethrough() *void*	
setCharacterFrame *(bool on, CharacterFrame chf)*	*void*
setCharacterBackground *(bool on, CharacterBackground chb)*	*void*
setCharacterRule *(bool on, CharacterRule chr)*	*void*
setAutomaticCharacterSpacing *(bool on, float min, float opt, float max)*	*void*
setAutomaticCharacterWidth *(float min, float opt, float max)*	*void*
setAutomaticWordSpacing *(float min, float opt, float max)*	*void*
setWidowsAndOrphans *(WidowsAndOrphans mode, int startLines, int endLines)*	*void*

removeProperty *(String propertyName)*

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', 'characterFrame', 'characterRule', 'characterBackground', 'tablesIndex', 'tablesContents', 'tablesRunningTitle', 'tablesBibliography', 'tablesPicture', 'tablesAbbreviation', 'writingDirection', 'opticalAlignment', 'supressLineNumbering', 'paragraphFrame', 'paragraphBackground', 'paragraphRuleAbove', 'paragraphRuleBelow', 'automaticCharacterSpacing', 'automaticCharacterWidth', 'automaticWordSpacing', 'paragraphLeftIndent', 'paragraphRightIndent', 'paragraphIndent', 'enumerations', 'dropCaps', 'hyphenation', 'textAlignment', 'widowsAndOrphans', 'keepParagraphTogether', 'baselineGrid', 'paragraphLineSpacing', 'spaceToPreviousParagraph', 'spaceToNextParagraph', 'paragraphTabs'

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:

removeProperty *(String propertyName)*
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'

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*

Methods:
removeProperty *(String propertyName)*
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

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*
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*
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*

Method:

setRunaround *(RunaroundMode mode, Runaround runaround = null)*

setHeader *(bool state, String value)*

setFooter *(bool state, String value)*

removeProperty *(String propertyName)*

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'

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)*

removeProperty *(String propertyName)*

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 sFirstSyleSheetName,
   int lastCount, String sLastSyleSheetName,
   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)*	
removeProperty *(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)*	
`removeProperty` *(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:

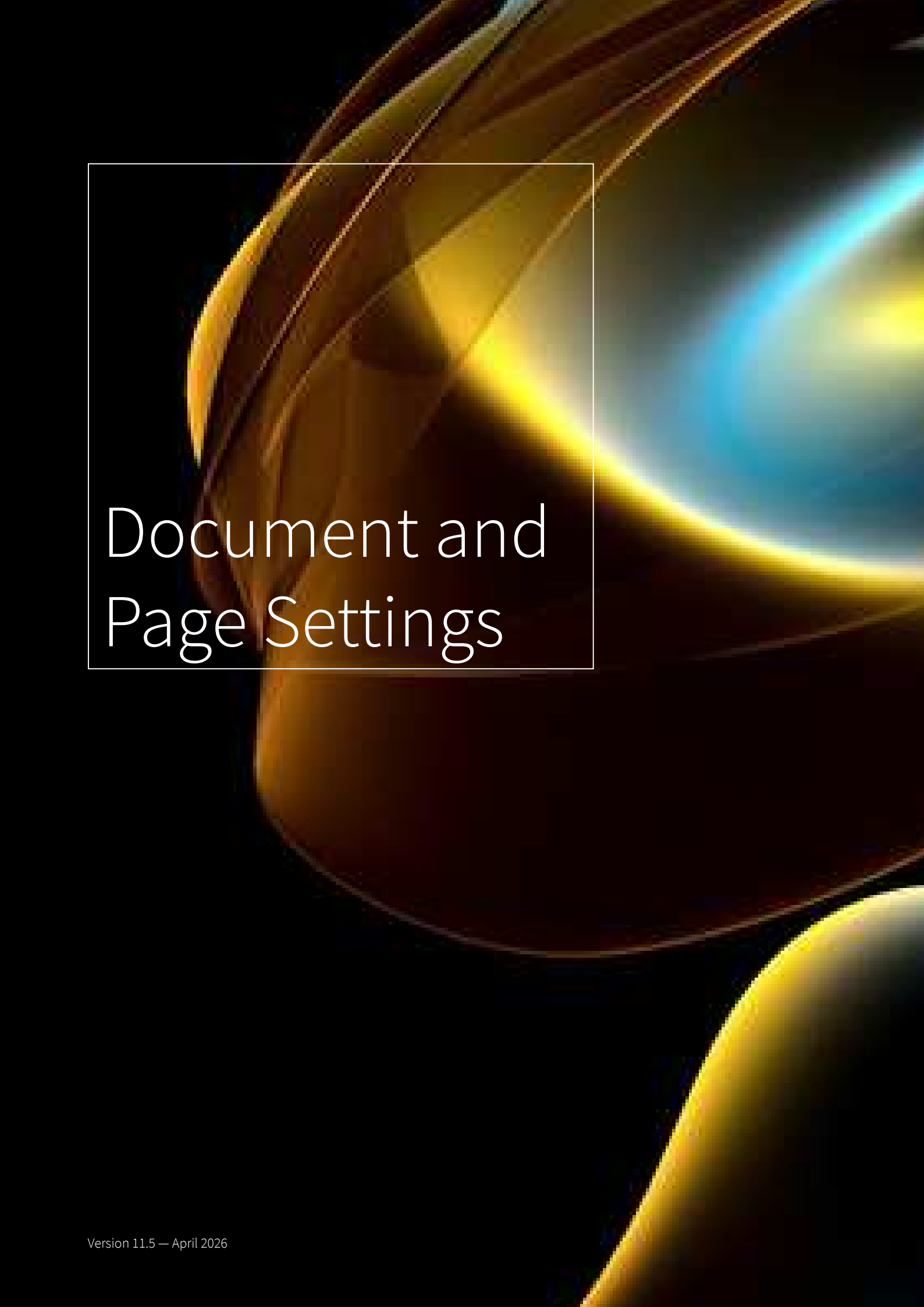
`fillBrush`	*Brush*	*Read/Write*
`fillShade`	*float*	*Read/Write*
`fillBlendAngle`	*float*	*Read/Write*
`horizontalAlignment`	*HorizontalAlignment (enum)*	*Read/Write*
`verticalAlignment`	*VerticalAlignment (enum)*	*Read/Write*
`rotation`	*ContentRotation (enum)*	*Read/Write*
`leftIndent`	*UnitValueString*	*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*
removeProperty *(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 11.5 — April 2026

Document and Page Settings

Class: DocumentSettings

An object holds the document parameters that are used when a new document is created.

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. The property is used for PageType = FacingPage only.

outerMargin	*UnitValueString*	*Read/Write*

Get/Set an outer margin for the new document page. The property is used for PageType = FacingPage only.

leftMargin *UnitValueString* *Read/Write*

Get/Set a left margin for the new document page. The property is used for PageType = SinglePage only.

rightMargin *UnitValueString* *Read/Write*

Get/Set a right margin for the new document page. The property is used for PageType = SinglePage only.

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.

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));
```

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 PageInsertationPosition type in this chapter.

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*.

Constructor:
new **ChapterInfo()**
new **ChapterInfo**(*PageNumberingFormat pnk, short pageNumber, String prefix)*

Properties:

numberingFormat	*PageNumberingFormat*	*Read/Write*
newPageNumber	*int*	*Read/Write*
prefix	*String*	*Read/Write*

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: SeachPaths

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()*** and the document parameters are set using following properties:

length	*int*	*Read*

Methods:

addPath *(String path, bool withSubdir)* The char '\' must be doubled.	*void*
removePath *(String path, bool withSubdir)* The char '\' must be doubled.	*void*
removePath *(String path)* The char '\' must be doubled.	*void*

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 11.5 — April 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*
hyphenationModuleIdent	*int*	*Read*
spellModuleIdent	*int*	*Read*
thesaurusModuleIdent	*int*	*Read*

Class: LanguagePreferences

The class is used in the Document class.

Methods:

languageInfo*(Language lang)*	*LanguageInfo*
setLanguageInfo*(LanguageInfo info)*	*void*

Examples:

```
var doc = application.currentDocument;
if (doc) {
    var pref = doc.languagePreferences;
    var langInfo = pref.languageInfo(Language.German);
    messageBox.information("MSG",
     "SpelModuleIdent: " + 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*
`automaticSpellCheck`	*bool*	*Read/Write*
`automaticGrammarCheck`	*bool*	*Read/Write*
`quickTextView`	*bool*	*Read/Write*
`quickPictureView`	*bool*	*Read/Write*
`viewZoomFactor`	*float*	*Read/Write*

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 TextPrefernces()*** 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*
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 11.5 — April 2026

Helped Data Objects

Class: Annotation

The object of the class ***Annotation*** represents a property of the class ***FormattedText.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.annotationAt(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 **AutomaticCharacterSpacing()**

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 **AutomaticCharacterSpacing()**

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	*BaselineGridNumbe ring*	*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.		

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:

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/Write*
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*

Methods:

```
setBullet(BulletCharacters type, Char customChar)   void
```

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.styleSheet = "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.on = 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.on);
  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 parameters of the character rule function for the text.

The class is used in classes ***FormattedText***, ***CharacterStyleSheet*** and ***ParagraphStyleSheet***.

Constructor:

new **Image()**
new **Image(String data)**

Properties:

data	String	*Read*
width	*int*	*Read*
height	*int*	*Read*
format	*ImageFormat*	*Read*

Methods:

setData(*String data*)	*void*

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	*LineNumberRestartType*	*Read/Write*
supressEmptyParagraphs	*bool*	*Read/Write*

Class: Link

The class is used in the class ***FormattedText***, ***CharacterStyleSheet*** and ***ParagraphStyleSheet***.

Constructor:

new **Link()**

new **Link** (*LineNumbersMode mode,*
String documentName,
LinkAction action,
int pageNumber)

Properties:

active *(obsosete)*	*bool*	*Read/Write*
documentName *(only if destination != None)*	*string*	*Read/Write*
linkAction *(only if destination = Page)*	*linkAction*	*Read*
pageNumber *(only if destination = Page)*	*int*	*Read*
destination	*LinkDestination*	*Read/Write*

Methods:

setAction *(LinkAction action, int pageNumber)*	*void*

only if destination = Page

obsolete

Class: Note

The class is used in a class `FormattedText`.

Properties:

`type`	NoteType	*Read/Write*
`text`	*String*	*Read/Write*
(Err: If text is set, the anchor is not shown in the note.)		
`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*

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

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	*String*	
fontEmbedding	*PrintFontEmbedding*	*Read/Write*
ignoreTrapping	*bool*	*Read/Write*
imageQuality	*PrintImageQuality*	*Read/Write*
opi	*PrintOPI*	*Read/Write*
pdfCompatibility	*PDFCompatibility*	*Read/Write*
pdfOptimize	*bool*	*Read/Write*

pdfCreateLayers	*bool*	*Read/Write*
pdfCompressText	*bool*	*Read/Write*
pdfWarningScale	*bool*	*Read/Write*
pdfWarningScaleBelow	*int*	*Read/Write*
pdfWarningScaleAbove	*int*	*Read/Write*
pdfWarningTextOverflow	*bool*	*Read/Write*
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(*UnitValue*String left, *UnitValue*String top,
*UnitValue*String right, *UnitValue*String bottom) void

Examples:

```
  var ra1 = new Runaround( RunaroundMode.Right,
RunaroundShapeType.ObjectCountours, "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

Constructor:

new **SpellLang**(*Language type*)

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*

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 andFormattedText

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*	

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*
numberingType	*ReferenceAnchorNumberingType*	*Read/Write*
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*
`numberingType`	*ReferenceAnchorNumberingType*	*Read/Write*
`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*
setStopAt	*(int index, TransparencyStop stop)*	*void*
removeStopAt	*(int index)*	*void*
appendStop	*(TransparencyStop stop)*	*void*

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 class is used in the class FormattedText.

Constructor:

new **Variable***(VariableType type)*
new **Variable***(String typeName)*

Properties:

type	*String*	*Read*

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*

User Interface Objects

Version 11.5 — April 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*
`isAbsolutre`	*bool*	*Read*

Methods:

`makeDir(String dir)`	*bool*
Creates the subdirectory `dir` in the current directory. Returns TRUE if the directory is created sucessfully.	
`removeDir(String dir)`	*bool*
Removes the subdirectory `dir` in the current directory. Returns TRUE if the directory is removed sucessfully.	
`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 ***olsDir*** in the current directory to new name newName. Returns TRUEif the directory is removed sucessfully.

getFileList(*String nameFilters, DirectoryFilter filters=DirectoryFilter::NoFilter, DirectorySortFlag sort = DirectorySortFlag::NoSort*) *String[]*

Returns list of file names from the directory.

getFileList(*DirectoryFilter filters = DirectoryFilter::NoFilter, DirectorySortFlag sort = DirectorySortFlag::NoSort*)

Returns list of file names from the directory.

zip (*String filePath = null*) *bool*

Returns TRUE if the zip operation on the directory was processed succesfully, 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*

Returns ***TRUE*** if the zip operation on the file was processed succesfully, 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*

Returns ***TRUE*** if the zip operation on the file was processed succesfully, 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*

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.

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: FileDialog

The class works with ***FileDialog*** for selecting the directory or the files.

Properties:

acceptMode	*FileDialogAcceptMode*	*Read/Write*
fileMode	*FileDialogFileMode*	*Read/Write*
viewMode	*FileDialogViewMode*	*Read/Write*
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 paramters that are set to the objects before opening. Returns the value true if the slection 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, QString dir, QString 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*
progressBarSubText	*String*	*Write*
progressBarPosition	*int*	*Read/Write*
progressBarLength	*int*	*Read*

Methods:

close()	*void*
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*

Methods:

~~**close()**~~	~~*void*~~
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*
addGroupBox*(String label)*	*GroupBox*
addGroupBox*(int id, String label)* The method adds a box for radio buttons.	*GroupBox*
addButton*(int id, String label, String clickedMethodName)*	*VJButton*
addAcceptButton*(int id, String label)*	*VJButton*
addRejectButton*(int id, String label)*	*VJButton*
addButtons*()*	*void*
addButtons*(QString acceptLabel, QString rejectLabel)* The method adds an accept button and a cancel button in the dialog.	*void*
exec*()* 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.	*int*
accept*()*	
done*(int resultCode)*	
reject*()*	

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.show();

messageBox.information("MSG", d.caption + ": Result = " + res + "\n" +
    "LABEL: " + label.value + "\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/Write*

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.currentIndexChangeMethod = "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.

Methods:

addItem*(String label, int value)*	*void*
addItem*(String label, int value, boolean selected)* The method adds an item into an end of a combo box list.	*void*
hide*()* The method hides a combo box object.	*void*
show*()* The method shows (makes visible) a combo box object and sets the item to the current position.	*void*
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 A callback method has one parameter: ID of a radio button.	*String*	*Read/Write*
isEnabled	*boolean*	*Read*
isHidden	*boolean*	*Read*

Methods:

addRadioButton*(int id, String label, int defaultValue)*	*RadioButton*
addRadioButton*(String label, int defaultValue)* The methods add a radiobutton item into a groupbox.	*RadioButton*
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()	

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*
returnPressedMethod	*String*	*Read/Write*
selectionChangedMethod	*String*	*Read/Write*

A previous five callback methods have one parameter: *ID*

textChangedMethod	*String*	*Read/Write*
textEditedMethod	*String*	*Read/Write*

The previous two callback methods have two parameters: *ID*,*TEXT*.

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*.		
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 11.5 — April 2026

Enumeration Objects Index

AutoLineSpacingMethod:	*AscentDescent*, *FontSize*
AttributeGroup:	*Character, Paragraph, Layout, Story, All*
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:	*Black*, *Blue*, *Bronze*, *Brown*, *Cyan*, *Gold*, *Green*, *Magenta*, *Orange*, *Pink*, *Red*, *Salmon*, *Silver*, *Turquoise*, *Violet*, *White*, *Yellow*
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:	*Text*, *Picture*, *Table*, *None*
DialogItemType:	*Unknown*, *LineEdit*, *TextEdit*, *CheckBox*, *Label*, *GroupBox*, *ComboBox*, *Button*
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.

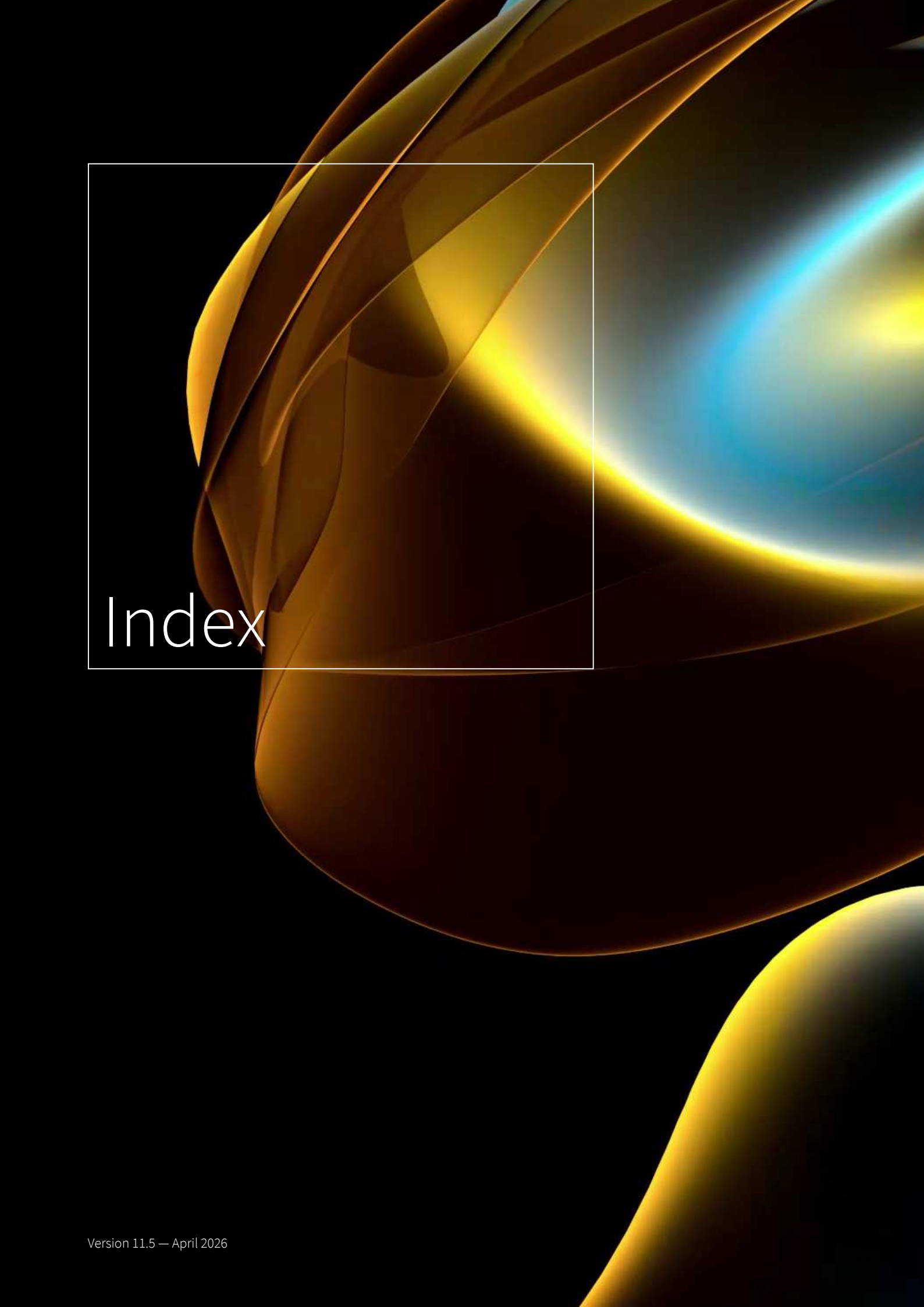
DirectorySortFlag:	*NoSort, Name, Time, Size, Unsorted, DirsFirst, Reversed, IgnoreCase, DirsLast, LocaleAware, Type*
DoubleQuotesType:	*InvalidDoubleQuotes*, *DoubleQuotes1*, *DoubleQuotes2*, *DoubleQuotes3, DoubleQuotes4*, *DoubleQuotes5*, *DoubleQuotes6*, *DoubleQuotes7*
DropCapsType:	*None*, *WholeWord*, *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*
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*, *IndentHere*, *EmSpace*, *EnSpace*, *ThreePerEmSpace*, *FourPerEmSpace*, *SixPerEmSpace*, *ThinSpace*, *HairSpace*, *PunctuationSpace*, *EmSpaceNoBreaking*, *EnSpaceNoBreaking*, *ThreePerEmSpaceNonBreaking*, *FourPerEmSpaceNonBreaking*, *SixPerEmSpaceNonBreaking*, *ThinSpaceNonBreaking*, *HairSpaceNonBreaking*, *PunctuationSpaceNonBreaking*, *SpaceNonBreaking*, *FixedSpaceNonBreaking*, *FigureSpace*, *EmDash*, *EnDash*, *Hyphen*, *DiscretionaryHyphen*, *EmDashNonBreaking*, *EnDashNonBreaking*, *HyphenNonBreaking*, *LineBreak*, *ParagraphBreak*, *LayoutColumnBreak*, *TextColumnBreak*, *TextObjectBreak*, *LayoutBreak*, *ChapterBreak*
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*
KerningType:	*Automatic*, *Manual*

Language:	*NoLang*, *Afrikaans_South_Africa*, *Arabic*, *Bulgarian*, *Catalan*, *Chichewa*, *Chinese*, *Croatian*, *Czech*, *Danish*, *Dutch*, *Dutch_Secondary*, *English*, *English_AU*, *English_CA*, *English_GB*, *English_NZ*, *English_US*,*English_UK_ise*, *English_UK_ize*, *Faroese*, *Finnish*, *French*, *French_BE*, *French_UppercaseUnaccented*, *Galician*, *German_AT*, *German_CH*, *German_CH_Reformed*, *German_DE*, *German_DE_Reformed*, *German_DE_Combined*, *Greek*, *Hebrew*, *Hungarian*, *Icelandic*, *Indonesian*, *Irish*, *Italian*, *Japanese*, *Kinyarwanda*, *Kiswahili*, *Korean*, *Kurdish*, *Lithuanian*, *Latvian*, *Malagasy*, *Malay*, *Maori*, *Norwegian*, *Norwegian_Nynorsk*, *Polish*, *Portuguese*, *Portuguese_Brazilian*, *Romanian*, *Russian*, *Russian_Je*, *Russian_Jo*, *Setswana*, *ScottishGaelic*, *Slovak*, *Slovenian*, *Spanish*, *Spanish_MX*, *Spanish_UppercaseAccented*, *Swedish*, *Tagalog*, *Tetum*, *Thai*, *Ukrainian*, *Vietnamese*, *Welsh*, *Zulu*, *Universal*
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* Example:1 2 3 4, A B C D, a b c d, I II III IV, i ii iii iv
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*
PictureFormat:	*TIFF*, *JPEG*, *PNG*
PictureHorizontalPosition:	*Ignore*, *Left*, *Center*, *Right*
PictureVerticalPosition:	*Ignore*, *Top*, *Center*, *Bottom*
PolygonPointFlag:	*After*, *Control*, *Before*, *Constraint*, *First*, *Last*
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:	*Left*, *Center*, *Right*
PrintVerticalAlignment:	*Top*, *Center*, *Bottom*
PrintFontEmbedding:	*None*, *AllButBase14*, *OnlyType1*, *All*
PrintImageQuality:	*None*, *LoRes*, *HiRes*
PrintOPI:	*None*, *TIFF*, *EPS*, *Both*, *Generic*

PrintNameChangePolicy:	*AppendPageNumber*, *ReplaceWithPageNumber*, *ReplaceWithPageName*
ProgressReporterType:	*NoItems*, *TextItem*, *TwoTextItems*
ReferenceAnchorNumberingType:	*ContinuousNumbering*, *ChapterWiseNumbering*
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*
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*
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:



Index

Version 11.5 — April 2026

Index

A

absolutePath 10
accept 184
acceptLabel 182
acceptMode 182
activatedMethod 186
active119, 120, 142, 143, 144, 145, 149, 150, 151, 153, 154, 156, 160, 162, 163, 164, 166, 167, 169, 170, 171, 173
active (. 157
add 16, 20, 55
addAcceptButton 184
addAliasPage 13
addBlend 100
addButton 184
addButtons 184
addCheckBox 183
addColor 100
addComboBox 183
addDocumentPage 13
addGroupBox 184
addItem 187
addLabel 183
addLayer() 14
addLineEdit 184
addOption 182
addPath 127
addRadioButton 187
addRejectButton 184
addTabulator 172
addTextEdit 184
afterPageNumber 125
aliasObjectsSelectable 133
aliasPage 18
aliasPageAt 13
aliasPageByName 13
aliasPageCount 11
aliasSp 11
aliasSpreadAt 14
alignCharacter 171
alignment 147, 171
alternatingColumns 119
alternatingColumnsData 119
alternatingRows 119
alternatingRowsData 119
alternativeText 169, 170, 171
anchorText 158
angle 167, 174
annotation 61
annotationTextRange 141
appendPlainText 60
appendPoint 96
appendStop 102, 174
applyAsIndent 172
applyElementOpacity 167
applyStandardTabs 171
applyValue 143
appScriptsPath 8
areaAt 44
areaCount 44
areaIndex 44
askForImages 133
at 26, 44, 90, 100, 109
AttributeGroup 192
Attributes() 142
author 127, 145
AutoLineSpacingMethod 192
automaticCharacterSpacing 78, 114
AutomaticCharacterSpacing() 142, 143
automaticCharacterWidth 79, 114
automaticGrammarCheck 133, 136
automaticSpellCheck 133, 136
automaticTextObject 124
automaticWordSpacing 78, 114

B

baselineDescenders 38, 117
baselineGrid 38, 117, 137
BaselineGrid 143
BaselineGrid() 143
BaselineGridDirection 192
BaselineGridNumbering 192
baselineGridType 77, 114
BaselineGridType 192
BaselineOffset 192
baselinePosition 38, 117
BaselinePosition 143
BaselineType 192
baseOpacity 117
baseTextRange 141
begin 90
bleed 160
bleedBottom 165
bleedLeft 165
bleedRight 165
bleedTop 165
blendAngle 47, 49, 50
BlendType 192
blue 103
blur 154, 160, 167
bottom 167
bottomAnchor 145
bottomBorder 163
bottomIndent 38, 50, 121, 145, 151, 152, 162, 163
bottomMargin 124
bottomSeparator 122
bottomSeparatorData 122
bottomTextIndent 117
brush 22, 48, 49, 50, 54, 66, 111, 113, 150, 151, 153, 161, 162, 163, 164, 168, 174
brushes 11
bullet 147
BulletCharacters 192
bwImageResolution 160
bwImagesMinimumResolution 134

C

caption 183
cell 46
changeColorSpace 43
changeDir 178
changeOrder 30
ChangeOrderType 194
ChangePictureColorSpaceSettings(); 150
changeTracking 137
chapterInfo 18
ChapterInfo 126
ChapterInfo() 126
chapterName 85
chapters 86
characterBackground 75, 112, 113
CharacterBackground 150
characterBaselineOrientation 72, 112, 113
CharacterBaselineOrientation 194
characterBaselineShift 71, 111, 113
characterCount 146
characterFrame 75, 112, 113
CharacterFrame 151
CharacterFrame() 151
characterFunction 63
CharacterFunction 194
characterHeightScale 70, 111, 113
characterKerning 71, 111, 113
CharacterKerning() 152
characterLigatures 73, 112, 113
CharacterOutlineJoinStyle 194
characterPosition 69, 111, 113
CharacterPosition 194
characterPosition(int begin, int end) 69
characterPosition(int pos) 69
characterPosition(int pos=0) 69
CharacterPositionCorrection 152
CharacterPositionCorrection() 152
characterRotation 72, 112, 113
CharacterRotation 194
characterRule 76, 112, 113
CharacterRule 153
CharacterRule() 153
characterSpacing 70, 71, 111, 113
characterStyleSheet 76, 112, 137, 147
characterStyleSheetByName 76, 77, 137, 147
characterUpperLowerCase 69, 111, 113
CharacterUpperLowerCase 194
characterWidthScale 70, 111, 113
charAsCode 60
charAsString 60
cleanupFolder 161
clear() 63, 110
clickedMethod 187, 188, 189
clippingEnabled 42
clippingTight 42
clippingTolerance 42
close 14, 180
close() 183
closeAndSave() 14
cmyk 101
color 144, 154, 160, 167
Color 192
colorBrush 103
colorDepth 133
ColorDepth 192
colorImageResolution 160
colorImagesMinimumResolution 134
colorMode 43
ColorMode 192
colorModel 101
ColorModel 192
colorOverlay 31, 118
ColorOverlay() 144
colorProfile 150
colorRgb 22
colorSeparation 101, 127, 165
ColorSeparation 192
ColorSeparationState 192
colorSpace 43, 150
ColorSpace
column 45
columnCount 38, 45, 117, 148, 155
columnDistance 38, 149
columnGutter 117, 125
columnLines 38, 117
ColumnLines 144
ColumnLines() 144
columns 115
columnSpan 50
comment 62
Comment() 145
connectTo 36

containsUserProperty 15, 17, 20, 29
content 36, 41, 57
contentExpandableObject 51
contentObject 51
contentOpacity 36, 41, 117
contentRotation 50
ContentRotation 192
contentType 51
ContentType 192
convertUnitValue 8
copy 179
copy() 29, 63
copyAsAlias 126
copyAttributes 30, 39, 44, 48, 49, 51, 54
copyObjectPreferences 15
copyObjects 125
copyOnlySelectedObjects 126
copyUserProperties 29
count 119, 155, 171, 178
Count 11
Courier Bold 2
coverShadow 167
create 25, 109
createAlias() 29
creationDate 145
crop 159
cropMarkDistance 165
cropMarkLength 165
cropmarks 165
currentDocument 10
currentIndexChangedMethod 186
currentLayer 12
currentPage 12
currentSpread 12
currentText 12
cursorPositionChangedMethod 188, 189
customHeight 124
customProperty 29, 30
customWidth 124
cyan 104

D

deadline 127
deleteObjects() 17
deletePoint 96
deleteSelectedObjects() 17
destination 157
Dialog 183
DialogItemType 192
direction 143
directory 182
Directory 178
directoryFilter 182
DirectoryFilter 192
DirectorySortFlag
disable 185, 186, 187, 189, 190
disableSystemFonts 132
displayPosition 172
distance 167
distanceBetweenStandardTabs 136
distanceToText 146
documentAt 10
documentCount 10
documentName 157
documentPageAt 13
documentPageCount 11
documentSettings 12
DocumentSettings 124
documentSp 11
documentSpreadAt 14
documentStatistics 12
DocumentStatistics 126
done 184
doubleQuotesType 132
DoubleQuotesType 193
dropCaps 79, 80, 114
DropCaps 145
DropCaps() 145
DropCapsLinesType 193
DropCapsRunaroundType 193
DropCapsType 193
dropShadow 30, 118
dx 97
dy 97

E

editingFinishedMethod 188
embedAllImages 133
embedAllUsedFonts 132
embeddedFontsPreferences 12
enable 185, 186, 187, 188, 189, 190
end 90
endnoteLabelStyleSheet 137
endnoteLabelStyleSheetByName 137
endnoteReferenceStyleSheet 137
endnoteReferenceStyleSheetByName 137
endnotes 137
EndnotesBreakType 193
EndnotesPositionType 193
endnotesRanges 86
endnoteTextStyleSheet 137
endnoteTextStyleSheetByName 137
enumerations 80, 114
Enumerations 147
Enumerations() 146
EnumerationsAlignment 193
EnumerationsNumberFormat 193
EnumerationsNumberingMode 193
EnumerationsType 193
eof 180
equals 45
exec 184
execute 7
exists 128, 178, 179
exportToPDF 14
exportToXML 14
exposure 127
ExposureType 193
extendToColumnBorder 162, 163
extendToNextParagraph 162, 163
ExtendToNextParagraphType 193

F

fax 127
File 179
FileDialogAcceptMode 193
FileDialogFileMode 193
FileDialogOption 193
FileDialogViewMode 193
fileMode 182
fileName 179
fileNameLabel 182
filePath 10, 12, 42
fileTypeLabel 182
fillBlendAngle 35, 117, 118, 120, 121
fillBrush 117, 118, 120, 121
fillCharacter 171
fillColor 34
fillOpacity 35, 117
fillShade 34, 117, 118, 120, 121
find 63, 100, 109
findAndReplace 64
findAndReplaceAll 64
findAndReplaceAllRegEx 64
findAndReplaceRegEx 64
findRegEx 64
firstBaseline 136, 143
firstCount 120
firstStyleSheet 120
firstStyleSheetByName 120
flagAfter 94
flagAfterConstraint 94
flagBefore 95
flagBeforeConstraint 95
flagFirst 95
flagLast 95
flags 94
focusPosition 103, 173
font 64, 111, 112
fontEmbedding 165
FontReferenceSizeType 193
fontSize 65, 111, 113
FontSize 148
fontsSubfolder 161
FontStyleLineRangeType 194
FontStyleLineType 194
footer 38, 117
footerFormattedText 39
footerRows 118
footerRowsData 119
footnoteLabelStyleSheet 137
footnoteLabelStyleSheetByName 137
footnoteReferenceStyleSheet 137
footnoteReferenceStyleSheetByName 137
footnotes 39, 115, 117, 137
FootnotesPosition 194
footnoteTextStyleSheet 137
footnoteTextStyleSheetByName 137
forceLineBreak 136
form 31, 34
format 150, 160
formattedText 38, 61
frameColor 34
FrameForm 194
frameOpacity 34
frameRadius 34
frameShade 34
frameStyle 34
frameWidth 34, 152, 163
fromPage 165
fullCopy() 29
fullPath 178

G

getAttributes 30, 39, 43, 48, 49, 51, 54
getExistingDirectory 182
getFileList 179
getObject 13, 17, 20, 23, 55
getOpenFileName 182
getPoint 95
getSaveFileName 182
getSaveFileName() 182
getSupressLineNumbering 80, 81
getTabulator 172
gotoPage 14
GraphicFootnotes() 148
graphicStyleSheet 133
grayImageResolution 160
grayImagesMinimumResolution 134
greekBelow 136
green 103
groupObject 28
guide 28
guideCount 16
guideObject 117
guideObjectsMagnetic 133
guideObjectsSelectable 133

H

halftoneAngle 43, 116
halftoneScreen 43, 116
header 38, 117
HeaderFooter() 149
headerFormattedText 39
headerRows 118

headerRowsData 118
height 34, 42, 47, 97, 116, 149
heightType 47
HeightType 194
hidden 22, 101
hide 185, 186, 187, 188, 189, 190
highlightedMethod 186
horizontalAlignment 50, 121, 165
HorizontalAlignment 194
horizontalCorrection 153
horizontalMirrored 42
horizontalOffset 42
horizontalPasteOffset 133
horizontalScale 42, 116
horizontalSeparator 120, 121
horizontalSeparatorAbove 45
horizontalSeparatorData 120, 121
hsv 101
Hsv 104
hue 104
hyphenation 80, 114
Hyphenation 149
Hyphenation() 149
hyphenationModuleIdent 132
HyphenationQuality 194

I

id 185, 186, 187, 188, 189
identifier 10
ignoreTrapping 165
Image() 154
Image(String data) 154
ImageFormat 194
imageQuality 165
imagesSubfolder 161
importPicture 43
include 7
includeDocument 161
includeFonts 161
includeImages 161
includeProtocol 161
indexKey 101
indexModel 101
IndexModel 194
individualAliasContent 36, 41, 117
information (. 178
innerGlow 30, 118
InnerGlow() 154
innerMargin 124
innerShadow 30, 118
insertAfter 55
insertAnnotation 62
insertAt 55
insertCharacter 61
insertComment 62
insertFormattedText 61
insertNote 62
insertPlainText 39, 60
insertPoint 96
insertPosition 125
insertVariable 63
isAbsolutre 178
isAliasPage 16
isEmbedded 43
isEmpty 43
isEnabled 185, 186, 187, 188, 189
isGroup 95
isHidden 185, 186, 187, 188, 189
isOnline 25
isReadable 178
isRelative 178
isRoot 178

J

joinStyle 161
justifiedSpaceBetweenLinesMaximum 83, 115

K

keepParagraphTogether 82, 114
keepRunaround 22
KerningType 194
key 104

L

lab 101
Lab 104
label 18
language 66, 111, 113, 132, 136
Language 195
languageByName 65
languageInfo 132
languagePreferences 12
layer 29
layerAt 14
layerCount 11
layersRange 165
layoutColumns 84
LayoutColumnsInfo() 155
LayoutColumnsType 195
layoutFootnotes 84
LayoutFootnotes() 155
LayoutFootnotesMode 195
layoutGrid 124
layoutLineDirection 83
LayoutLineDirection 195
layoutLineJustification 83
LayoutLineJustification 195
layoutLineOrientation 83
layoutPositionColumn 16
layoutPositionRow 16
layoutPreferences 12
layouts 86
layoutStyleSheet 84, 137
layoutStyleSheetByName 85, 137
left 167
leftBorder 163
leftColumns 119
leftColumnsData 119
leftIndent 38, 50, 121, 151, 152, 153, 162, 163, 164
leftMargin 125
leftSeparator 121
leftSeparatorData 121
leftTextIndent 117
length 25, 44, 55, 60, 90, 91, 95, 100, 109, 127
lengthType 164
level 147, 170, 171
lightness 104
lineAngle 31
lineBrush 118, 156
LineCap 195
lineColor 31, 117, 144
lineCornerRadius 117
LineEditType 195
lineEnd 32, 117
LineForm 195
lineJustification 115
lineLength 31
LineNumbers 157
LineNumbers() 156
lineNumbersColor 136
lineNumbersData 136
lineNumbersFont 136
lineNumbersFontSize 136
lineNumbersMode 136
LineNumbersMode 195
LineNumbersRestartType 195
lineOpacity 31, 117, 144
lineOrientation 115
lineShade 31, 117, 118, 144, 156
lineStart 32, 117
lineStyle . . 31, 117, 118, 145, 152, 153, 156, 161, 163, 164, 169, 174
LineStyle 156
LineStyle() 156
linesType 145
lineWidth . .31, 117, 118, 144, 153, 156, 161, 164, 168, 174
link 74, 75, 112, 114
Link 157
Link() 157
linkableWithOriginal 38, 117
linkAction 157
LinkAction 195
LinkDestination 195
linkToTextChain 126
localizedName 100
locked 22, 28
logicalNumber 18
lookInLabel 182

M

m11 97
m12 97
m21 97
m22 97
magenta 104
mainText 141, 158
makeDir 178
margin 155
marginAt 155
matrix 28
Matrix 97
Matrix() 97
maxHyphensInARow 149
maximum 142, 143
maximumLines 145
maximumPreviews 134
maximumWidth 183
message) 178
mimeTypeFilter 182
minimum 142, 143
minimumLines 145
minimumWidth 183
minPrefix 149
minSuffix 149
mirrored 28, 116
mirroredHorizontally 116
mirroredVertically 116
mode 144, 147, 154, 155, 157, 160, 166, 167
modificationDate 145
move 30
moveLayer 14
moveOnPageTo 30
moveOnSpreadTo 30
movePicture 43
MoveReferencePoint 195

N

name 19, 21, 22, 27, 110, 178
nameChangePolicy 165
nameFilter 182
new Cmyk 104
new PDFExportSettings() 164
new Rgb 103
new TextEndnotes 172
new TextFootnotes 173
newDocument 10
newDocument() 10
newName) 179
newPageNumber 126
note 28, 62
notesVisible 60
NoteType 195
numberFormat 147, 172, 173
numbering 143
numberingFormat 126

numberingType 172, 173
numberOfColumns 125
numberOfDigits 165
numberOfSizingPoints 95
numberOfSubPolygons 95

O

object 61
objectCount 13, 17, 20, 23, 55
objectOpacity 35
objectPage 20
objectPreferences 12
objects 13, 16, 20, 55
objects(ObjectType objectType = All) 23
objectType 55
ObjectType 195
offset 153, 157, 164, 166, 167, 169, 174
offsetOnSpreadX 16
offsetOnSpreadY 16
offsetX 116
offsetY 116
oldName, ... 179
on 157
opacity . . .67, 103, 111, 113, 144, 151, 153, 154, 160, 161, 162, 163, 164, 166, 167, 168, 173, 174
open 180, 182
openDocument 10
OpenFileMode 195
openType 74, 112, 114
OpenType() 158
OpenTypeNumerals 195
OpenTypePlacement 195
OpenTypePositionalForms 195
opi 165
opticalAlignment 80, 114
OpticalAlignment 195
optimizePicture 43
OptimizePictureSettings() 159
optimum 142, 143
orderedBy 127
orderNumber 127
outerGlow 30, 118
OuterGlow() 160
outerMargin 124
outFilePath 150, 160
outlineOptions 67, 112, 113
OutlineOptions 160
OutlineOptions() 160
outputPath 165
overwriteExistingFiles 161
owner 56

P

package 14
PackageSettings() 161
page 28
pageAt 20
pageBoundingBox 28
pageCount 20
pageGeometryPoints 32, 34
pageHeight 12
PageInsertPosition 196
pageMode 11
PageMode 196
pageNumber 157
pageNumbering 165
PageNumberingFormat 196
pageOrder 124
PageOrder 196
pageOrientation 124
PageOrientation 196
pagePreferences 12
pageSelection 165
PageSettings() 125
pageSize 124
PageSize 196
pagesPerJob 165
pagesRangeType 165
pageType 124
PageType 196
pageWidth 12
pagex 27
pagey 27
paragraphAlignment 114
ParagraphAlignment 196
paragraphAtCharacterPosition 86
paragraphBackground 81, 114
ParagraphBackground 162
ParagraphBackground() 161
paragraphFrame 81, 114
ParagraphFrame 162
ParagraphFrame() 162
paragraphIndent 79, 114
paragraphLeftIndent 79, 114
paragraphLineSpacing 78, 114
paragraphRightIndent 79, 114
ParagraphRule 164
ParagraphRule() 164
paragraphRuleAbove 81, 114
paragraphRuleBelow 81, 114
ParagraphRuleLengthType 196
paragraphs 86
paragraphSpaceMaximum 83, 115
paragraphStyleSheet 82, 137
paragraphStyleSheetByName 82, 137
paragraphTabs 79, 114
parentId 186
path 178
pathAt 128
pdfColorConversion 166
PDFColorConversion
pdfCompatibility 165
PDFCompatibility 196
pdfCompressionMethod 166
PDFCompressionMethod 196
pdfCompressionQuality 166
pdfCompressText 166
pdfCreateLayers 166
pdfOptimize 165
pdfTargetProfile 166
pdfWarningScale 166
pdfWarningScaleAbove 166
pdfWarningScaleBelow 166
pdfWarningTextOverflow 166
pdfxAllowEmbeddedICCProfiles 166
pdfxGrayProfile 166
pdfxOutputIntent 166
phone 127
physicalNumber 16
PictureFormat 196
PictureHorizontalPosition 196
pictureIndividualAliasContent 117
pictureStyleSheet 133
PictureVerticalPosition 196
placeMaximumColumnCount 156
placeStartColumn 156
plainText 39, 60
Point 94
PointList 95
PointList() 95
PolygonPointFlag 196
position 103, 141, 148, 158, 171, 173
positionCorrection 62
postfix 147, 150, 160, 172, 173
prefix 126, 147, 150, 160, 172, 173
preflightDocument 134
preflightPreferences 12
PreflightPreferences 134
PreflightPreferences() 134
previewQuality 134
printable 22, 27, 117
PrintColorSeparation 196
printDeviceType 127
PrintDeviceType 196
printEmptyPages 165
PrintFontEmbedding 196
printImage 127
PrintImageQuality 196
PrintImageType 196
PrintLayersRange 196
printMaterial 127
PrintMaterialType 196
PrintNameChangePolicy 197
PrintOPI 196
PrintPageNumbering 196
PrintPagesRangeType 196
printResolution 127
PrintResolutionType 196
printSeparately 165
printSpellCorrection 165
printSpreads 165
PrintVerticalAlignment 196
progressBarFinish 183
progressBarLength 183
progressBarPosition 183
progressBarReset 183
progressBarSubText 183
progressBarText 183
ProgressReporter 183
ProgressReporterType
protected 27

Q

quality 149
quickPictureView 133
quickTextView 133

R

range 141
rangeType 169, 175
readLine 180
red 103
ReferenceAnchorNumberingType 197
reflection 31, 118
Reflection() 166
refresh 26
registrationMarks 165
reject 184
rejectLabel 182
remove 179
remove() 18, 19, 21, 23, 29, 46, 55, 57, 101, 110
removeAll() 128
removeAllTabulators 172
removeContent() 51
removeDir 178
removeDocumentPages 14
removeOption 182
removePath 127
removeProperty . . .112, 115, 116, 118, 119, 120, 121, 122
removeRange 63
removeStopAt 102, 174
removeStyleSheets 64
removeTabulator 172
rename 179
renameDir 179
resetStrikethrough() 115
resetUnderline() 114
resolutionX 42
resolutionY 42
responsible 127
restartType 157
returnPressedMethod 188
rgb 101
right 167
rightBorder 163
rightColumns 119

rightColumnsData 119
rightIndent 38, 50, 121, 151, 152, 153, 162, 163, 164
rightMargin 125
rightSeparator 122
rightSeparatorData 122
rightTextIndent 117
rotation 28, 42, 116, 121
row 46
RowColumnStyle 119
RowColumnStyle() 119
RowColumnStyleAlt 120
rowCount 45
rowSpan 50
RubyPosition 197
runaround 29, 117
Runaround 166
Runaround() 166
RunaroundMode 197
RunaroundShapeType 197
runaroundType 146

S

saturation 104
save 14
saveLargePreview 134
saveSmallPreview 134
saveTo 172
scale 127, 146, 167
scalePicture 43
ScalePictureType 197
screenAngle 101
ScreenAngle 197
searchPaths 12
secondCount 120
secondStyleSheet 120
secondStyleSheetByName 120
selected 28
selectedFiles 182
selection 91
selectionChangedMethod 188, 189
selections 91
separator 46, 118
separatorHorizontalOffset 172, 173
separatorLineBrush 172, 173
separatorLineLength 172, 173
separatorLineStyle 172, 173
separatorLineWidth 172, 173
separatorVerticalOffset 172, 173
separatorVisible 172, 173
setAction 157
setAliasPage 18
setAlternatingColumns 119
setAlternatingRows 119
setAttributes 30, 39, 43, 48, 49, 51, 54
setAutomaticCharacterSpacing 78, 115
setAutomaticCharacterWidth 79, 115
setAutomaticStrikethrough 115
setAutomaticUnderline 114
setAutomaticWordSpacing 78, 115
setBaselineGridType 78
setBlock 167
setBottomSeparator 122
setBrush 66
setBullet 147
setChapterName 85
setCharacterBackground 75, 112, 115
setCharacterBaselineOrientation 72
setCharacterBaselineShift 72
setCharacterFrame 75, 112, 115
setCharacterFunction 63
setCharacterHeightScale 70
setCharacterKerning 71
setCharacterLigatures 73
setCharacterPosition 69
setCharacterRotation 73
setCharacterRule 76, 112, 115
setCharacterSpacing 71
setCharacterStyleSheet 76
setCharacterStyleSheetByName 77
setCharacterUpperLowerCase 69
setCharacterWidthScale 70
setClipping 43
setCount 155
setCustomProperty 30
setData 154
setDropCaps 80
setDynamicLines 146
setEnumerations 80
setFillStyle 46, 48, 49, 51
setFixLines 146
setFont 64
setFontSize 65
setFooter 39, 118
setFooterRows 119
setFrameStyle 54
setHeader 39, 118
setHeaderRows 119
setHorizontalSeparator 120, 121
setHyphenation 80
setIndents 51
setJustifiedSpaceBetweenLinesMaximum 83
setKeepParagraphTogether 82
setLanguage 66
setLanguageByName 65
setLanguageInfo 132
setLayoutColumns 84
setLayoutFootnotes 84
setLayoutLineJustification 83
setLayoutLineOrientation 83
setLayoutStyleSheet 84
setLayoutStyleSheetByName 85
setLeftColumns 119
setLeftSeparator 122
setLink 75
setManualStrikethrough 115
setManualUnderline 114
setMarginAt 155
setObject 61
setOpacity 67
setOpenType 74
setOpticalAlignment 80
setOutlineOptions 67, 68
setParagraphBackground 81
setParagraphFrame 81
setParagraphIndent 79
setParagraphLeftIndent 79
setParagraphLineSpacing 78
setParagraphRightIndent 79
setParagraphRuleAbove 81
setParagraphRuleBelow 81
setParagraphSpaceMaximum 83
setParagraphStyleSheet 82
setParagraphStyleSheetByName 82
setParagraphTabs 79
setPoint 95
setPositionCorrection 63
setQrCodeToImage 11
setRange 91
setRightColumns 119
setRightSeparator 122
setRunaround 118
setShade 67
setSpaceToNextLayout 84
setSpaceToNextParagraph 78
setSpaceToPreviousLayout 84
setSpaceToPreviousParagraph 78
setStopAt 102, 174
setStoryEndnotes 85
setStoryFootnotes 85, 86
setStoryLineCounterData 86
setStrikethrough 68
setSupressLineNumbering 81
setTablesAbbreviationEntry 74
setTablesBibliographyEntry 74
setTablesContentsEntry 73, 74
setTablesIndexEntry 73
setTablesPictureEntry 74
setTablesRunningTitleEntry 74
setTextAlignment 77
setTopSeparator 122
setUnderline 68
setUserProperty 15, 17, 20, 29
setUserXMLProperty 15, 17, 21, 29
setVerticalSeparator 120, 121
setWidowsAndOrphans 115
setWidowsOrphans 81, 82
setWidthAt 155
setWritingDirection 80
shade .47, 49, 50, 53, 66, 103, 111, 113, 144, 150, 151, 153, 154, 160, 161, 162, 163, 164, 167, 168, 174
Shadow() 167
shapeType 167
show 185, 186, 187, 188, 189, 190
showAliasObjects 133
showBaselineGrid 133
showGuideObjects 133
showGuides 133
showInvisibles 136
showRuler 133
showSmartGuides 133
showTextRuler 133, 136
singleQuotesType 132
SingleQuotesType 197
size 148, 166, 179
skew 28, 42, 116
skipFirstCount 120
skipLastCount 120
smallCapsCharacterHeight 136
smallCapsCharacterWidth 136
smallestWord 149
smallPreviewsMaximumSize 134
snapDistance 133
spaceToNextLayout 84, 115
spaceToNextParagraph 78, 114
spaceToPreviousLayout 83, 84, 115
spaceToPreviousParagraph 78, 114
spacing 143
spans 86
SpecialCharacters 197
SpellLang 168
spellModuleIdent 132
spread 16, 28
spreadBoundingBox 28
spreadGeometryPoints 32, 34
spreadx 27
spready 27
start 143
startColumn 149
startNumber 172, 173
startsWith 157
startWith 147
stateChangedMethod 186
step 157
Stop 103, 173
Stop(. 173
stopAt 102, 174
stopCount 102, 174
story 86
storyEndnotes 85
storyFootnotes 85
storyLineCounterData 86
strikethrough 68, 111, 113
strikethroughLineWidth 135
strikethroughOffsetHorizontalLayout 135
strikethroughOffsetVerticalLayout 135
StrikethroughOptions 168
String fillCharacter) 171

styleSheet 31, 43, 45, 48, 49, 51, 119
styleSheetByName 31, 43, 45, 48, 49, 51, 119
StyleSheetFamily 197
styleSheets 11
subdirFlagAt 128
subscriptCharacterHeight 135
subscriptCharacterWidth 136
subscriptOffset 135
superiorCharacterHeight 136
superiorCharacterWidth 136
superscriptCharacterHeight 135
superscriptCharacterWidth 135
superscriptOffset 135
supressEmptyParagraphs 157
supressLineNumbering 114

T

tableHeight 45
TableLineSeparator 197
TablesAbbreviation 169
tablesAbbreviationEntry 74, 112, 113
TablesBibliography 169
tablesBibliographyEntry 74, 112, 113
TablesContents 170
tablesContentsEntry 73, 112, 113
TableSeparatorPosition 197
TablesIndex 170
tablesIndexEntry 73, 112, 113
TablesPicture(bool active, String alternativeText) . . . 170
tablesPictureEntry 74, 112, 114
TablesRunningTitle(bool active, String alternativeText,
int level) 171
tablesRunningTitleEntry 74, 112, 114
tableWidth 45
tableWidthType 45
TabsType 197
tabulator 147
Tabulator 171
Tabulator (String position, TabulatorAlignment
alignment,. 171
TabulatorAlignment 197
Tabulators 171
templateStylesheet 110
testDirectoryFilter 182
testOption 182
text 44, 91, 145, 158, 185, 186, 187, 189
textAlignment 77
textChain 36, 57, 91
textChains 13
textChangedMethod 188, 189
TextColumnLineAnchor 197
textCursor 11
textEditedMethod 188
TextFile 180
textLinesAutomaticSpacing 136
textLinesMethod 136
textOffset 172, 173
textPreferences 12
textRange 38
TextRange 90
thesaurusModuleIdent 132
title,. 178
toggleColumn 12
tolerance 135
top 167
toPage 165
topAnchor 145
topBorder 163
topIndent 38, 50, 121, 145, 151, 152, 162, 163
topLevel 28
topMargin 124
topSeparator 121
topSeparatorData 121
topTextIndent 117
transparency 31, 118
Transparency 173
TransparencyType 197
type . .27, 102, 145, 147, 148, 152, 158, 168, 171, 173, 174,
175
typographicQuotationMarks 136
TypographicQuotesMode 197

U

underline 68, 111, 113
underlineLineWidth 135
underlineOffsetHorizontalLayout 135
underlineOffsetVerticalLayout 135
UnderlineOptions 174
ungroup() 55
uniqueName 100
unzip 180
updateCustomPropertis 15
updateView() 15
useBleed 165
userProperty 15, 17, 20, 29
userScriptsPath 8
userXMLProperty 15, 17, 20, 29
useSubfolders 161

V

value 104, 143, 152, 186, 188, 189
var myObjGroup 4
var myRgb 4
var variableName1 4
var variableName2 4
var variableName3 4
var variableName4 4
variable 63
Variable 175
VariableType 197
verticalAlignment 50, 121, 165
VerticalAlignment 197
verticalCorrection 153
VerticalGuideZone 197
verticalMirrored 42
verticalOffset 42
verticalPasteOffset 133
verticalScale 42, 116
verticalSeparator 120, 121
verticalSeparatorData 120, 121
viewMode 182
viewZoomFactor 133
visible 28, 53
visualizeStyleSheets 133, 136

W

warning (. 178
warningForBwColorMode 135
warningForEffects 135
warningForGrayColorMode 135
warningForRgbColorMode 134
warningForTransparency 135
WhichObjects 197
WhichSpreads 197
widowsOrphans 81, 114
WidowsOrphans 175
width 33, 42, 48, 97, 116, 149
widthAt 155
widthType 48
workingTime 127
writeLine 180
writingDirection 80, 114

Y

yellow 104

Z

zip 161, 179, 180