

OWrite v5.4

by Brainy Data Limited

About OWrite

Latest Changes

For a list of the latest changes please scroll to the end of this chapter or [click here](#).

Important note about version 4.1: Documents that are loaded with version 4.1 will be converted to the version 4.0 format and will no longer be compatible with OWrite versions prior to version 4.0. Please make backups of all your documents prior to using this release.

Introduction

OWrite is a cross-platform word processor component for Omnis Studio. It is fully integrated with our OSpell2 spell checker software and can be used with PDFDevice our latest PDF generation tool. All three products can be purchased as a single package. For detailed information about OSpell2 please refer to the separate documentation provided in the documentation folder.

The following is a brief description of the various parts of this software.

1. The OWrite, OSpell2 and PDFDevice external components contain the word processor, spell checker kernels and PDF printing engine. The OWriteW components implement the OWrite JavaScript client design components.
2. The OSpell2 library implements the entire interface that is required to enable spell checking in your desktop libraries. It is designed in such a way that it can be used as is, alongside your own libraries.
3. The example library and data file provide practical working examples of how PDFDevice, OSpell2 and OWrite can be used with your own library.

Installing the Software

There are a number of components to install and the component names vary between platforms. As a general rule, the downloaded folder containing the software will be organised so that you may follow these generic steps.

The downloaded folder contains folders for different versions of studio, i.e. *studio430*, *studio500*, *studio520*, *studio600*, *studio610*, *studio800* and *studio810*.

- Open the appropriate Omnis Studio folder. Always use the latest version that is not later in version than your version of Omnis Studio. For example, for Studio 5.0.2 you would use components from the *studio500* folder and **not** the *studio520* folder.

The Omnis Studio folder may contain two further folders called *JSCOMP* and *XCOMP*. These may or may not be post fixed with a three letter platform identifier, i.e. *_mac*, *_win* or *_lin*.

- Copy the components from inside JSCOMP and XCOMP to your Omnis-tree. You will find identical named folders inside the tree. On Mac OSX these can be found inside the *Omnis bundle->Content->MacOS*.
- For software that supports the Omnis Java script platform, there will be an additional folder named *html*. This folder will contain additional java scripts and modification to *jsctempl.htm* that require integration with your Omnis tree. Please refer to the accompanied *readme.txt* file

You can run the example library directly from the *OWritePlus* folder. If you move the example library from this folder, you must also move all other files and folders contained within.

Deploying your software

Please refer to the license agreement for rules on deployment.

Documentation

This documentation describes the functionality provided by OWrite only. It is recommended that you also read the OSPELL2 and PDFDevice documentation to gain a better understanding of the complete OWrite Plus product. The OSPELL2 documentation is currently only available as HTML files in the Documentation's folder of the OWrite Plus examples download.

As a minimum we recommend that you read the following Chapters:

OWrite: Introduction and Designing OWrite.

JS-OWrite: Introduction, Designing OWrite and Designing JS-OWrite.

OSPELL2: From the separate OSPELL2 documentation read [Welcome](#), [Introduction](#) and [Integrating OSPELL2](#).

PDFDevice: From the separate PDFDevice documentation read Welcome, Introduction and Designing PDFDevice.

Table of Contents

History

Below is a summary of the most recent enhancements.

Version	Enhancements
5.4.0	<p>The new property \$grammaroptions can be assigned a set of constants that define various grammar options. We may add to these options in future releases. The current supported options are:</p> <p>kWriGrammarCapFstWrđ: First word of a sentence is capitalized.</p> <p>kWriGrammarDelDbłSpace: double spaces are removed when deleting the current selection.</p>
5.3.0	<p>The following properties have been renamed to work-around tokenization issues in Studio 10.2 when oWrite uses properties that override internal notation names:</p> <p>\$::paper -> \$docpaper \$::paperlength -> \$docpaperlength \$::paperwidth -> \$docpaperwidth \$::orientation -> \$docorientation \$::topmargin -> \$doctopmargin \$::leftmargin -> \$docleftmargin \$::bottommargin -> \$docbottommargin \$::rightmargin -> \$docrightmargin \$::pagecount -> \$docpagecount \$::pagenumber -> \$docpagenumber \$::showrulers -> \$showpaperrulers \$::insert() -> \$docinsert() \$::print() -> \$docprint()</p> <p>The new property \$evalpermanent is a read only property that returns kTrue if a document is loaded that was previously saved using \$savedata with bMakeDataPermanent set to kTrue</p>
5.2.0	<p>oWrite picture objects with click calculations that specify an a valid URL beginning with 'http://' or 'https://' will now produce clickable links when printed to PDFDevice.</p> <p>Simple RTF returned from table cell calculations may now contain additional calculations that reference data in the list associated with the table using the calculation “\$ref.COLUMN_NAME”. Thus, one can now use oWrite itself to create simple templates and export them as RTF and insert this RTF in table cells via a cell’s calculation.</p>

5.1.0	<p>OWrite now supports the import of basic MSWord fields (i.e. date, time, page count and page number)</p> <p>The property \$maximagesize is now also applied when using \$docinsert().</p> <p>The new property \$pasteoptions controls what can be pasted into a oWrite desktop document from the clipboard.</p>
5.0.0	<p>New property \$curlistlevel for the support of sub-lists as well as new editing behaviour when editing lists and sub-lists.</p> <p>New saving mode kWriOutputHF to force the output of a single header at the beginning of the document and a single footer at the end of the document for non-paginated output formats such as plain text and HTML.</p> <p>Major improvements to \$papercontinuous. Static tables will now generate headers and footers around page breaks, display of main header and footer when in continuous paper mode, and manual page breaks no longer cause structural breaks in single page mode.</p> <p>New \$headfootnoedit property which disables the editing of headers and footers if set to true.</p> <p>New property \$docbulletchar. This property can be used to specify a different unicode character for the bullet list. Note: not all browsers support custom bullet characters.</p> <p>New property \$curobjevalmaxheight which specifies the maximum height for a table cell or text box when the content is evaluated. A positive value indicates centimetres or inches and a negative value the number of lines of text.</p> <p>The standard Omnis properties \$bordercolor and \$linesyle are now supported by the oWrite desktop control.</p>
4.5.0	<p>New JS-OWrite find and replace support. Please view technical note TN0031 for a full description of this new feature.</p> <p>Implemented \$curobjalign for table cells for specifying the vertical align of table cells.</p> <p>There are a number of new info object types. Please see constants kWriObjTypeInfo...</p> <p>New custom parameter kWriLoadMetaQuality for loading RTF data.</p>
4.3.0	<p>Change to \$::insert. The third parameter now takes one of the constants kWriInsertSelect or kWriInsertKeepStyles.</p> <p>Click calculations can now be plain text if they begin with "http://" or "https://". When these objects are clicked, OWrite will generate an evObjClick event instead of attempting to execute it as Omnis notation.</p>

4.1.0	<p>Major changes to the way OWrite measures text. You must read case 1641 release notes before using this software.</p> <p>OWrite now supports dictation on Macintosh.</p> <p>New method \$getselpageoffset for retrieving offset of current selection.</p> <p>Method \$getfontlist has new parameter for retrieving font heights for JS-Client. See release notes for case 1641.</p> <p>New warning kWriWarnBadPagenateData which is generated during loading when a JS-Client document contains only partial pagination data.</p>
4.0.0	<p>Implements JS-OWrite. Please see chapter Designing JS-OWrite.</p>
3.8.6	<p>New border-less printing option. See \$::print.</p>
3.8.5	<p>New property \$watermarks for providing a list of watermarks that are to be added to the specified document pages during printing.</p> <p>New property \$docdecimaltabchar for changing the character that is used for positioning decimal tab positions.</p>
3.8.0	<p>New property \$evalkeeplf. If true it enables the importing of line feeds via plain text calculation results.</p> <p>New property \$nouserscroll. If true prevents users from scrolling OWrite content.</p>
3.6.5	<p>New property \$checkoverflow. If true OWrite prevents the user from entering more content than can be displayed in the field.</p>
3.6.0	<p>RTF Import</p> <ul style="list-style-type: none"> - added support for importing headers and footers - added support for importing page and date fields - table inside headers or footers are converted to tabs <p>Paragraph lists</p> <ul style="list-style-type: none"> - auto indent calculation changed to (1.5cm * font size / 12). Result is snapped to 0.25 cm. - Roman numerals are now right justified at first line indent position. - Using backspace to remove paragraph list now changes document style to the default style if the current document style specifies the paragraph list. <p>Calculated fields</p> <ul style="list-style-type: none"> - empty results will now be displayed using unicode character 200B which is a zero width space character (unicode OWrite only).

3.5.0	<p>New progress feature</p> <ul style="list-style-type: none"> - New property \$showprogress. if true, progress events (evProgress) will be generated, for NV objects \$progress method is called. - New event evProgress event type constants kWriProgress.... <p>Document Merging Optimisation</p> <ul style="list-style-type: none"> - New merge options kWriMerge... for \$mergedocs(). <p>PNG and JPEG raw picture support</p> <ul style="list-style-type: none"> - Can now insert raw PNG or JPEG images or return raw images for calculated picture fields. See Calculated Picture and the \$::insert method relating to picture objects. - New load data option kWriLoadRawPicts. - New OWrite option \$pasterawpicts. <p>All changes made to the examples are marked with a change marker and a date. All recent changes can be found by searching for "CHANGE_2014". This will display all changes made in 2014. Searching for "CHANGE_2014_05" will display all changes made in May 2014, searching for "CHANGE_2014_06" will display all changes made in June 2014, etc.</p>
3.0.6	\$owrite_export can now return full hyper-links as in ''
3.0.5	<p>Changes:</p> <ul style="list-style-type: none"> - New parameter bSortInUse for \$getfontlist. - The method \$loadfontmap can now be used on the web-client - Web-Client examples have been updated to work with new version 3 features.
3.0.0	<p>New Main Features: Bookmarks, Advanced Font Handling, Headers & Footers, Split Table Rows, Editing Evaluated Documents, Multi-Selection Find, Plain Text Analyzes.</p>
	<p>New OWrite document object methods: \$setdatafromsrc, \$getbookmarks.</p>
	<p>New OWrite document object properties: \$showinvisibles, \$headfootenabled, \$headermargin, \$footermargin, \$headfootoddeven, \$headfootfirstpage, \$firsttabiconid, \$notableoutline, \$nosspellcheck, \$curobjcontainer, \$linktextcolor, \$linktextstyle, \$maximagesize, \$newprimeasure, \$curhighlight, \$curlistnumstart, \$curbookmark, \$curtblrowevalcansplit, \$curobjdatasrc, \$curobjnoenter, \$curobjname, \$curobjdisplay, \$curobjcalc, \$curobjclickcalc, \$curtblname, \$curtblcalc, \$curobjdatadpi.</p>

	<p>New OWrite document object events: <code>evGetDataFromSrc</code>, <code>evMultiFind</code>.</p>
	<p>Improvements to OWrite document object properties: <code>\$scurfontcolor</code>, <code>\$scurtbldata</code>.</p>
	<p>New OWrite search object methods: <code>\$setresultlist</code>, <code>\$getresultlist</code>.</p>
	<p>New OWrite search object properties: <code>\$multiselection</code>, <code>\$multiselectlist</code>, <code>\$multiselectbackcolor</code>, <code>\$multiselecttextcolor</code>, <code>\$selectbackcolor</code>, <code>\$selecttextcolor</code>, <code>\$multiwordfind</code>.</p>
	<p>Improvements to static methods: <code>\$loadfontmap</code>.</p>
	<p>New static methods: <code>\$docversion</code>, <code>\$getfontlist</code>.</p>
<p>3.0.0 continued</p>	<p>New OWrite constants: <code>kWriObjTypeHeadFoot</code>, <code>kWriObjTypeInfo</code>, <code>kWriObjTypeInfoPgCnt</code>, <code>kWriObjTypeInfoPgNum</code>, <code>kWriObjTypeInfoDate</code>, <code>kWriObjTypeInfoTime</code>, <code>kWriMenuItemEditUndo</code>, <code>kWriMenuItemEditRedo</code>, <code>kWriMenuItemEditCut</code>, <code>kWriMenuItemEditCopy</code>, <code>kWriMenuItemEditPaste</code>, <code>kWriMenuItemEditClear</code>, <code>kWriMenuItemEditSelectAll</code>, <code>kWriTextCreateMap</code>, <code>kWriHtmlNoAutoSize</code>, <code>kWriHtmlBgColor</code>, <code>kWriHtmlTitle</code>, <code>kWriSelectHeadEvenDef</code>, <code>kWriSelectFootEvenDef</code>, <code>kWriSelectHeadOdd</code>, <code>kWriSelectFootOdd</code>, <code>kWriSelectHeadFirst</code>, <code>kWriSelectFootFirst</code>, <code>kWriSelectHeadFoodNext</code>, <code>kWriSelectHeadFoodPrev</code>, <code>kWriSRPlainText</code>, <code>kWriSRBookmark</code>, <code>kWriErrNoTextMap</code>, <code>kWriErrEndOfTextMap</code>, <code>kWriErrNotMultiSelect</code>.</p>
<p>2.6.0</p>	<p>New OWrite constants: <code>kWriHtmlNoAutoSize</code> and <code>kWriHtmlBgColor</code></p>
<p>2.4.1</p>	<p>New OWrite document object properties: <code>\$docwarnings</code></p> <p>New OWrite constants: <code>kWriErrWarning</code></p>
<p>2.2.4</p>	<p>New OWrite document object properties: <code>\$scurobjuserdata</code> and <code>\$scurtbluserdata</code>.</p>

<p>2.2.2</p>	<p>New OWrite document object properties: <code>\$scrtblmultidatarows</code>, <code>\$datanametype</code></p> <p>New OWrite constants: <code>kWriSpIIInvalSelection</code>, <code>kWriSpIIInvalAll</code></p> <p>Other enhancements: The table notation <code>\$table.\$total</code> can now specify cells from other tables in the document, i.e. <code>\$table.\$total(tableA.cell1,tableB.cell1,2)</code> The last parameter of the table notation <code>\$table.\$total</code> can now specify a <code>jst()</code> style formatting string as in <code>\$table.\$total(cell1,cell2,"N2,")</code></p>
<p>2.2.0</p>	<p>New OWrite constants: <code>kWriSaveNonUnicode</code>, <code>kWriTextFmtUTF16</code> and <code>kWriTextFmtUTF8</code>.</p> <p>Other enhancements: Unicode support. Both unicode and non-unicode versions of OWrite now save document data in UTF8 format. That means that documents saved with version 2 cannot be loaded by previous versions of OWrite.</p>
<p>2.1.2</p>	<p>New OWrite document object properties: <code>\$docuserdata</code>.</p> <p>Improvements to: <code>\$mergedocs()</code></p> <p>New OWrite constants: <code>kWriHtmlRawSupport</code></p>
<p>2.0.0</p>	<p>New Main Features: OWrite Table fields</p> <hr/> <p>New OWrite document object methods: <code>\$picturefrompage()</code>, <code>\$addstyle()</code>, <code>\$removestyle()</code>, <code>\$getselection()</code>, <code>\$convselection()</code>, <code>\$tableaction()</code>, <code>\$globalpos()</code></p> <hr/> <p>New OWrite document object properties: <code>\$readonly</code>, <code>\$forecolor</code>, <code>\$backcolor</code>, <code>\$backpattern</code>, <code>\$scrtbldata</code>, <code>\$scrtblresult</code>, <code>\$scrtblcellspacing</code>, <code>\$scrtblid</code>, <code>\$scrtblrowid</code>, <code>\$scrtblpageheaders</code>, <code>\$scrtblpagefooters</code>, <code>\$scrtblextendable</code>, <code>\$scrtblrowtype</code>, <code>\$scrtblcolumnwidth</code>, <code>\$scrtblrowheight</code>, <code>\$scrtblapplymode</code>, <code>\$scrtblalign</code>, <code>\$scrtblindent</code>, <code>\$firstselrow</code>, <code>\$lastselrow</code>, <code>\$firstselcol</code>, <code>\$lastselcol</code>, <code>\$scurobjframeoptions</code></p> <hr/> <p>Improvements to OWrite document object methods: <code>\$print()</code>, <code>\$popupmenu()</code>, <code>\$setselection()</code>, <code>\$getobjslis()</code></p> <hr/> <p>Improvements to OWrite document object properties: <code>\$scurobjautosize</code>, <code>\$scurstrikethrough</code></p> <hr/> <p>Improvements to OWrite document object events: <code>evPaperChanged</code></p>

New OWrite constants:

`kWriObjTypeTable`, `kWriObjTypeTableRow`, `kWriObjTypeTableCell`

New OWrite constant groups:

`kWriSelect...`, `kWriSR...`, `kWriTblRow...`, `kWriTblAct...`, `kWriFrame...` and
`kWriTblApply....`

Introduction

Overview

The OWrite software consists of the following

- i. The OWrite example classes in the "OWriteDocumentManager.lbs" library.
- ii. The external component libraries OWrite.xcomp (OWrite.dll on windows)
- iii. The JS client components OWriteW.jscomp for Omnis and Client (OWriteW.dll on windows)
- iv. The spell checker examples and OSpell2 external component library. For a description of the spell checker software please refer to the separate spell checker documentation.

For an introduction on how to integrate the OWrite software into your library, please read the chapters Designing OWrite and Designing JS-OWrite.

Examples

There are a number of classes that demonstrate the use of the OWrite software. These classes are mainly concerned with the OWrite interface and consist of menus, windows and object classes. The example library also contains classes that demonstrate the use of the spell checker software. These classes are not covered by this documentation. All example classes are organised into separate folders. All OWrite Desktop classes are located in the folder *OWrite Examples* and all JS-OWrite server classes are located in the folders *OWrite Javascript* and *OWrite Javascript Simple*

External component Library

The external component library consists of the main editor control and non visual objects for procedural document handling, including printing and searching without the use of the window control. A number of constants are defined for the use with properties and methods of the component objects. These constants can be accessed from the Omnis Catalog -> Constants -> OWrite. For a description of the external components please read the chapter External Component Reference

Designing OWrite

Introduction

You should read this entire chapter before attempting to develop OWrite version 3 or later for the first time, even if you are already familiar with OWrite and have read this chapter before. For version 3, Large sections of this chapter are new and other sections have been substantially updated.

This chapter gives a brief description of what is involved to add OWrite to your application. For a more detailed description of the example classes and a complete list of the external component properties and methods please refer to the chapters Examples Reference and External Component Reference.

If spell checking is a requirement, you should also read the separate OSpell2 documentation which can be found in the documentation folder in the OWrite Plus examples download. You should read this document first and pay special attention to the chapter “Integrating OSpell2”.

To use OWrite in its simplest form you drop the OWrite control onto your window class and add a few controls for applying formatting to the current selection. The section Formatting Text provides basic instructions for doing this. In addition, you will need to add code for loading and saving documents which is explained in the section Saving and Loading Documents.

To get an overall indication about the work involved to implement many of the OWrite features including the more advanced features such as page headers, page footers and tables, you should study this entire chapter in detail, starting with OWrite Basics.

Contents

OWrite Basics - describes basic features such as text formatting, find and replace, clipboard handling, and saving and loading documents.

Advanced Font Handling - describes font family and typeface interface handling as well as cross platform/cross word processor font mapping.

Headers & Footers - describes the page headers and footers feature

Data Merging - describes how data merging works and what OWrite document objects can be used to merge data.

OWrite Tables - describes in detail the OWrite table feature and how it can be used to merge Omnis list data.

Multi-Selection Find - describes the OWrite multi-selection word or phrase search feature.

Plain Text Analyzes - describes how plain text selection ranges can be used to select content in a rich text OWrite document.

Web Client - gives a brief description of the limitations and differences one must be aware of when designing a web-client interface for OWrite.

OWrite Basics

With the efficient use of Omnis notation, OWrite has implemented a complete customizable interface that is simple and highly productive during implementation, but also very efficient in execution. The OWrite Document Manager example demonstrates how flexible and efficient the OWrite interface is, as the entire interface (with the exception of the rulers) is implemented using nothing more than standard Omnis controls and notation.

“OWrite Basics”, introduces most of the basic OWrite features and includes links to code that demonstrate their use (links to code examples will only work if this documentation is viewed from within Omnis using the example Documentation Manager library). Many OWrite features require nothing more than a simple introduction, whereas some of the more complex features have been given a more detailed explanation later in this chapter. “OWrite Basics” will merely concentrate on creating awareness and basic understanding of the more commonly used features.

Important Note: During copy and paste or import and export actions OWrite requires access to a variable called `OWRITE_TMP_VAR` while working with picture data. OWrite uses the image conversion functions provided by Omnis and requires this variable as storage so that these functions can access the image data. The variable `OWRITE_TMP_VAR` must be a binary variable and it must be an instance variable of the window instance or better still, a task variable that can be accessed by all methods that deal with OWrite documents.

Formatting Text

The bread and butter of every word processor is its ability to format text and other document objects such as pictures. In order to provide an interface for OWrite for many of the formatting options, simple controls such as the check box or radio button are sufficient when they are directly linked to formatting properties of the OWrite control using notation. Every OWrite formatting property that effects the current selection has two basic requirements.

1. the control must tell OWrite when the user wants to change the formatting, i.e. the user toggles a state by clicking the control.
2. the control must reflect any change in the current formatting as the user moves the input cursor.

For example, to provide a control to display and change the Bold state of the selected text, one can place a check box on a window and set the `$dataname` of the check box to `ivEditor.$curbold` where `ivEditor` would be an item reference variable to the OWrite control. Clicking the check box would result in `$curbold` being toggled. In order for the check box to reflect changes in the bold state as the user changes the current selection, the OWrite `$event` method must implement the `evFormatChanged` event and when the event is triggered, issue a redraw command for the check box. `evFormatChanged` example...

It is easy to see how simple it is to implement controls to manage OWrite properties such as `$curbold`, `$curitalic`, `$curunderline`, etc. OWrite implements many of these on/off state formatting properties for the current selection. Many other formatting options require no more than radio-button groups to choose between a small range of options and these can be coded just like check boxes. However, formatting options such as the current style name, font name and font size require the use of combo boxes or lists. Combo boxes and lists need a little extra work

as they require the building of Omnis data lists for the choices that are presented to the user. In addition, because of Omnis intricacies, the main data name of a combo box cannot be directly linked to OWrite properties. When the *evFormatChanged* event is received the variables linked to the combo boxes have to be populated from the appropriate OWrite properties and the controls have to be redrawn. Equally, when the user picks from a combo box list, or changes the content via the combo box edit field, code is required that assigns the appropriate OWrite properties. *evStylesChanged* example... *style picker* example...

OWrite implements two methods that can be used to build selection lists for styles and font names. The list of style names is provided by the OWrite control via the *\$getStylelist()* method. Whenever the list of styles is changed, either by a document being loaded, text from other editors being pasted, or styles being added by the user, OWrite issues an *evStylesChanged* event which can be used to rebuild the styles list. To build a list of font names, OWrite provides the method *\$getfontlist()* which can work in conjunction with the OWrite font mapping feature. Both these features are described in more detail later. *\$getStylelist()* example...

OWrite provides no functionality for building a font size list. *font sizes* example...

Advanced Formatting

Besides providing basic interface controls for toggling the various text styles you may also wish to provide interfaces for some of the more complex features. As some formatting options involve choices that are not easily represented using a simple control such as a check box, radio button group or combo box, you may have to implement windows that format complex features such as tabs, bullets and numbering or pictures and tables. The OWrite Document Manager example implements a large number of windows that demonstrate how more complex features may be formatted. These windows are designed so that they can be copied to your own library and used as a starting point for your own interface. Please refer to the Examples Reference for a description of some of these classes.

Context Menu

The OWrite Document Manager example provides a context menu that is sensitive to the current selection and will offer different choices depending on what is selected in the document. These choices may be related to the current bookmark, misspelled word or selected image, etc. OWrite provides a number of events that are triggered when the user right-clicks the document content, which are *evContextText*, *evContextSpell* and *evContextObject*. Some of these context clicks may require additional options within sub-menus which may be complex and varied. For this purpose, OWrite provides a method called *\$popupmenu()* for displaying a hierarchical context menu using Omnis lists which can contain sub-lists for your sub-menus, and which can easily be build on the fly. The same method also has a set of reserved menu item IDs, that implement standard edit menu options such as cut, copy and paste (see *kWriMenuItemEdit...* constants). The enabled state of these reserved menu items will be set by OWrite if they are present in the pop-up menu list. *context menu* example...

Bookmarks

Introduced in version 3, the property *\$curbookmark* sets or clears the bookmark for the selected text. One or more characters must be selected for this property to work. When assigning a bookmark name that already exists within the document, the assignment will fail. The method

`$getbookmarks(&List)` returns a single column list of current bookmark names. These names can be used with `$setselection` and the constant `kWriSRBookmark` to go to the specified bookmark in the document.

Example:

```
;; the following example positions the document at the first bookmark
Do ivEdit.$getbookmarks(BookmarkList)
Do ivEdit.$setselection(BookmarkList.[1].C1,0,0,kWriSRBookmark)
```

The OWrite plus document manager example implements the bookmark feature via the context menu. You can right-click selected text to add or remove a Bookmark. The context menu has a sub menu of bookmarks that exists within the document. Selecting a bookmark name from the sub menu will select the bookmark in the document. bookmarks menu example...

Bookmarks have no visible representation by default. To display bookmarks within the document, you must turn on `$showinvisibles`. In this mode book marked text is surrounded by a box.

Undo, Redo, Copy and Paste

OWrite will automatically plug into the standard Omnis *Edit* menu. If, however, you would like to have additional buttons on your window, or implement a context menu with edit menu options, this can be achieved by linking your buttons or menu code to the appropriate OWrite methods and properties.

The methods `$editcopy()`, `$editcut()`, `$editclear()`, `$editpaste()`, `$editselectall()`, `$editundo()`, `$editredo()`, implement standard edit menu functionality. context menu example...

In addition, you can use the properties `$undotext` and `$redotext` in the tool tips properties of your undo and redo buttons.

When you assign OWrite properties from methods, be aware that each property assignment is added to the undo buffer. If there are a number of property assignments that require to be collected as a single undo operation you can use the methods `$startundo()` and `$endundo(undo text)`. `$startundo()` example...

If you need to assign properties without affecting the undo buffer, you can assign the property `$undoenabled` to `kFalse` while you make your changes.

Paste from file

Paste from file has been implemented as an event. When the user selects this option from the edit menu, OWrite generates an `evPasteFF` event. What you do with this event is entirely up to you. Typically you would present the user with a file selection dialog and convert the data from the file to a format compatible with OWrite. You can then use the `$::insert` method to insert the data at the current position in the document. paste from file example...

View Options

OWrite has a number of view options. You can show or hide rulers by toggling the property `$::showrulers`, and switch between Normal, Page Layout and Field view by assigning on of the `kWriView...` constants to the `$pageview` property. You can link radio buttons and check boxes to these options by assigning `ivEditor.$pageview` and `ivEditor.$showrulers` as their data names. The

variable *ivEditor* would be an item reference to the OWrite control.

Saving and Loading Documents

To save and load your documents use the methods *\$savedata()* and *\$loaddata()*. OWrite supports four formats during saving and loading.

1. OWrite binary format. It is recommended that you store your master documents in this format. Use binary variables for storage. Example...
2. RTF. Use this format for distribution to clients who do not have OWrite. You can use text variables for storage in the non-Unicode version of Omnis. But in the Unicode version, **binary fields must be used** when saving to RTF. RTF is saved as 7bit ANSI characters. You can load RTF documents produced by other word processors, but be aware that OWrite does not support all RTF and some formatting may be ignored during loading. Example...
3. HTML (v1.50). This format is for saving documents as HTML. You cannot load data in this format. You can use text variables for storage in the non-Unicode version of Omnis. But in the Unicode version, **binary fields must be used** when saving to HTML. HTML is saved as 7bit ANSI characters in the non-Unicode version and UTF8 in the Unicode version. Example...
4. Plain text. In the Unicode version, OWrite can save plain text as UTF32, UTF16, UTF8, Mac Roman and Win ANSI character sets. You can use text variables for UTF32 but **binary fields must be used** for any of the other formats. Example...

The constants *kWriFmtDefault*, *kWriFmtRTF*, *kWriFmtHTML*, and *kWriFmtText* are provided to specify the format.

In addition, *\$savedata* has custom parameters. Parameters that are specific for individual formats. See *kWriHtml...*, *kWriSave...* and *kWriText...* for details of custom parameters.

Saving to HTML

When saving data to HTML, OWrite will require additional information prior to starting the export and during the export. To this end additional custom parameters can be passed to *\$savedata*. During the export OWrite will call the method *\$owrite_export* to resolve images and hyper-links.

The additional parameters are described in the *External Component Reference* sections *kWriFmt...* and *kWriHtml...*. The export method *\$owrite_export* we will discuss here.

The export method must be implemented in the current window class that contains the OWrite control, or the object class that is derived from the OWrite non-visual object. The method receives two parameters: the export format, this will be set to 'html'; and the export type, either 'href' for hyper-links or 'src' for images. The method is required to return a valid HREF path for the former, and a valid SRC path for the later. For HREF requests you may also return a fully formed hyper-link tag such as '' (requires OWrite version 3.0.6). *\$owrite_export* example...

In future version of OWrite the export method may be called for other export formats also.

Find and Replace

OWrite supports the finding and replacing of text and styles. To implement find and replace two instances of the external non-visual search objects *OWriteSearch* are required. These two objects are used to specify the search criteria and the replace information. To execute a search or replace operation you call the methods `$findnext()` or `$replace()`.

find & replace example window...

Note: When passing objects to OWrite methods you must use the notation *searchInfo.\$owriteobj*.

In addition to the standard find and replace feature, OWrite also provides a multi-selection find, a feature that can search for multiple words or entire phrases and simultaneously highlight all occurrences within a document. This feature is explained in more detail in the section Multi-Selection Find towards the end of this chapter.

Printing

To print an OWrite document you call the method `$::print()` or you can place an OWrite report object in your Omnis reports. mail merge omnis report example...

The `$::print()` method starts an Omnis print job that can be sent to any Omnis print destination. `$::print()` example...

Important Note on Printing:

When printing OWrite documents via an Omnis report to the Disk destination and when viewing this report on another platform, in prior versions, some of the words would overlap or gaps between words were larger than they should be. It depended very much on the font and sizes that are used. This was only an issue when viewing the report on screen. When the report was printed the text was positioned more accurately.

The screen appearance issue was caused by an inability to accurately map some font sizes between platforms because of the difference in DPI between Macintosh and Windows. OWrite measures, positions and renders individual words and as a result the cross platform inaccuracy was effecting the space between the words. For example when using a 10 point font on Macintosh at 72DPI this should become 13.33 points at 96DPI on Windows. However, Omnis is forced to either choose 13 or 14 as the size to render the 10 point text. This up-sizing or down-sizing was causing the issues that were experienced.

In corporation with Tiger Logic we have provided a solution that allows OWrite to add special text to an Omnis report that is measured more accurately and allows the exact placing of individual characters in the horizontal plain. This feature is disabled by default and can be enabled by setting the property `$newprimeasure` to `kTrue`, but it will require Studio 5.2 or better.

This feature is not compatible with the current release version of PDFDevice. A new updated version will be provided in due course.

Printing Watermarks

During printing text based watermarks can be added to the specified document pages by assigning the property `$watermarks`. This property can be assigned with the name of a list (instance or task variable) or list data directly. The list must contain the following columns:

Page: One of the new kWriWM... constants or a positive value in the range 1 to n indicating a specific page number.

Angle: The angle in degrees (0 = horizontal, 90 = vertical-up, 270 = vertical-down)

InFront: If true, the watermark text is displayed in front of the document content

Font: The name of the font (normal OWrite font mapping rules are applied)

FontStyle: The style of the font, i.e. kPlain, kBold, kItalic, kUnderline

FontSize: Size of the font in points

Text: The watermark text

TextColor: The text colour (can use rgb() function or Omnis colour constants)

HorzPos: Horizontal offset from the left paper edge in cms or inches

VertPos: Vertical position from the top paper edge in cms or inches

One can query the \$watermarks property to receive a defined list with these columns in place. For example:

```
;;clear current watermarks and fetch empty defined list
Calculate ivEdit.$watermarks as #NULL
Calculate myList as ivEdit.$watermarks
;; add watermarks to list
Do myList.$add(kWriWMallPages,45,kTrue,"Arial",kPlain,24,"Brainy Data
                Limited",kDarkRed,1,15)
Do myList.$add(2,305,kTrue,"Arial",kItalic,24,"Brainy Data
                Limited",kGreen,8,5)
;; apply the watermarks
Calculate ivEdit.$watermarks as myList
```

There is no limit to the number of watermarks in the list.

Watermarks are not displayed in the editor and are only added to the content during printing. When using watermarks with the report object, watermarks are only displayed if \$ignorepos is set to kTrue. Watermarks are NOT saved with the document data. If watermarks belong to a specific document, they could be saved as part of the \$userdata or as a separate column in the document's record.

Spell checking

To enable spell checking for OWrite, you must integrate the spell checker software with your library. The steps required to integrate the spell checker are fully documented in the OSpell2 documentation inside the Documentation's folder.

In addition to the instructions given in this document you must initialise a spell checker session for the OWrite background spell check during construction of your OWrite window.

In your windows \$construct method, first initialise your main spell checker object. It should have been implemented as a task variable that is accessible from your OWrite window. Next initialise the spell checker session for OWrite. The spell checker session should be an instance of an object class that is derived from the external object Spll.SpllSession. Once you have initialised the session, OWrite will directly communicate with OSpell2. OWrite spell checker example...

Translation

OWrite supplies a small number of text strings for the undo menu and buttons and for the display of page breaks and header and footer text. You can translate these text strings by editing the string table *OWriteStrings* in the examples folder.

You must load the string table from your Omnis code and call the OWrite static method \$loadstrings.

Example:

```
;; our string table is located with the library
Do FileOps.$splitpathname($clib.$pathname,DRIVE,DIR,NAME,EXT)
Do StringTable.$loadstringtable('OWriteStr',con(DRIVE,DIR,'OWriteStrings'))
;; set the appropriate column
Do StringTable.$setcolumn('Deutsch')
;; tell OWrite to load the strings
Do OWrite.$loadstrings('OWriteStr')
;; we have finished with the string table, unload it.
Do StringTable.$unloadstringtable('OWriteStr')
```

You only need to do this once per Omnis session and it can be done from any of your libraries during startup or any other appropriate time. If you change the OWrite strings while displaying documents, you must issue redraw commands to update the displayed documents.

Translating the web client version of OWrite has to be implemented differently. The web client does not support string tables. OWrite provides the two properties \$stringlist and \$stringcolumn. These need to be assigned on the server during \$construct.

Web Client Example:

```
;; our string table is located with the library
Do FileOps.$splitpathname($clib.$pathname,DRIVE,DIR,NAME,EXT)
Do StringTable.$loadstringtable('OWriteStr',con(DRIVE,DIR,'OWriteStrings'))
;; set the appropriate column
Do StringTable.$setcolumn('Deutsch')
;; tell OWrite to load the strings
;; OWrite is the item reference to the form control
Calculate OWrite.$stringlist as StringTable.$loadlistfromtable('OWriteStr')
Calculate OWrite.$stringcolumn as StringTable.$getcolumnnumber('OWriteStr')
;; we have finished with the string table, unload it.
Do StringTable.$unloadstringtable('OWriteStr')
```

The property assignments have to be done for every client, but the string table loading could be done just the once and the resulting list and column number could be cached.

In addition, you may need to change the decimal tab character from a dot to a comma for some countries. This can be done by assigning the property \$docdecimaltabchar.

Font Mapping

For documents to be displayed correctly on all platforms, you can manage your own font

mapping table. The example library implements the window *wFontMapping* that manages the font mapping table in a disk file. OWrite implements two static functions to facilitate the storage of Omnis list data in a binary file on disk. The function `OWrite.$listtobin` converts a list to binary data, and the function `OWrite.$listfrombin` converts it back again. font map window example...

To turn on font mapping you must load the font mapping table during startup. You load the table from disk and call the static method `OWrite.$loadfontmap`. The `$construct` method of the startup task in the example library demonstrates how to load a font table. load font map example...

The detailed behaviour of the font map and its varied uses will be discussed later in the section *Advanced Font Handling*.

Document Objects

Besides rich text, OWrite supports the following document objects.

- Calculated Fields (see the section “Objects with data merging capabilities”)
- Pictures/Calculated Pictures
- Text boxes
- Tables
- Headers and Footers

Pictures, text boxes and tables can be copied from other word processors that support RTF and pasted from the clipboard directly into the OWrite document.

All objects, except headers and footers, can be inserted at the current cursor position from Omnis code. The OWrite method `$.insert()` takes a number of parameters that tell OWrite the type, associated data and basic properties. When an object is inserted it is automatically selected, and additional object properties can be assigned or queried.

The OWrite table, picture, header and footer objects will be explained in more detail later on.

Objects with data merging capabilities

The beating heart of OWrite is it's ability to merge Omnis data from a database or other sources using standard Omnis calculations or method calls that can be directly embedded in the document content. Because these embedded calculations can include any valid Omnis function or Omnis notation string including notation method calls, there is no limit to their ability to build, control and format the data that is merged into the document. As a result, the user interface for merging data can be as simple or as rich as a developer requires.

The facilitators for data merging are some of the OWrite objects described in the previous section. The *Calculated Field*, the *Calculated Picture*, the *Text box* and the *OWrite Table* object implement properties for managing the objects data merging features. Data merging is documented in more detail in the *Data Merging and OWrite Tables* sections of this chapter.

Objects with Hyper-Links

As well as providing data merging capabilities, some objects such as the picture and calculated field objects, provide properties for managing hyper-links. When turning on the property `$curobjclicks`, the object can be clicked by a user. The additional property `$curobjclickcalc`, just as with `$curobjcalc`, accepts any valid Omnis function or notation string, but in this case the

calculation is executed when the user clicks the object. If \$scurobjclicks is true, but \$scurobjclickcalc is empty, OWrite will generate the evObjClick event instead.

Web-Client Note: Because it is not possible to execute custom calculations on the web-client, even if \$scurobjclickcalc contains a calculation, OWrite will always generate the evObjClick event to facilitate the execution of the click calculation on the server.

Picture alternative data

Having many documents stored in a database can increase the size of a database substantially if these documents contain images such as logos. OWrite provides a feature that prevents OWrite from storing image data in the OWrite document directly. Instead, a picture object can be given a calculation that fetches the image data when required. This can substantially reduce the size of a database if documents share the same images.

To use this feature you assign an Omnis calculation, i.e. a notation method call such as \$ctask.\$getImage(Logo) to the property \$scurobjdatasrc. When OWrite loads the document, it will execute the calculation to fetch the picture data.

Web-Client Note: Because it is not possible to execute custom calculations on the web client, OWrite implements an event and method to facilitate the execution of the calculation on the server. When the OWrite web-client control loads a document, it will generate the event evGetDataFromSrc, passing it the parameters pObjID, pObjName and pObjCalc. The final parameter contains the calculation from \$scurobjdatasrc. On receiving the event you can call a server method to evaluate the calculation using the eval() function and on return, you can use the method \$setdatafromsrc(pObjID,pObjData) to set the image data for the object in the OWrite remote form control.

OWrite Version Numbers

In order to ensure that you distribute the correct DLL's it is a good idea to always check the version numbers of the external components during startup of your libraries. OWrite version 3 has been changed to return a version number that allows the checking of the OWrite product version and the Omnis Studio version for which the external component DLL was build. This has become necessary as the OWrite product ships with three different sets of DLLs for three different versions of Omnis Studio. It is vital that the correct DLL is used to prevent errors.

From OWrite version 3 onwards, the version number returned by **\$components.OWrite.\$version** will be xx.yyy. The xx portion stands for the 2 digit Studio version 52 for Studio 5.2, 50 for Studio 5.0 or 43 for Studio 4.3 or better. The yyy portion of the version number is the OWrite version number as a three digit value. For version 3.0.0 this will be 300. It is prudent for your libraries to check both the Studio version and the OWrite version to prevent using the incorrect DLL. see sample code

Document Version Numbers

OWrite implements the static method OWrite.\$docversion which returns the internal document version of an OWrite binary. This method can be used to test if a document may require additional conversion by the developer. The examples implement a conversion method that alters the formatting around some calculated fields. A change that was prompted by the new version 3 hyper link formatting properties \$linktextcolor and \$linktextstyle. conversion example...

www.brainydata.com



Advanced Font Handling

OWrite version 3 changes the way fonts are mapped when importing and exporting documents, when pasting content from the clipboard and when mapping fonts between platforms. The new font mapping functionality can also be used to limit the set of fonts that are available to the end user and that OWrite will allow during import or pasting. In addition, OWrite now provides new features that allows the displaying of font family and typeface names in way more consistent with other word processors.

At that heart of the new font handling are two lists. A list of all available fonts build by OWrite during startup, referred to from heron as the *OWrite-Font-List*, and the font map which can be installed via the `$setfontmap` function, referred to from hereon as the *OWrite-Font-Map*.

The OWrite-Font-List

The *OWrite-Font-List* is returned by the static function

```
OWrite.$getfontlist(&lList, cCurFont, bCurBold, bCurItalic, bApplyMap,
lSizeList)
```

or by the equivalent OWrite document object function

```
OWriteRef.$getfontlist(&lList, cCurFont, bCurBold, bCurItalic, bApplyMap,
lSizeList)
```

List Columns

The list returned by `$getfontlist` consists of three main columns:

Family: the fonts family name

Typefaces: sub-list of font type-faces consisting of four columns

Typeface: The typeface name, i.e. Bold, Italic, Regular, Condensed, etc

OmnisFontName: The Omnis equivalent font name

OmnisStyleBold: The Omnis font bold state used with OmnisFontName

OmnisStyleItalic: The Omnis font italic state used with OmnisFontName

Used: flag that indicates if the font is used or has been used in the OWrite non-visual, window or remote form object session. All used fonts will appear at the top of the list, separated by an empty divider row. This column is only populated when the `$getfontlist` function is called via a reference to an OWrite document object.

\$getfontlist - Example of use:

```
OWriteRef.$getfontlist(ivFontList, OWriteRef.$scurfont, OWriteRef.$scurbold, OWriteRef.$scuritalic, kTrue, ivSizeList)
```

\$getfontlist - Parameters:

lList: The variable that is to receive the list of fonts. When calling this function on the web-client, due to web-client limitations, the list must be an instance variable and the lists name must be specified using the `nam()` function.

```
OWriteRef.$getfontlist(nam(ivFontList), ...)
```

cCurFont: The current font as returned by `$scurfont`. Passing this information in conjunction with the next two parameters will select the appropriate font family and typeface in the

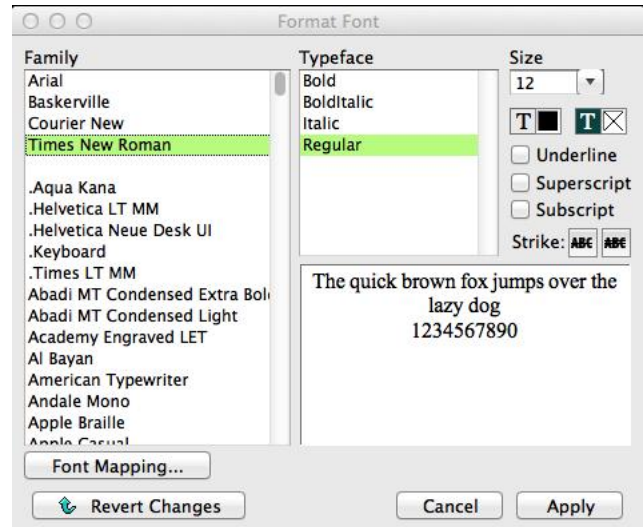
returned list.

bCurBold: The current bold state as returned by \$curbold. See cCurFont.

bCurItalic: The current italic state as returned by \$curitalic. See cCurFont.

bApplyMap: If kTrue is passed for this parameter and the *OWrite-Font-Map* was loaded with the flag set to limit OWrite fonts, the list returned by \$getfontlist will only include fonts that are specified in the *OWrite-Font-Map* and that exist on this platform. Passing kFalse for this parameter will return all available fonts regardless of the settings for the *OWrite-Font-Map*.

ISizeList: The list returned by the method will contain the font heights for all installed fonts, or all fonts specified in the *OWrite-Font-Map* if *bApplyMap* is true. Heights are specified in the font's unit size. The unit size is also included with the data.



When using the *OWrite-Font-List* in the user interface for selecting fonts and typefaces, the three columns *OmnisFontName*, *OmnisStyleBold* and *OmnisStyleItalic* in the typeface list are used to apply the chosen font and typeface via the OWrite properties \$curfont, \$curbold and \$curitalic. The Format Font window has been updated to use this new interface for selecting fonts.

Internally, the *OWrite-Font-List* consists of additional information such as postscript names for all font type-faces. This information is not available to developers but is used to find unknown font names when font names are assigned via the \$curfont property, or when documents are imported or pasted from the clipboard via RTF. This means that it is perfectly valid for example to assign ArialMT (the postscript name for Arial) to \$curfont. OWrite will convert this to the Omnis equivalent font name Arial.

The OWrite-Font-Map

The *OWrite-Font-Map* is not new to OWrite version 3. In the past it was used to map font names when loading documents that were created or edited on other platforms. The font map list had to provide rows of font names where each row provided three columns for the three supported platforms, Macintosh, Windows and Linux.

In version 3, the font map is essentially still a list, but the columns of the list are no longer linked to a specific platform, and the number of columns are no longer limited to three.

THE PURPOSE OF THE NEW FONT MAP

- Provide alternative font names for exporting to HTML
- To map unknown font names from other platforms or programs
- To limit OWrite to the specified set of fonts

ALTERNATIVE FONTS FOR HTML EXPORT

It is common practice when specifying font family names in HTML to specify a list of alternative names in a comma separated list. A browser will typically choose the first font in that list which is available on the client's platform. When exporting OWrite documents to HTML, OWrite will use the line from the map that contains the font name to be exported in the left-most column of the map. For example, if row two contains the font in third place and row five contains the font in second place, OWrite will pick row five as the list of font names for export to HTML.

MAPPING UNKNOWN FONT NAMES

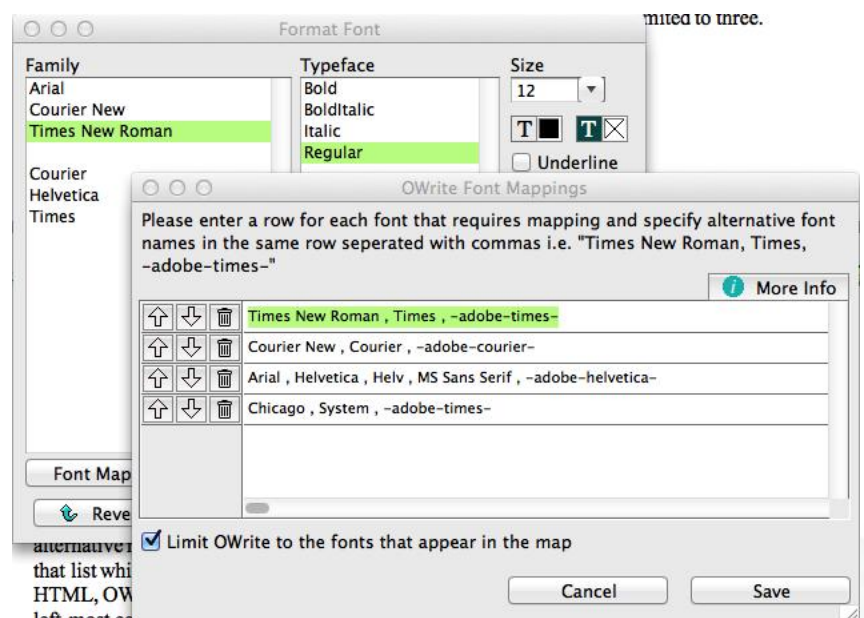
When loading or importing documents from other platforms or programs or pasting content from the clipboard, the documents or content may contain font names unknown to this platform. OWrite will first search for the font in the system, comparing it with the systems family and postscript names of each font and typeface. If this search fails, OWrite will consult the font map and use the row that contains the font name in the left-most column (see 'Alternative Fonts...'). If this second search also fails, OWrite will use the first row in the map to choose the first available font starting from the left. If no font-map is provided, OWrite will use as its default font the font specified in the \$font property.

LIMIT OWRITE TO THE SPECIFIED FONTS

When loading the font-map via the method \$loadfontmap one can tell OWrite to limit the fonts that can be used in a document to the fonts specified in the map. **We especially recommend this for JS-OWrite.**

When loading OWrite documents or importing documents via RTF or pasting content from the clipboard, OWrite will only allow fonts specified in the map. If fonts are encountered that are not specified in the map, OWrite will use the fonts specified in the first row of the map and pick the first one that exists on the clients system.

The same limitation is applied when calling \$getfontlist to fetch the list of available fonts. If the *OWrite-Font-Map* is set to limit the fonts and the \$getfontlist function is called with parameter *bApplyMap* set to kTrue, the *OWrite-Font-List* will populate the interface list with only fonts that are present in the *OWrite-Font-Map*.



Headers & Footers

Page headers and footers are new to version 3. This section describes their basic behaviour and programming features.

Header & Footer options

By default, the headers and footers feature is enabled. It can be disabled by setting the property `$headfootenabled` to `kFalse`. If disabled, the user will not be able to see or modify headers and footers. It is a property of the window control and is not stored with the document.

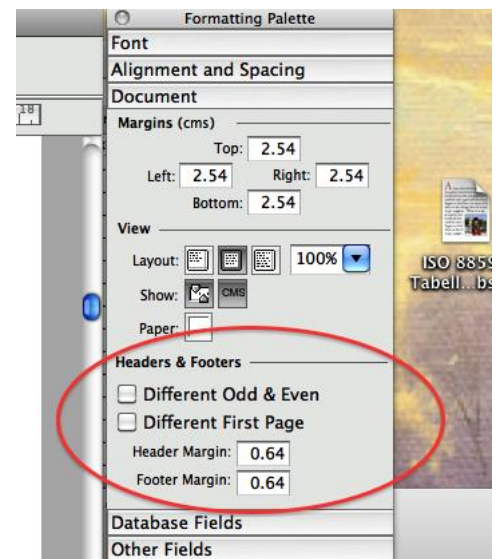
There are a number of header and footer options. The examples implement these options in the Document section of the OWrite Formatting Palette.

Header Margin (\$headermargin), specifies the distance between the header and the top edge of the paper as centimetres or inches. This is a document specific property and is stored with the document data.

Footer Margin (\$footermargin), specifies the distance between the footer and the top edge of the paper as centimetres or inches. This is a document specific property and is stored with the document data.

Different Odd & Even (\$headfootoddeven), if true, creates different headers for odd and even pages of the document. This is a document specific property and is stored with the document data.

Different First Page (\$headfootfirstpage), if true, creates different header and footer for the first page. This is a document specific property and is stored with the document data.



All-in-all, there are six possible headers and footers (three of each). A header and footer for the first page, a header and footer for subsequent even pages, and a header and footer for subsequent odd pages.

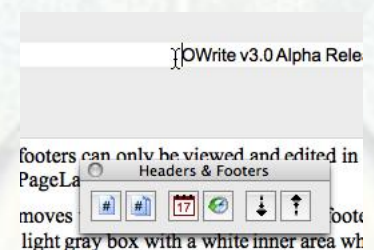
Interface Behaviour and Appearance

Headers and footers can only be viewed and edited in the page layout mode (`$pageview` is set to `kWriViewPageLayout`).



When a user moves the mouse across a header or footer, OWrite indicates the header/footer location by a light gray box with a white inner area where the header/footer content is positioned (image above).

This box remains when the header/footer is selected for editing. The examples have been programmed to open a special formatting palette while editing a header or footer. This palette implements buttons to insert special date, time, page number and page count objects. It also includes buttons to move to the next or previous header or footer (see image to the right).



NOTE: It is not possible to edit headers or footers of a type for which there are no pages in the document.

When OWrite creates headers and footers in a document, OWrite automatically creates a new style named Header & Footer.

Selecting Headers & Footers

It is possible to procedurally set the selection inside a header or footer. There are new kWriSelect... constants that can be used with the \$setselection method in place of an object ID.

kWriSelectHeadEvenDef: Used as the object ID with \$setselection, it sets the selection in the even page and default page header.

kWriSelectFootEvenDef: Used as the object ID with \$setselection, it sets the selection in the even page and default page footer.

kWriSelectHeadOdd: Used as the object ID with \$setselection, it sets the selection in the odd page header

kWriSelectFootOdd: Used as the object ID with \$setselection, it sets the selection in the odd page footer

kWriSelectHeadFirst: Used as the object ID with \$setselection, it sets the selection in the first page header

kWriSelectFootFirst: Used as the object ID with \$setselection, it sets the selection in the first page footer

kWriSelectHeadFoodNext: Select next header or footer. From the current selection point, this option will select the next footer or header in line.

kWriSelectHeadFoodPrev: Select previous header or footer. From the current selection point, this option will select the previous footer or header in line.

Example:

```
Do OWriteRef.$setselection(0,0,kWriSelectHeadEvenDef)
;; sets the cursor at the beginning of the default/even page header
```

Header and Footer Object Types

There are two object types (\$curobjtype) related to the implementation of headers and footers. One is the type for a header and footer object itself, the other for an info object that can be placed inside a header and footer and show information such as the current time or page count. The info object has a subtype that indicates its function. You use the following constants when inserting an info object.

kWriObjTypeHeadFoot: Document header or footer.

NOTE: You cannot insert objects of this type. They are automatically created by OWrite.

kWriObjTypeInfo: Document info object such as page number or count. You can insert info objects of the following sub types.

kWriObjTypeInfoPgCnt: Displays the total page count of the document

kWriObjTypeInfoPgNum: Displays the current page number

kWriObjTypeInfoDate: Displays the current short date (formatting specified by #FD)

kWriObjTypeInfoTime: Displays the current short time (formatting specified by #FT)

To insert an info object you use the `$::insert` message and specify the subtype as the second parameter. The property `$curobjdata` returns one of these constants.

Example:

```
Do OWriteRef.$::insert(kWriObjTypeInfo, kWriObjTypeInfoDate)
```

The info objects will export to RTF as compatible MS Word fields.

Printing

When printing a document directly using the OWrite `$::print()` method, headers and footers will be printed as they appear in page layout view. When printing using an Omnis report and OWrite report object, the properties `$ignorepos` and `$headfootenabled` must both be set to `kTrue`, for headers and footers to print.

Exporting & Importing

OWrite will currently only export headers and footers to RTF. There are no plans to export them to plain text or HTML. The importing of headers and footers via RTF is currently not supported and any headers and footers will be skipped.

Data Merging

As already touched on in a previous section, the most important OWrite feature is the way in which Omnis data from a database or other sources can be merged with OWrite document content. OWrite provides the *Calculated Field*, the *Calculated Picture*, the *Text box* and the OWrite *Table* object, all of which can be used to merge Omnis data in different ways.

The *Calculate Field* is a simple in-line object that can be used to merge text based data in the current paragraph. There is no limit to the amount of data that is merged via this object as long as it is text and does not contain any objects such as pictures or tables, but it may consist of multiple paragraphs. The data that is returned can be RTF based which allows the individual formatting of words or phrases within the text that is to be merged.

The *Calculated Picture* is an OWrite object that can merge picture data. The picture data must be Omnis picture data as accepted by the Omnis Picture data type or it can be raw PNG or JPEG data provided by a binary variable (requires version 3.5.0). OWrite currently does not support any of the other raw image formats that are supported by Omnis. OWrite picture objects can be formatted as in-line objects, or floating objects that cause text to wrap around the object in various ways, or as background objects that are positioned behind the document's text.

The OWrite *Text box* is useful for containing merged text data, which may consist of multiple paragraphs, within the rectangular area of the text box. A text box can be formatted just like a picture object so that the document's text wraps around the text box or is positioned behind or in front of it.

The OWrite *Table Field* is used for displaying Omnis list data. This object is documented in greater detail in the next section.

The calculated field, calculated picture and the text box have properties for assigning an object's internal name (see \$scurobjname), the display text or icon ID for picture objects when a document has not been evaluated (see \$scurobjdisplay) and the calculation for merging data (see \$scurobjcalc).

A document is evaluated by setting the property \$evalcalcs to true. At this point, OWrite will evaluate, in turn, the calculations of all the OWrite objects and replace/display the returned data. A already evaluated document can be populated with new data by simply assigning kTrue to \$evalcalcs a second or subsequent time.

By default, all calculations will be evaluated within the instance context. By context we mean where we look for variables if we encounter variable names within the calculations, and by instance context we mean the current window instance if we evaluating a document using the OWrite window object, or the object instance if we are using the OWrite non-visual document object. It is possible to create an object class and set as its super-class the external OWrite document object. When evaluating documents using this object, all calculations will look for variables or any notation that begins with \$cinst inside the object class.

Note: To ensure correct behaviour in all situations, all instance variable names should be prefixed with \$cinst. Our testing has indicated that when using object classes this is the only way to ensure correct behaviour across all supported versions of Studio.

However, sometimes it may not be desirable to evaluate calculations against the instance context

of the non-visual object or window object, regardless of whether the document is loaded in a window object or non-visual object. Perhaps you intend to have just one class that deals with all document evaluation. To cater for this situation, OWrite provides the property `$evallocal`. Assigning this property to true will change the OWrite context from the instance context to the local context. By local context we mean the context of the method that will assign `$evalcalcs` to evaluate the document. This means one can refer, in calculations, to local variables of the method, instance variables of the class instance that owns the method, or task variables of task instances that own the class instance of that method. This would be especially useful if one wants to use table classes to evaluate documents as the table class could encapsulate both the code to fetch data from a database and the code to handle the data merging of OWrite documents.

As already discussed in the section “OWrite Basics”, OWrite objects can be inserted using the `$::insert()` method. When inserting objects that can be used to merge data, the second parameter for the insert method is of special interest as it can be used to specify the object name, display, the calculation and the click calculation, using the special syntax

name~display;data_calc~click_calc. For example, the line

```
Do refOWrite.$::insert(kWriObjTypeCalc,  
"fldName~Client Name;$cinst.ClientName~$cinst.$showDetails('ClientName')")
```

would insert a calculated field with the internal name “fldName”, the display text “Client Name”, the data calculation “\$cinst.ClientName” and the click calculation “\$cinst.\$showDetails('ClientName’)”.

OWrite Tables

OWrite tables are an excellent medium for presenting Omnis list data. As well as the common standard header rows, OWrite tables provide footer rows that can display subtotal and total values, special frame options give the user effective control over the table appearance, and powerful table functions greatly simplify the more complex tasks. Of course, tables cannot only be used to present Omnis list data. They are just as useful for organising and presenting static content or are a useful tool for WYSIWYG label designers.

Tables are added to a document using the `$::insert()` method.

Table Row Types

As already mentioned, an OWrite table can consist of header rows, footer rows and normal rows. The following is a brief description of each row type.

Header Rows (`kWriTblRowHeader`)

Typically, header rows would not display data from an omnis list. They may simply provide column headers, or other related data. However, a header cell can be given calculations that can refer to Omnis data outside the list, or calculated fields can be directly inserted into a header cell which refer to such data.

Footer Rows (`kWriTblRowFooter`)

One good use for footer rows is to display subtotals and totals at the end of each page. For this purpose OWrite provides a special table function that can be assigned to a footer cell's `$curobjcalc` property. The function `$tables.$total()` displays either subtotals or totals in the appropriate places. It takes a variable number of parameters. You can specify one or more cell names (the name assigned to `$curobjname`), or the notation `table_name.cell_name` to reference cells in other tables. In the above example, the table name was specified together with the table's calculation in parameter two of the `$::insert` method. The final parameter specifies the number of decimal places or a `jst()` formatting string such as "N2," for the display of the result. The function will total/subtotal the results of all calculations of the provided names in previous rows or tables.

Normal Rows (`kWriTblRowNormal`)

A table's normal row, also referred to as a data row. Cells within this row type can be used to display data from list columns by assigning a calculation to the cell's `$curobjcalc` property.

Linking a Table to an Omnis List

A table can be given a standard Omnis calculation that returns an Omnis list or the name of an Omnis list variable (see `$curobjcalc`).

Important: If your list is an instance or task variable and your table calculation returns the name of the list, OWrite will operate directly on the instance/task variable. If your table calculation returns the entire list, Omnis will operate on a copy of that list and you must declare an instance or task variable called `OWRITE_TMP_VAR` of type binary. OWrite will use this variable to store the list so that Omnis calculations can evaluate against it.

When a document is evaluated (see `$evalcalcs`), OWrite will make sure that the table has

sufficient data rows (kWriTblRowNormal) to accommodate all of the list's rows. If need be, OWrite will add additional data rows. This action may result in additional header and footer rows being created at page boundaries. This is the only time OWrite will create header and footer rows. When editing a document, header and footer rows will not respect page boundaries and behave in the same way as normal rows. Header and footer rows are only intended for tables that display Omnis list data. Documents that have been evaluated and contain tables with headers and footers should always be edited in OWrite's single page mode (see \$papercontinuous) which is explained in mode detail later on.

As well as assigning a table calculation, each cell in a data row also requires a calculation. Any Omnis functions or notation can be used in the calculation as long as the calculation returns number, text, RTF or picture data. To refer to a column in the Omnis list you use the syntax \$ref.*ColumnName* (\$ref refers to the list returned by the table calculation).

If OWrite needs to add additional data rows during document evaluation, the last table row of type kWriTblRowNormal is used as the template for all additional rows that are added to the table.

Example:

Let us assume we have a contacts list in a file class or database table called “fClients” and the contacts list is called “Contacts”. The list has the columns “Name”, “EMail” and “Phone”.

In order to display this list we would need to create an OWrite table with a header row and a data row with three columns to display the three list columns. This table would be our template for displaying all of the list’s data. When the document is evaluated and our table grows to accommodate the list data, our header row will be repeated at the top of each page.

```

;; Insert a three column table with two rows (one header and one data row).
;; (after the insert the table will be selected and $curtblid will be set)
Do refOWrite.$::insert(kWriObjTypeTable,"Contacts~fClients.Contacts",
    kWriBordLine,kWriLineSolid,kBlack,lineSizeInPoints,
    columnCount,rowCount,columnWidth,rowHeight)

;; Make the first row a header row as new rows default to kWriTblRowNormal
;; - select row 1
;; (to select cells or rows in a table one specifies row*65536+cell)
Do refOWrite.$setselection(1*65536,1*65536,refOWrite.$curtblid)
;; - assign the row type
Do refOWrite.$curtblrowtype.$assign(kWriTblRowHeader)

;; insert the column header text "Name", "E-Mail" and "Phone" in each
;; header cell
;; - select cell 1 in row 1
Do refOWrite.$setselection(1*65536+1,1*65536+1,refOWrite.$curtblid)
;; - insert the column header text in the cell
Do refOWrite.$::insert(kWriObjTypeText,"Name")
;; - select cell 2 in row 1 and insert column header text
Do refOWrite.$setselection(1*65536+2,1*65536+2,refOWrite.$curtblid)
Do refOWrite.$::insert(kWriObjTypeText,"E-Mail")
;; - select cell 2 in row 1 and insert column header text
Do refOWrite.$setselection(1*65536+3,1*65536+3,refOWrite.$curtblid)
Do refOWrite.$::insert(kWriObjTypeText,"Phone")
;; Set the calculations for the cells in row two
;; The calculation will consist of the notation $ref which refers to the
;; Omnis list, and the column names "Name", "EMail" and "Phone"
;; We will also assign an internal name to each cell
Do refOWrite.$setselection(2*65536+1,2*65536+1,refOWrite.$curtblid)
Do refOWrite.$curobjcalc.$assign("$ref.Name")
Do refOWrite.$curobjname.$assign("Contacts_Name")
Do refOWrite.$setselection(2*65536+2,2*65536+2,refOWrite.$curtblid)
Do refOWrite.$curobjcalc.$assign("$ref.EMail")
Do refOWrite.$curobjname.$assign("Contacts_EMail")
Do refOWrite.$setselection(2*65536+3,2*65536+3,refOWrite.$curtblid)
Do refOWrite.$curobjcalc.$assign("$ref.Phone")
Do refOWrite.$curobjname.$assign("Contacts_Phone")

```

Now our table is ready to be evaluated.

Split Table Rows

From OWrite version 3 onwards, when merging list data with an OWrite table, it is possible to have table rows split across page boundaries if a table row contains at least two rows of text.

This feature only works when evaluating an OWrite table object that is linked to an Omnis list and it is not a general feature during the editing of a document. Editing a row that has been split will result in unpredictable behaviour. It must only be used as an aid for producing documents for viewing or printing. For a row to be split across pages you must set the property \$scrtblrowevalcansplit to kTrue for each row that you allow to be split. However, it is possible to edit documents with split table rows if the document is placed into the single page mode as explained in the next section.

Editing Evaluated Documents

One of the strengths of OWrite documents is the ability for users to edit a document after it has been merged with data. Something that is not possible with Omnis reports. It empowers users to make final tweaks to a merged document. However, as already mentioned, when tables with headers and footers are edited after a merge, changing the content of a table's cell may alter the position of the rows and any headers and footers that were added by OWrite during data merging, may no longer line up with the page boundaries.

In OWrite version 3 we have updated the property \$papercontinuous to respect evaluated documents in relation to page boundaries and table headers, footers and split rows when switching to or from the continuous paper mode.

When \$papercontinuous is turned on (single-page mode) and a table is subsequently evaluated, no extra headers and footers will be inserted and table rows will not be split across page boundaries as the table will be contained within a single extended page.

When turning off continuous paper (multi-page mode), OWrite will insert the appropriate headers and footers and split rows where page breaks are encountered. **IMPORTANT:** The code that assigns \$papercontinuous must have access to the same variables as the code that originally evaluated the document (assigned \$evalcalcs) so that OWrite can calculate the content for the extra headers and footers that it needs to insert.

In contrast, when an evaluated document is switched from multi-page mode to single-page mode, OWrite will strip any page boundary headers and footers that were added, and merge any rows that had been split.

This feature allows end users to edit evaluated tables in single-page mode without adversely effecting split rows or headers and footers that will be inserted around page boundaries in multi-page mode. When the user has finished editing, \$papercontinuous can be turned off to paginate the modified document for printing or exporting. The only limitation is that the user cannot edit headers and footers that are inserted when switching to the multi-page mode as they will not be available in the single-page mode.

IMPORTANT SIDE EFFECTS (new to version 3)

Evaluated documents save all result data for the table and all its cells. This is done so documents can be saved in single-page mode, allowing OWrite later on to calculate subtotals and other content for headers and footers that are inserted when switching to multi-page mode for printing or exporting.

To strip all result data you must save the document with parameter three (bMakeDataPermanent) set to kTrue. This will permanently fix all cell content and all other calculated fields without the possibility of re-evaluating the document.

Example:

```
Do OWrite.$savedata(binary, kWriFmtDefault, kTrue)
```

In addition, the `$papercontinuous` and `$evalcalcs` properties are now properties of the document and will also be saved and loaded with the document data.

STATIC TABLES (new to version 5)

Static tables (tables that have NO list attached) will now paginate correctly when using the `$papercontinuous` feature in order to edit tables in a single flow and then paginate the tables post-edit. Turning off `$papercontinuous` will produce correct table headers and footers and split table rows if the appropriate table properties are set, i.e. `$curtblpageheaders`, `$curtblpagefooters` and `$curtblrowevalcansplit` are set to `kTrue`.

For the process of creating, evaluating, editing and previewing document templates that contain static tables we recommend the following steps:

1. create document in `$papercontinuous` mode
2. evaluate document
3. edit post-evaluated document in `$papercontinuous` mode
4. preview paginated document by turning off `$papercontinuous` mode

We recommend that evaluated documents that contain tables for which headers and footers have been generated around page breaks and/or table rows have been split, are previewed with the document disabled, i.e. `$readonly.$assign(kFalse)`. However, the order of assigning `$papercontinuous` and `$readonly` is important as `$readonly` prevents programmatic changes to the document (including changes caused by `$papercontinuous`)

When editing the evaluated document, first set `$readonly` to `kFalse`, then assign `$papercontinuous` to display the document in a continuous flow, which will remove the inserted headers, footers and merge split table rows, so the table can be edited without destroying the integrity of the table.

Example:

```
;; clear $readonly first so $papercontinuous can make changes
Do ivEdit.$readonly.$assign(kFalse)
Do ivEdit.$papercontinuous.$assign(kTrue)
```

To preview document post-editing, turn off `$papercontinuous` first and then disable the editing.

Example:

```
Do ivEdit.$papercontinuous.$assign(kFalse)
Do ivEdit.$readonly.$assign(kTrue)
```

Multi-Selection Find

From version 3 onwards, it is possible to search a document for a matching phrase or for the individual words within the search string, and have OWrite highlight all matching content within the document. The *OWriteSearch* object has a property called `$multiselection` which turns on this feature and the property `$multiwordfind` which specifies if we are searching for the exact phrase or individual words. If `$multiselection` is set to `kTrue`, calling `OWriteObject.$findinit` will search the entire document and store matching content in a list. The property `$multiselectlist` will return a list with the following columns:

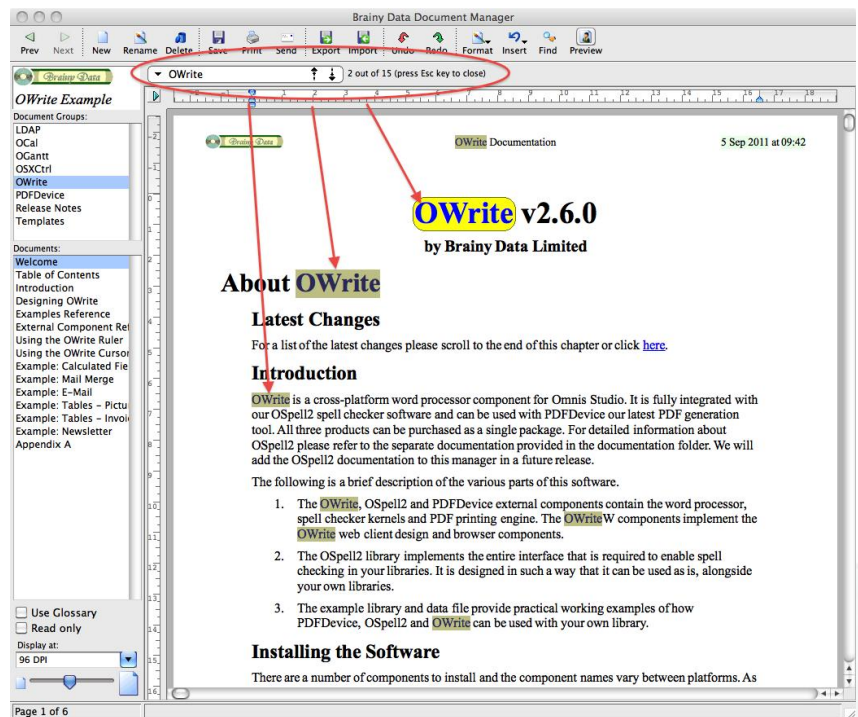
- `FirstSel` - The start of the selection in `kWriSRDefault` range
- `LastSel` - The end of the selection in `kWriSRDefault` range
- `ObjID` - The ident of the object or zero if not inside an object
- `PageNumber` - The page number of the selection
- `RowText` - The plain text of the row (only when using `$getresultlist`, see below)

It is also possible to fetch the result list using the new method `OWriteSearch.$getresultlist(&IList, SelRange, bIncludeRowText)`. Using this method one can specify the `kWriSRxxx` range constant and if to include the plain text of the entire row in which the match was found.

The current line of the returned list will be set to the match that is currently focused. You can move through the matches by calling the method `OWriteObject.$findnext()`, just as before. Alternatively, you can change the current line in the result list and assign the list to `$multiselectlist` or call the method `$setresultlist`.

There are additional properties for the *OWriteSearch* object that allow you to control the appearance of the multi-selection highlight. These are:

- `$multiselectbackcolor` - The background fill colour that is used to highlight all matches other than the current focused match.
- `$multiselecttextcolor` - The text colour that is used to highlight all matches other than the current focused match.
- `$selectbackcolor` - The background fill colour that is used to highlight the current focused match.



- `$selecttextcolor` - The text colour that is used to highlight the current focused match.

The OWrite examples implement a new quick find interface in the main Document Manager window (`wOWrite`). Simply press command-F (`ctrl-F` on windows) when the main edit window is in front and a panel will become visible just above the OWrite ruler. See the new window class `wFindMultiSelect`.

Associated with this feature is the event `evMultiFind` which is generated when OWrite either shows or hides the multi selections. The event parameter `pSelectionShown` is `kTrue`, if the editor shows multiple selections, otherwise it is false. The examples use this event to hide the quick find panel. This event is required as OWrite will automatically hide the multi selection when the user clicks in the editor. `evMultiFind` example...

Plain Text Analyzes

It is sometimes much easier to analyze text content using plain text. Especially when using third party tools that require the provision of plain text. However, when saving an OWrite document as plain text it is virtually impossible to relate the position of a word or sentence in the plain text with the same word or sentence in the rich OWrite content. This is especially true when the document contains complex objects such as tables or text boxes.

In OWrite version 3 we have introduced a feature that allows OWrite to create a plain text map so that when calculating the selection range of a word or sentence using the plain text copy of a document, the same selection range can be used in the OWrite document.

To use this feature, the OWrite document must first be saved as plain text using the new option `kWriTextCreateMap`. After that, the selection range constants `kWriSRPlainText` can be used to tell OWrite to use the plain text map during a call to `$setselection`.

Example:

```
Do OWriteRef.$savedata(kWriFmtText,text_var,kFalse,kWriTextCreateMap,kTrue)
...
Do OWriteRef.$setselection(10,20,0,kWriSRPlainText)
```

When using `kWriSRPlainText` to set the selection, the following errors may occur

```
kWriErrNoTextMap (-14): Attempt to use kWriSRPlainText with no text map
```

This could mean that you have not saved the plain text with the option `kWriTextCreateMap`

```
kWriErrEndOfTextMap (-15): OWrite has reached the end of the text map.
Document content must be out of sync.
```

This could mean that the plain text or rich text content was altered since the last save.

NOTE: The content of headers and footers is not included in the plain text map.

Web Client

The web client implementation of OWrite does not support all the functionality of the fat client version, and some functionality has been implemented in another way. This section explains the limitations and differences.

Limitations

- `$savedata` and `$loaddata` only support plain text, RTF and OWrite binary formats.
- Calculated fields cannot be evaluated on the client. Documents must be evaluated on the server and send to the client in evaluated form.
- Calculations that are entered for calculated fields cannot be validated.
- Spell checking is not supported.
- OWrite static methods are not supported, but many of the OWrite static method have also be implemented as method of the remote form object.
- The following methods are not supported, `$print`, `$spell`, `$findinit`, `$findnext` and `$replace`. Future versions may implement `$findinit`, `$findnext` and `$replace`.

Differences

The web client control primarily uses the instance variable specified by `$dataname` to load and save documents. The methods `$savedata` and `$loaddata` can still be used to save a document to a different format that is supported by the web client control.

Omnis Web Client does not support menus or the pop-up menu command. In order to provide a reasonable interface the OWrite control provides a new method `$popupmenu()` that can be called to pop-up a menu. The example interface demonstrates how this method can be used to implement standard and context menus on the web client.

In the fat-client, measurements must be given in centimetres or inches according to the `$root.$prefs.$usecms` property. This property does not exist on the web client and we now use the OWrite property `$showcms` to evaluate measurements.

The web client does not support string tables. Please see the section on translations in this document.

Designing JS-OWrite

Introduction

This chapter introduces the steps required to add JS-OWrite to your application. If you familiar with OWrite desktop (if you are not, it may be of benefit to quickly read through the chapter “Designing OWrite”), you will have a head start as many of the properties, methods and events are the same in JS-OWrite. However, there are some important differences. Firstly, as with OWrite desktop, you will be able to attach editor properties directly to interface controls, except you will have to use the special property controls provided by JS-OWrite. Secondly, not all OWrite desktop properties, methods and events are supported by JS-OWrite, some work differently and JS-OWrite has added some new methods and properties.

NOTE: Before you continue, make sure you have followed the installation instructions in the “Welcome” chapter, before opening the example library. We also recommend that you read the section related to Advanced Font Handling in the ‘Designing OWrite Chapter’ and refer to \$loaddata and \$loadautosave.

Examples

The most important classes that implement the JS-OWrite functionality can be found in the library’s “OWrite Java Script” folder which implements a JS version of our document manager.

rfOWriteJSDemoFullSize - implements the client interface for the JS OWrite example. Click the class name to test the form or click here to view the code.

rfOWriteSuper - the super-class for all the above demo forms. It implements most of the code that handles the interface and its events.

objOWriteEval - is an object class that implements the merging of document templates with data from the sample database. You will find out more about data merging in the next section.

The various remote forms that start with “rfOWFormat” in their name implement the sub windows used for displaying document details such as current paragraph and style info etc. The controls contained within these forms are special OWrite Property Object controls which can be linked to the various OWrite properties for displaying and manipulating document content.

NOTE: To select calculated fields without running them, make sure the OWrite document has the focus than hold down the CTRL key on Windows or CMD key on Macintosh and click the calculated fields.

Data Merging Examples

The example library provides a number of documents that demonstrate the power of the OWrite data merging feature. These documents can be found in the OWrite group of documents and their name starts with “Example”

The important server based class involved in data merging is the object class objOWriteEval. This class inherits the external non-visual OWrite document object. Consequently, this class inherits all the OWrite properties and methods that are relevant for the programmatic manipulation of documents without the overhead of a visual interface. We recommend the use of

such an object class for implementing all required data merging functionality.

Typical data merging involves loading an OWrite document into the object and merging it with data by assigning \$evalcalcs to kTrue. This evaluates all its embedded calculated fields, pictures and tables. The document is then saved and returned to the client for display and further editing. Calculations assigned to OWrite objects can be any valid Omnis calculation including calls to notation/custom methods as long as they are available within the context of the objOWriteEval instance and return appropriate data for merging.

Browser Compatibility

We cannot guarantee full compatibility across all browsers. Our testing has mainly been carried out using Google Chrome and we recommend this browser for above all others. Other browsers that we have tested and which only display minor issues are Firefox, Safari and Microsoft IE/Edge. There are some major problems with Safari in iOS. We are still investigating possible solutions.

The following desktop browsers have been tested:

On Windows 7 & 10: Microsoft Edge, Internet Explorer v11, Chrome, Opera and Firefox.

On Macintosh: Safari, Chrome, Opera and Firefox.

The following tablets have been tested with some success, but more work is required:

Surface Pro 2017 - Windows 10: Microsoft Edge (works best)

Galaxy Tab S2 - Android 7.0: Firefox (intermittent focusing issues)

PHP Spell Checker (v5.0)

In version 5, we have added PHP based spell checking to jsoWrite. There is a new property called \$spellcheckerurl which is a property of the control and specifies the URL of the PHP script on the server. The URL may provide additional parameters for the PHP script, such as spell checker initialisation options. The jsoWrite document manager examples include a basic PHP script that you can modify for your own requirements. The script's URL must provide a number of parameters for additional functionality such as adding words to a server dictionary, or spell checking in a language other than US/UK English. Below is a list of all the parameters supported by the provided script and jsoWrite kernel:

Known Issues

This section lists issues that we are aware of which do not require reporting unless you feel you may have encountered a variation of the problem description shown here. We are working hard to resolve these a.s.a.p.

1. Performance issue when resizing or moving an image when text has to wrap around it.
2. When clicking calculated field with command held-down the click calculation is executed if the document did not have the focus.

Contents

JS-OWrite Basics - brief introduction to the client and server objects.

Your First Form - set of instructions for designing a basic functional JS-OWrite form

JS-OWrite Quick Reference - short reference of most important properties and methods.



JS-OWrite Basics

Client Controls

The JS-OWrite client implementation consists of two client controls. The JS-OWrite editor-control and the JS-OWrite property-control. The property-control is used to build an interface that directly communicates with the editor-control to manipulate document content without having to write any Omnis code. The property-control has a property called \$property which lists all the editor-control's properties that can be manipulated. Depending on which property the property-control interfaces with, the property-control will create different modes of input on the client. The property-control has additional properties that allow one to manage the appearance, i.e. labels, tool-tips and icons. For a full description please see the section JS-OWrite Quick Reference.

Essentially, these two controls are all that is needed to create a sophisticated client interface for editing OWrite documents.

Server Object

There are a number of things that the client controls cannot do because some tasks are difficult to accomplish in Java script or would increase the size of the script too much or because resources are required that only exist on the server.

For example, the client control cannot create new blank documents. Generally, documents will be stored within a database on the server, so the creation of new documents was considered a server function that can be carried out by the OWrite non-visual document object. We recommend however that one or more new document templates are prepared during development or by your users that can readily be loaded by the client, whenever a new document is required. Document templates could be stored in your database as Omnis lists. See \$savedata() for how to save OWrite documents as Omnis lists. Document templates can contain pre-defined set of document styles (new documents created by the non-visual component will only contain a single default style called 'Normal'). The examples demonstrate how new documents are created using a document template from a database.

Another main server function is to export OWrite documents to alternative formats such as HTML, RTF or to print PDF files. One reason for producing a PDF file from the document being edited on the client may be so the client can print or save the document. This involves sending the document to the server where it is loaded into the OWrite non-visual object for printing to PDF. The resulting PDF file can then be displayed on the client for printing or saving on the client. The JS-OWrite example remote form demonstrates how this is achieved using the Brainy Data PDFDevice which can produce PDF documents in memory and return them to the client without the need to access the hard-disk. Exporting OWrite documents as HTML is useful if documents are to be send as e-mails. Alternatively, documents can be saved as RTF if they are to be opened by other word-processors or editors.

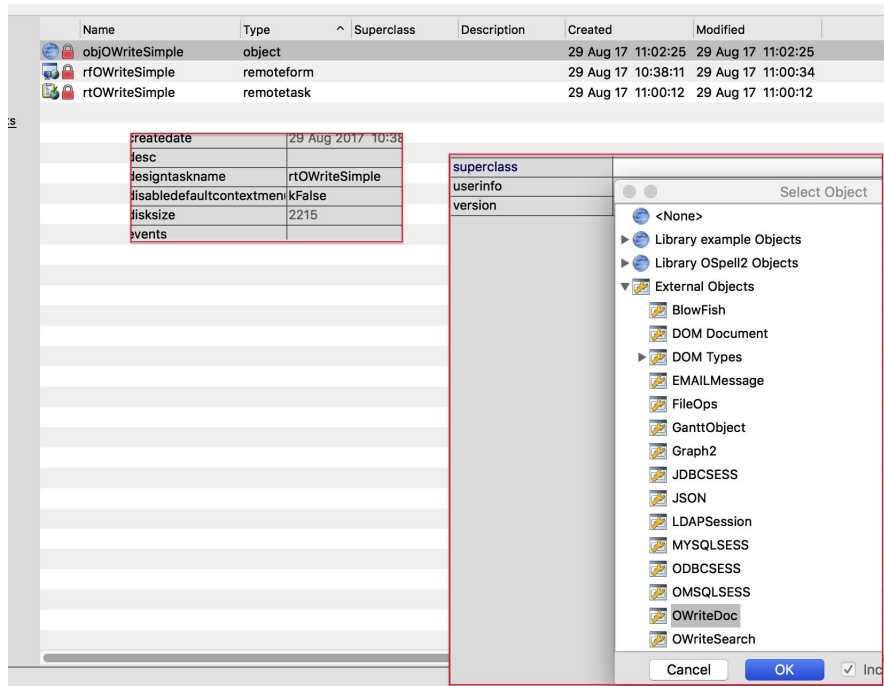
The final main server function is the merging of data. As data is mainly accessible on the server, the document to be merged has to be send to the server and loaded into the OWrite non-visual object to execute the merge. Once data has been merged, the document can be returned to the client, stored in the database or converted to alternative formats as discussed above.

Your First Form

In this section we will create a very basic editor with a few formatting options to demonstrate the minimum steps required for a functional OWrite Form. See OWriteSimple example.

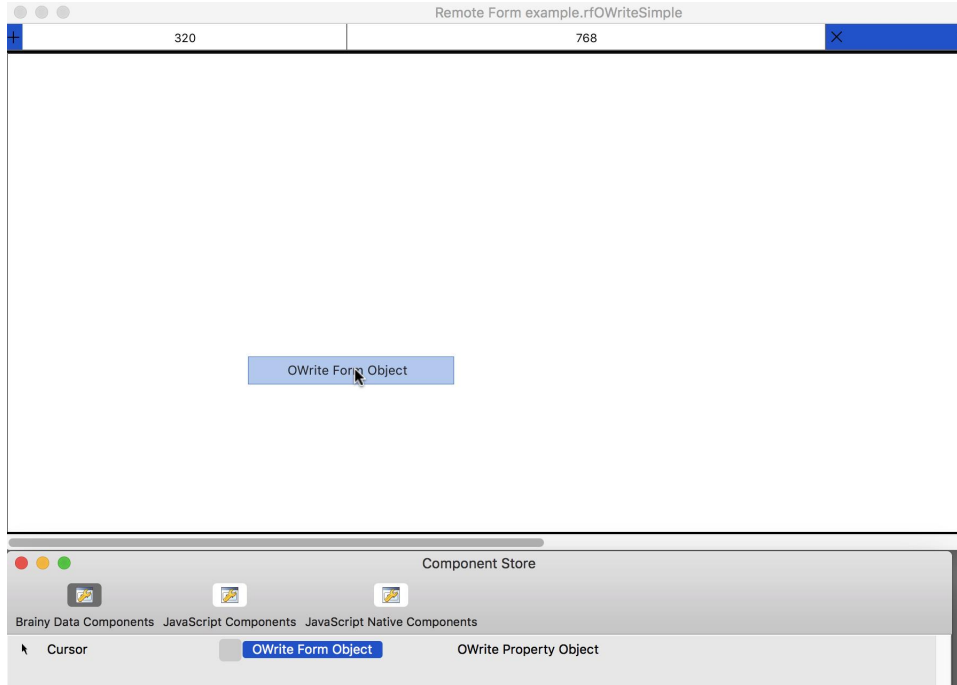
Create the required classes

First create a remote form, a remote task and an object class and name them appropriately. We named ours `rfOWriteSimple`, `rtOWriteSimple` and `objOWriteSimple` respectively. Make sure `$designtaskname` of the remote form is set to `rtOWriteSimple` and that the `objOWriteSimple` class inherits the external OWrite document object.



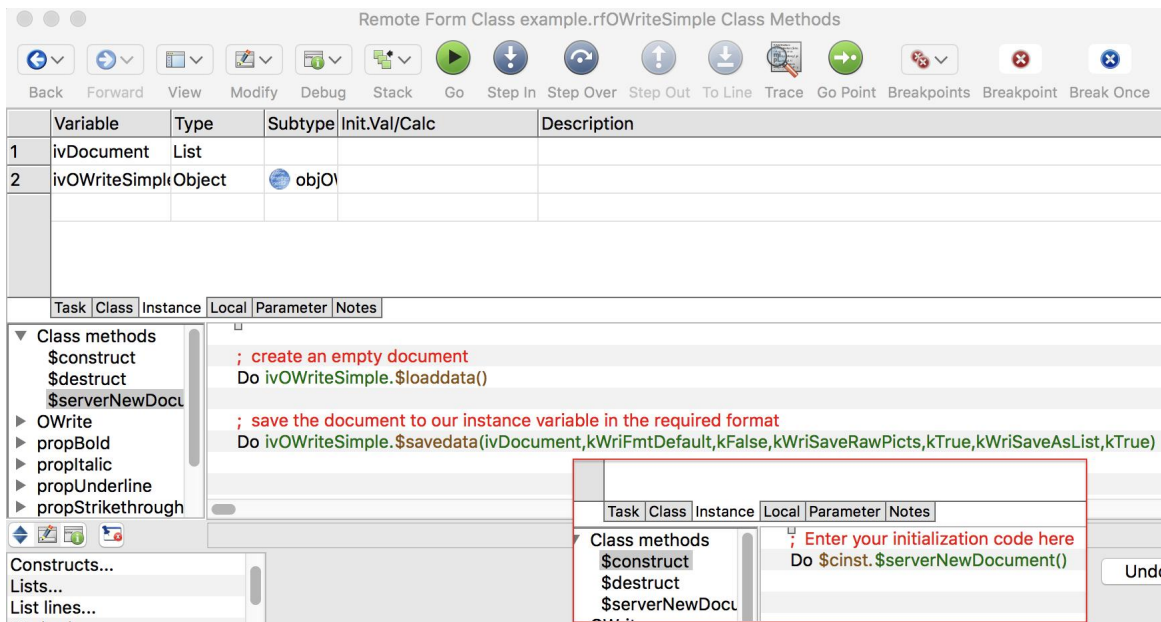
Place the OWrite object

Now modify the remote form and drag the OWrite Form Object from the Component Store onto the remote form. Set the object's name and appearance. We called ours 'OWrite' and set the border to single inset.



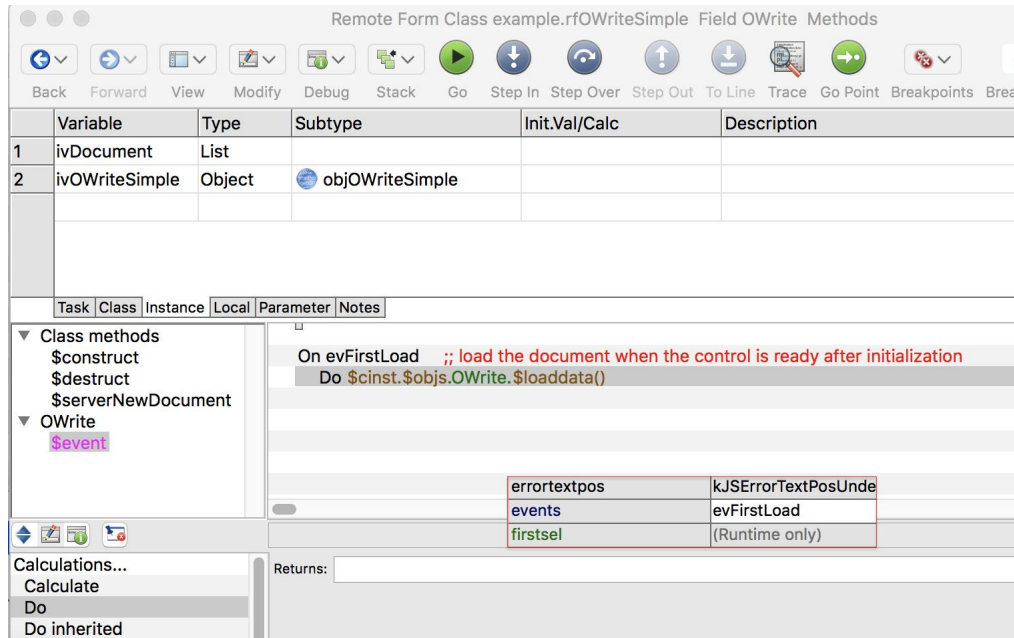
Creating a new document

Code a server method for creating a new document. For this we require two instance variables. One of type list for storing the document data and one of type Object that is an instance of our objOWriteSimple class. This method we must call from \$construct.



We can now set the object's \$dataname to that instance variable and create the client method to load the document. The first document (following construction) cannot be loaded until the evFirstLoad event is received by the OWrite \$event method (\$event must be set to be executed on the client). **Make sure you enable the evFirstLoad event in the OWrite \$events property.**

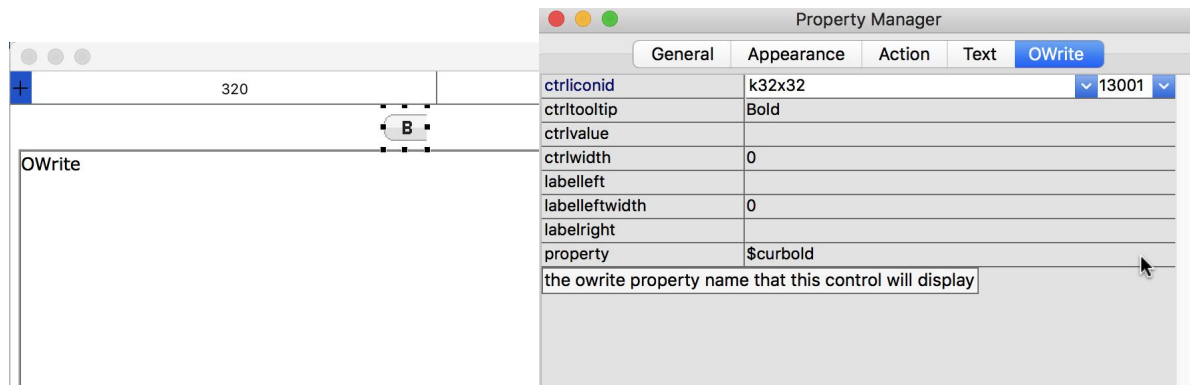
Note: OWrite does not automatically load the data from the instance variable as the developer may need to pass additional information such as auto-save information.



You should now be able to test the form and enter some text.

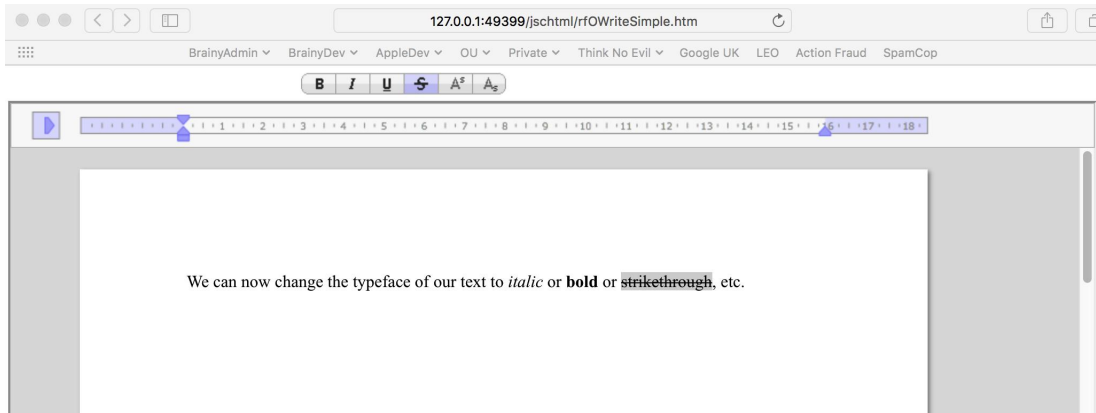
Add text formatting controls

Let us now add some interface controls for changing the text bold, italic and underline states. From the component store drag the OWrite Property Object to the remote form and apply the appropriate icon, tool-tip and property name.

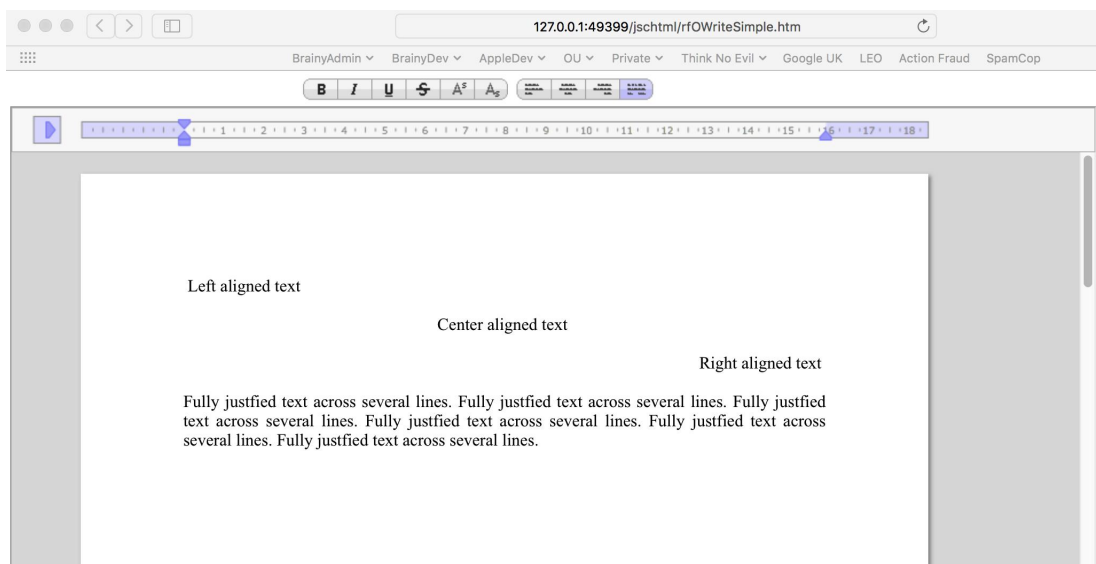
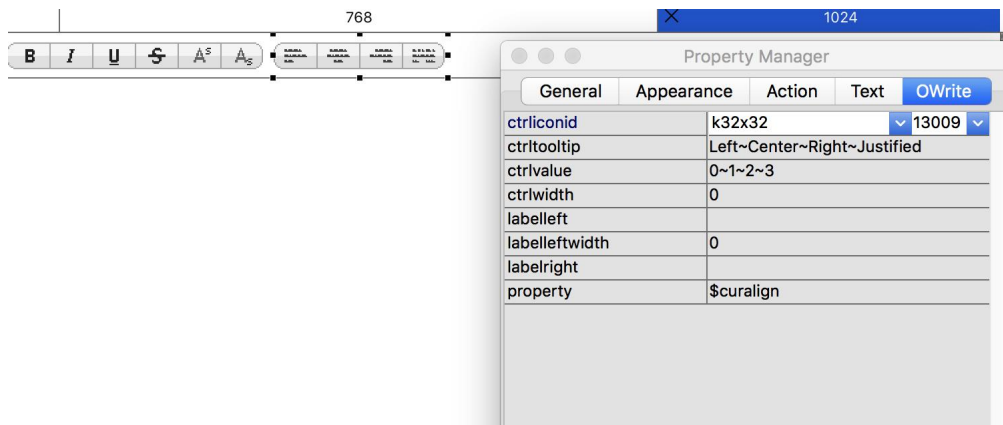


Repeat for italic, underline, strike-through, etc.

We can now change the typeface of our text.



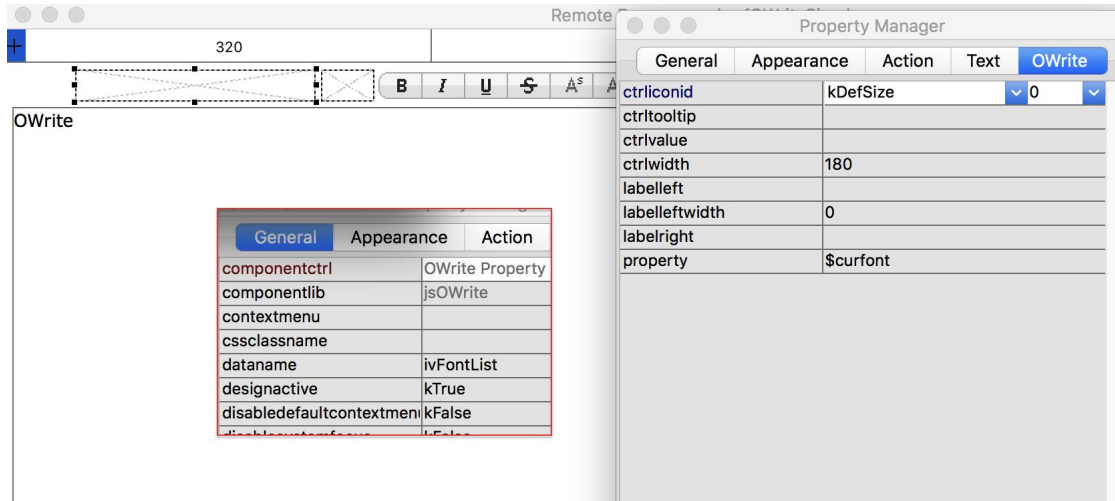
The properties so far were all individual boolean properties that required a property control each (when placed together, our example icons make them appear as a single bar of buttons). Some properties such as alignment properties are enumerated properties with multiple settings. Such properties require a single property object that specifies multiple values, tool-tips and the first icon out of a sequential set.



www.brainydata.com

Add style and font controls

Finally we will add style and font interface controls. Our document style control (\$curstylename) and font size control (\$curfontsize) only require to be linked to the property \$curstylename and have a fixed \$ctrlwidth. OWrite will do the rest. However, our font control requires a list of available fonts that is set as its \$dataname. Thus we also require a server method that builds the list of fonts which is called from \$construct.



Variable	Type	Subtype	Init.Val/Calc	Description
1	ivDocument	List		
2	ivOWriteSimple	Object	objOWriteSimple	
3	ivFontList	List		

```

Task Class Instance Local Parameter Notes
▼ Class methods
  $construct
  $destruct
  $serverNewDocument
  $serverBuildFontList
  OWrite
  ▶ propCurBold
  ▶ propCurItalic
  ▶ propCurUnderline
  ▶ propCurStrikethrough
  ▶ propCurSuperscript
  ▶ propCurSubscript
  ▶ propCurAlign
  ▶ propCurFont
  ▶ propCurfontsize
  ▶ propCurstylename
  ; build list of fonts for the client interface
  ; ;; OWrite can build a list of fonts based on available fonts on the server
  ; ;; However, this list of fonts would not be web-safe
  If not(cvFontList.$linecount) ;; only build once for all clients using class variable
  If not(pBuildWebSaveFonts)
  Do OWrite.$getfontlist(cvFontList,0,0,0,kTrue)
  Else
  ; ;; here we build a list of websave fonts (there are others that can be added)
  ; ;; Note: OWrite can accept multiple font names separate by commas
  Do cvFontList.$define(FontName)
  Do cvFontList.$add('Georgia, serif')
  Do cvFontList.$add('Palatino Linotype, Book Antiqua, Palatino, serif')
  Do cvFontList.$add('Times New Roman, Times, serif')

  Do cvFontList.$add('Arial, Helvetica, sans-serif')
  Do cvFontList.$add('Arial Black, Gadget, sans-serif')
  Do cvFontList.$add('Comic Sans MS, cursive, sans-serif')
  Do cvFontList.$add('Impact, Charcoal, sans-serif')
  Do cvFontList.$add('Lucida Sans Unicode, Lucida Grande, sans-serif')
  Do cvFontList.$add('Tahoma, Geneva, sans-serif')
  Do cvFontList.$add('Trebuchet MS, Helvetica, sans-serif')
  Do cvFontList.$add('Verdana, Geneva, sans-serif')

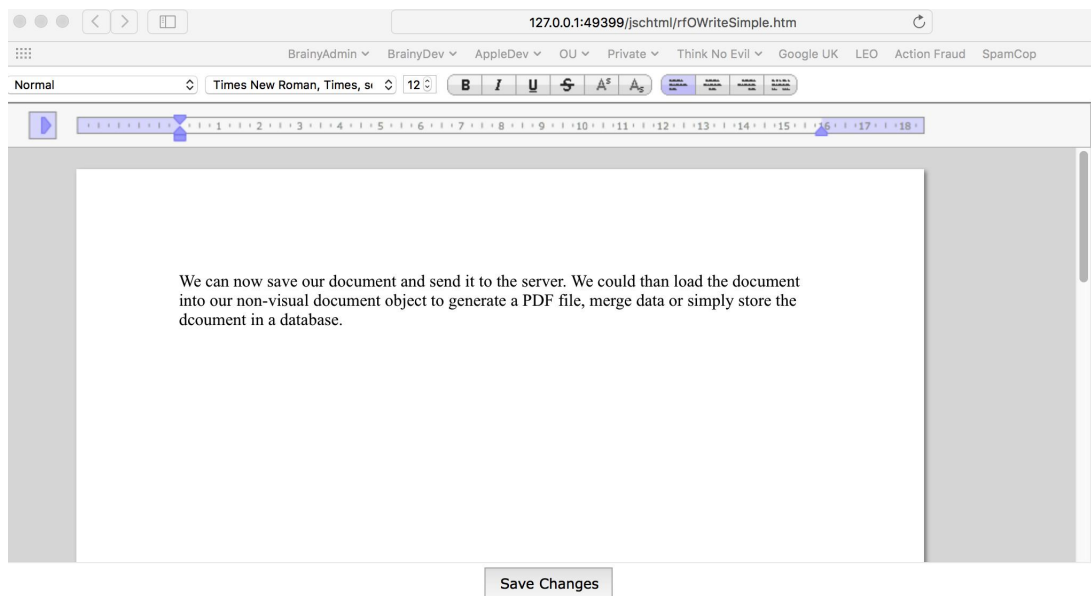
  Do cvFontList.$add('Courier New, Courier, monospace')
  Do cvFontList.$add('Lucida Console, Monaco, monospace')
  End If
  End If
  Calculate ivFontList as cvFontList
    
```

www.brainydata.com

Saving the document

Finally, we must add a control to save our document. For this we use a standard Omnis button control. Make sure to set the \$event method to execute on the client and call \$savedata() on the click event. Our button can be enabled and disabled as the document is modified or saved respectively. When a document is first modified, OWrite will generate the evModified event, at which point we enable the button and when we save the document we disable the button. **Make sure all the required events are enabled in OWrite's Sevents property.**

It is important to note that data is saved asynchronously so any code placed after the \$savedata() call would be executed before the data has actually been saved to the instance variable. OWrite will generate an evSaveData event when the task is complete. At that point it will be possible to call a server method to store the document or do whatever is required in your case. If an error occurred during the save, the event evSaveDataError is generated instead.



Some final words

This section only covered the very basics of implementing JS-OWrite. Please also have a look at some of the additional methods the non-visual document object implements and the additional functionality implemented by the document manager examples.

JS-OWrite Quick Reference

Property-Control Properties

<code>\$property:</code>	the name of the OWrite <code>\$cur...</code> property, this must include the <code>\$</code> sign, i.e. <code>\$curstylename</code>
<code>\$ctrliconid:</code>	the icon id for single state properties such as <code>\$curbold</code> or the first icon ID for multi-choice properties such as <code>\$curalign</code> .
<code>\$ctrltooltip:</code>	the tooltip text for single state properties or tilde separated list of tooltip text for multi-choice properties.
<code>\$ctrlvalue:</code>	typically used for multi-choice properties, this specifies a tilde separated list of values, one for each of the choices that you would like to make available. This list determines how many icons and tooltip text items are required. for example: <code>\$ctrlvalue = "0~1~2~3"</code> <code>\$ctrltooltip = "left~center~right-justified"</code> <code>\$ctrliconid = k32x32 13009 (#ICONS must contain multi state icons with IDs 13009,13010,13011 and 13012 that depict these options)</code>
<code>\$ctrlwidth:</code>	typically used for standard text or number input, this specifies the width of the actual input control in pixels (excluding labels)
<code>\$labelleft:</code>	specifies the label text to be placed to the left of the input control (right justified within <code>\$labelleftwidth</code>).
<code>\$labelleftwidth:</code>	specifies the width of the left label span in pixels. This property can be used to help horizontally align input controls with different length left labels.
<code>\$labelright:</code>	specifies the label text displayed to the right of the input control (typically used for text such as units, i.e 'cm' or 'pts').

Client Methods

The following methods are or will be supported on the client:

`$deleteautosave(autoSaveID)`

Deletes the auto-save data associated with the given ID.

`$getstylelist(outList)`

Returns the list of document styles in the current document.

`$getbookmarks(outList)`

Returns the list of bookmarks in the current document.

`$hasautosave(autoSaveID)`

This method returns the date and time as a string if the specified auto-save exists,

otherwise NULL is returned. This can be useful if the developer wants to compare the date and time with the last successful save prior to calling \$loadautosave.

`$insert(objType,objData,insertPos,asyncId,asyncData)`

This method can be called to insert plain text or objects (tables, text-boxes, pictures, calculated fields and info fields). The web-client version of this method differs from the fat-client version in that all attributes of an object to be inserted must be specified as part of the object data when calling the \$insert method. The following client-only methods have been added to help with building the data required for complex objects such as tables.

`$initparams(objType)`

return a java-script object that can store information for the specified object type.

`$addparams (obj, param, value[, attribute, value, ...])`

adds the attributes to the specified object.

`$addparamstblrow(tbl, rowNum, attribute, value[, attribute, value, ...])`

assigns the specified attributes to a table row in a table object.

`$addparamstblcell(tbl, rowNum, cellNum, attribute, value[, attribute, value, ...])`

assigns the specified attributes to a table cell in a table object.

`$addparamstblcellcontent(tbl, rowNum, cellNum, text[, attribute, value, ...])`

assigns text content to the specified cell in a table object. Additional parameters can be specified for setting style information.

Please refer to the examples for further details about inserting objects.

Note: This is an asynchronous operation, so the operation will not have been completed by the time the function returns control to the Omnis code. When the operation has been completed, the evAsyncDone event is generated.

`objType` the object type. One of the kWriObjType... constants.

`objData` the object data as described above

`insertPos` either kWriInsertOver which will replace the current selection, or kWriInsertAfter which will insert after the current selection

`asyncId` passed through to evAsyncDone on completion

`asyncData` passed through to evAsyncDone on completion

`$interfacevisibilitychanged(subFormName)`

When using the JS-OWrite property controls to provide an interactive interface for the editor, the controls can be placed on different sub-forms for different purposes. It is assumed that only one sub-form is shown at a time in which case this method can be called to limit updates to the controls shown in the specified sub-form (no need to redraw controls that are currently not required).

`$loadautosave(autoSaveID,autoSaveMinutes,userPrompt,fontSizes)`

This method will load the specified auto-save data if it exists. The method returns kTrue if the auto-save data was loaded, but if a prompt was specified the client may decline the loading of the auto-save data in which case the method returns kFalse.

autoSaveID	Identifies the auto-save data to be loaded.
autoSaveMinutes	This optional parameter specifies the auto-save interval in minutes. A copy of the data is also stored locally when calling \$savedata in case the server save fails, which may require the client to reconnect. (default is 5 minutes)
userPrompt	The text to be presented to the client if auto-save data exists. JS-OWrite will insert the date and time of the auto save if the text contains a '\$' character.
fontSizes (v4.1)	The list of font sizes returned by the server OWrite method \$getfontlist. If provided JS-OWrite will use the provided font sizes to measure font heights so they are compatible with the fonts on the Omnis server.

\$loaddata(autoSaveID,autoSaveMinutes,theData,fontSizes,asyncId,asyncData)

This method will load the document that is stored in the instance variable that is assigned to the JS-OWrite control's \$dataname property. The client does not support loading of empty document data it is therefore recommended to prepare template documents using the fat-client OWrite control that can be loaded when new documents are required. These templates may contain a set of prepared document styles. Note that loading data is asynchronous process thus when control is returned from calling \$loaddata the data is unlikely to have been loaded. JS-OWrite will generate the evFormatChanged event when all data is loaded.

autoSaveID	This optional parameter can specify an id that must be unique for the document to be edited. This ID is used for storing local auto-save data. Server connections can be fragile and the developer can query the existence of auto saved data for a document when the client re-connects and call \$loadautosave instead. Note: auto-save data must be deleted when a document has been saved successfully.
autoSaveMinutes	This optional parameter specifies the auto-save interval in minutes. A copy of the data is also stored locally when calling \$savedata in case the server save fails, which may require the client to reconnect. (default is 5 minutes)
theData	This optional parameter specified the data to be loaded. If theData is not specified, data is loaded from the variable specified by \$dataname.
fontSizes (v4.1)	The list of font sizes returned by the server OWrite method \$getfontlist. If provided JS-OWrite will use the provided font sizes to measure font heights so they are compatible with the fonts on the Omnis server.
asyncId (v4.3)	passed through to evAsyncDone on completion
asyncData (v4.3)	passed through to evAsyncDone on completion

\$loadstrings()

Loads translated strings from the remote form's string table. Currently only string 7100 for the Page Break text is supported.

\$performautosave()

This method can be called by the developer to perform a local auto-save prior to entering a risky operation such as contacting the server which can sometimes result in the loss of the server connection, or rather the client realising that the server connection is lost and locking out the client from further interaction with the form.

\$reformat()

Reformats the entire document by performing an internal save and reload. This can be useful when some formatting has gone wrong during complex paste or editing actions.

\$savedata()

Call this method to save data to the instance variable that is assigned to the JS-OWrite control's \$dataname property. Saving data is an asynchronous process which means \$savedata will return before the data has been saved. JS-OWrite will generate a evSaveData event once the save-data operation is completed. At this point a server method should be called to submit the data to a database. When calling server methods, Omnis will call a method on the client when the server method completes execution. This method must have the identical name appended with "_return". When the "_return" method is called, the auto-save data should be deleted.

\$search(searchCriteria,asyncId,asyncData)

Searches for the specified text, class or bookmark, and selects the item.

Note: This is an asynchronous operation, so the operation will not have been completed by the time the function returns control to the Omnis code. When the operation has been completed, the evAsyncDone event is generated.

searchCriteria specifies the search type ("text", "class" or "bookmark") and the value to be searched for, separated by a colon. For example, the search criteria "class:Heading 1" will select the next paragraph where the paragraph style is set to "Heading 1".

asyncId passed through to evAsyncDone on completion

asyncData passed through to evAsyncDone on completion

\$setselection(selStart,selEnd,asyncId,asyncData)

Sets the specified selection. This function allows one to synchronize selection ranges between server OWrite objects and the JS-OWrite object on the client. The selection range consists of two dot notation strings that identify the document object/paragraph selection by use of an object type code and order number within the document. Nested objects are identified by a sequence of type codes and order numbers separated by dots. A valid selection range can be retrieved by calling \$getselection(...,kWriSRJSDefault) for the OWrite object on the server.

Note: This is an asynchronous operation, so the operation will not have been completed by the time the function returns control to the Omnis code. When the operation has been completed, the evAsyncDone event is generated.

selStart	the starting selection address
selEnd	the end selection address
asyncId	passed through to evAsyncDone on completion
asyncData	passed through to evAsyncDone on completion

\$tableaction(action,execute,asyncId,asyncData)

Tests or executes the specified table action.

Note: Executing the action is an asynchronous operation, so the operation will not have been completed by the time the function returns control to the Omnis code. When the operation has been completed, the evAsyncDone event is generated.

action	the table action to perform. One of the kWriTblAct... constants.
execute	specify kTrue to execute immediately, or kFalse to simply test if the specified operation is allowed in the current context.
asyncId	passed through to evAsyncDone on completion
asyncData	passed through to evAsyncDone on completion

\$updatecontrols(property,needsRebuild)

This method can be called to redraw the control that displays the specified property. If the control requires a complete rebuild, pass kTrue for the second parameter.

Examples Reference

This reference serves as a guide to the classes that are provided by the OWrite example. It gives a brief description of each class. More information can be found in the classes' \$desc property and the many comments in the classes' code.

Contents

Main Examples - describes the document manager and label writer example windows.

Formatting Interface - describes window classes related to the formatting of OWrite document content.

Insert Windows - describes windows related to inserting of document content.

Other Windows - describes windows related to miscellaneous OWrite features such as find and replace, font mapping, document info, exporting, sending mail, etc.

The Menus - describes the menu classes that are used by the various window classes.

The Objects - describes the object classes that are used by the various window classes.

Other classes - describes various classes used by the window classes, such as schemas and tables, tool bars and reports.

JS-Client Classes - describes all the classes that implement the JavaScript examples.

Main Examples

This group describes the main OWrite example windows that are available from the examples menu.

wOWrite

Implements the OWrite document manager example. This interface is also used to manage the Brainy Data component documentation. It is the most complex of the examples and it may be beneficial to investigate some simpler examples first. Brainy Data has produced a library called OWriteSimple.lbs that includes a much simplified implementation of an OWrite control with the full use of the spell checker.

The document manager interface utilises many of the classes in the example library for formatting, evaluating, printing, mailing etc. Some of these classes will be described later.

wOWriteLables

A fully functioning label printing interface using OWrite tables. This example has been designed so it can be easily adapted into your library with a minimum of work. The window utilises a table and schema class for storing label templates. The table class' \$dowork method has been stubbed to allow you to implement your own DB storage of label templates without having to search through pages of code in the window. The OWrite example folder includes a small set of Avery label templates that can be imported via the interface by dropping them onto the OWrite control, or by clicking the import button. These templates are MS Word files that were downloaded from the Avery web site. All that we did is convert them to RTF using MS Word, so OWrite can read them.

To print labels using this interface you simply provide a list of data and a pick list. The table class implements two methods that build sample data and pick lists.

Formatting Interface

The following classes are used to format styles, paragraphs and text of an OWrite document. Most of these are used by both wOWrite window. Nearly all of the windows that start with wFormat... in their name have a super class called wFormatSuper. This class provides common functionality for the formatting windows, so that the individual formatting windows can concentrate on simply loading and saving properties to and from the windows fields.



wFormatBorderBackground

Used for changing table border and background options.



wFormatBullets

Used for setting paragraph bullet and numbering options. This window can be opened from the format menu to format the selected paragraphs. It is also used by wFormatStyle as a sub window for formatting bullet and numbering options of a style.



wFormatDocLink

This window is not derived from wFormatSuper. It is used when inserting a link to another part of the same document or to another document. Links in OWrite documents are just calculated fields that execute Omnis notation when the field is clicked. The notation can be anything as long as your window implements it. The document manager example inserts `$inst.$openDocument(...parameters...)`. You will find a method in the wOWrite named `$openDocument` that is called when the field is clicked.



wFormatField

Used for formatting an OWrite calculated field. Such a field can be given an Omnis calculation or notation that returns a result. When a document is evaluated, the field is exchanged with the result of your calculation. It is valid for a calculation to return RTF or text.



wFormatFont

Used for formatting font properties of the current selection. This window can be opened from the format menu to format the selected text. It is also use by wFormatStyle as a sub window for formatting font properties of a style.



wFormatPaper

This window provides an interface for changing the document paper size and margins. This window can be opened from the format menu.



wFormatParagraph

Used for formatting paragraph properties such as line spacing and tabs. This window can be opened from the format menu to format the selected paragraph(s). It is also used by wFormatStyle as a sub window for formatting paragraph properties of a style.



wFormatPicture

This window provides an interface for formatting a picture object. The same window is used to format static pictures and calculated picture fields. It has data tab that is hidden or shown appropriately. This window can be opened from the format menu to format the selected picture object.



wFormatStyle

This window is not derived from wFormatSuper. This window provides an interface for adding, removing and formatting all the styles in the current document. It has a function to remove all unused styles. It uses wFormatBullets, wFormatFont and wFormatParagraph as sub windows to format the styles' properties. This is done with the aid of OWrite by setting the \$editstyle name property of the current document. When this property is set all formatting notation is directed against that style instead of the current selection. This window can be opened from the format menu.



wFormatSuper

This is the super class for all the standard formatting windows. It provides methods for construction, loading and saving of properties, and methods for testing changes. All that is required of sub classes is to implement \$initialize, \$load and \$save methods for handling the properties appropriate for this window.



wFormatTable

This window provides a sophisticated interface for formatting tables, selected table rows or columns and selected cells. It has a pick list that allows the user to apply changes in different modes, i.e. current selection, entire table, etc. From this window, the user can also open the wFormatBorderBackground window. This window can be opened from the format menu to format the selected table object.



wFormatTextbox

This window is used to format a text box. Text boxes have properties such as their size, borders and background, floating properties and calculations. A text box can be calculated just like a calculated field. This window can be opened from the format menu to format the current text box object.



wOWriteTools

This window does not derive from wFormatSuper. It implements the main examples formatting palette. This is not an easy window to implement or maintain because of its floating nature of the field groups. But the final appearance and use is well worth the effort.



wOWriteToolsFields

This window does not derive from wFormatSuper. It is used as sub windows by wOWriteTools, and provides an interface for inserting database and other fields into an OWrite document.

 **wOWriteToolsHF**

Version: 3.0

This is a po-pup utility window that is opened when editing a page header or footer. It implements buttons for inserting page numbers, page count, date and time objects, as well as navigation buttons to advancing through the various headers and footers of the document.

Insert Windows

There are a number of windows that aid with the inserting of more complex document objects such as tables, and custom fields.

wOWriteInsertCustomField

It is possible for users to create their own custom fields by selecting some content in a document and choosing “Insert Custom Field” from the insert menu. This content may include calculated fields. This makes it possible to combine several calculated fields into a new custom field and that may include some additional formatted text. The current OWrite interface adds this new field to the Custom group of the Other Fields tab in the formatting palette. The examples do not save these back to a database, so they are lost when you quit Omnis. We leave this up to you, the developer. This window allows the user to name the custom field and give it a description.

wOWriteInsertDocument

This window aids with the inserting of a document link. It lists all the documents in the current group and provides additional fields for specifying the text for a heading or a search. The window builds the notation `$inst.$openDocument(...parameters...)` which it assigns to the calculated field.

wOWriteInsertTable

This window implements an interface to insert a new table object. It provides fields for specifying the row and column count, as well as a list for choosing data from the sample database for calculated tables. OWrite tables can be associated with an Omnis list, and the table's cells can be linked to columns in the list. When a document is evaluated, the table will display the list's content.

Other Windows

There are a number of additional windows that provide features such as find and replace, document info., a document-object editor and more

wFindReplace

This is a fairly sophisticated find and replace interface for searching the current document. As well as searching documents for simple text, styles can be selected to be included in or excluded from a search. For replace actions, styles can also be specified allowing the user to find and replace styles such as bold and italic with just bold, etc. The window uses two object variables that are derived from the OWrite search object to specify the style information. One for the find and one for the replace. This window also uses the windows wFormatFont, wFormatParagraph, wFormatBullets and wFindStyle, for specifying the find and replace options. This window can be opened from the find button in the window toolbar.

wFindStyle

This window is only used by wFindReplace to specify a style name in a find and replace action. It allows the user to search for styles and replace them with another style.

wFontMapping

OWrite allows you to map font names to improve the cross-platform appearance of fonts within a document. This may not be required for many of today's true-type font, but some fonts may still require some user intervention, especially if OWrite is used on Linux. OWrite has a static method called \$loadfontmap() that can be picked from the Methods tab of the Omnis Catalogue. The examples include a basic font mapping table in a disk file. This window is used to edit this table.

wOWriteDocInfo

This window displays document information such as word count, words per sentence, etc. This window can be opened from the context menu.

wOWriteExport

This is a fairly sophisticated interface for exporting one or more documents as OWrite format, RTF, HTML and plain text. The user can choose to export just the current document, or select several documents from the current group. When exporting to HTML, document links are converted to HTML links and, as long as all the documents that are referred to are exported, the links will work very well.

wOWriteLabelDetails

This is a small utility window that is used by the wOWriteLabels window for editing the details of a label. This window can be opened from the context menu in the label example window.

wOWriteMail

This is a simple example that demonstrates how an OWrite document can be converted to HTML and TEXT mail content, including images, and sent using the Omnis SMTPSend command. The window provides an interface for the user to enter the mail server address, recipient and sender address and subject. This window can be opened from the Send Mail button in the document manager window's toolbar.

wOWritePrint

This window provides an interface for printing the current document, or two or more documents from the same group to various destinations. For the purpose of a complete example, you can choose to print using an Omnis report, or the OWrite \$print method. This window only implements Screen, Printer and PDF as the choices for the destination, but theoretically, all Omnis destinations can be printed to. This window can be opened from the Print button in the toolbar of the example windows.

wOWriteSnapShot

This window demonstrates how one can create a bitmap based picture from an OWrite document page. It is useful if one wants to automate the creation of thumb images for document templates. When users create new documents in your application they could be presented with a list of these thumb images. This window can be opened from the OWrite context menu.

wOWriteTestSelection

This window was created for our benefit, to thoroughly test the \$setselection and \$getselection methods. It is a useful example that demonstrates the more sophisticated capabilities of these two methods. This window can be opened from the OWrite context menu.

wOWriteViewDocObjects

This window demonstrates the potential use of the \$getobjslst method. It provides an interface for viewing and manipulating all the objects in the current document.

The Menus

There are a small number of menus that are installed in the main menu bar or are used as context or pop-up menus in some of the windows.



mFormat

This menu is used by wOWrite and wHeaderFooter for the format button in the windows toolbar.



mFormatContext

This menu is used by wHeaderFooter as the main OWrite context menu.



mFormatFind

This menu is used by the wFindReplace window. It opens the various formatting windows for specifying the formatting options for the find and replace.



mInsert

This menu is used by wOWrite as the insert menu for the insert button in the toolbar.



mInsertHF

This is the main insert menu for the wHeaderFooter example window.



mOWrite

This menu is installed as a sub menu on the examples menu in the main menu bar.

The Objects

Three object classes are used throughout the examples, for tasks such as printing, searching and spell checking.



oOWrite

This object can be used for manipulating, printing and merging of OWrite documents. It derives from the external OWriteDoc object. It is actually not used by any of the examples, but could be used as your main NV object to which you can add additional methods for managing documents without the need of a window.



oOWriteSearch

This object is used by wFindReplace. It derives from the external OWriteSearch object. Its main purpose is to specify search and replace criteria. For a find and replace action, two of these objects are required.



oOWriteSession

This object is used by the wOWrite and wHeaderFooter example windows. It derives from the OSpell2 external object SpllSession. Its main purpose is to provide a spell check session for an OWrite window control. Each window control requires its own session during loading and editing of documents. In addition, the OSpell2 library requires the OWrite window control to provide a notation method called \$spellsession that returns an item reference to this object.

Other classes

There are a small number of additional support classes that are used throughout the examples.

rOWrite

This prints a single OWrite document with headers and footers provided by the reports header and footer sections. It is used by wOWritePrint and rfOWrite for the web-client document preview.

rOWriteMailMerge

This documentation includes a mail merge example document. The document has an active link that will run a mail merge which uses this report.

sOWriteLabels

This schema class is used by the label examples. It defines the label template list. You can adapt this class to store label templates in your own database. The wOWriteLabels window is the only class that requires it.

tOWriteLabels

This table class implements functionality for managing label template records. For the purpose of the label examples, the \$dowork method has been stubbed so it can be easily adapted for storing label templates in your own database. The wOWriteLabels window is the only class that requires it.

tbOWrite

This is the main toolbar class for the wOWrite window.

tbOWriteLabels

This is the main toolbar class for the wOWriteLabels window.

JS-Client Classes

This section lists the OWrite JavaScript client example classes. They consist of the main document form and forms for formatting the document and additional classes for evaluating and printing documents on the server.

Note: All rfOWFormat.... classes subclass rfOWFormatSuper.



rfOWriteSuper

The main web-client super class. This window implements most of the common client and server functionality for the document manager sub-class.



rfOWriteJSDemoFullSize

The main web-client form in full size. This window implements a js-client version of the document manager. It can be used to view, edit and print documents.



rfOWFormatSuper

This is the super class for all the remote formatting forms. It provides some common functionality for loading and saving properties.



rfOWFormatDocument

Remote form for formatting document properties such as page size, orientation, margins, etc.



rfOWFormatField

This form is used to edit/format calculated fields.



rfOWFormatInfo

This form is used to edit/format info fields (page count, page number, date, time, etc).



rfOWFormatNone

This form is displayed when no object is selected.



rfOWFormatParagraph

Remote form for formatting paragraph attributes.



rfOWFormatPicture

Remote form for editing the attributes of the selected picture object.



rfOWFormatTable

This form displays the table, table row and table cell properties



rfOWFormatText

This form is used to set the font options for the selected text in the document.



rfOWFormatTextbox

Remote form for formatting the current text box object.



rfOWOptions

Remote form for general example behaviour.



objOWriteEval

Object used for evaluating documents on the server. It directly inherits from the OWrite non-visual external object OWriteDoc, providing it with all the functionality for loading, evaluating and saving documents.



rmOWriteContext

Main remote context menu.



rmInsertObject

Remote context sub-menu for inserting items, such as pictures, fields and page breaks.



rmTableOptions

Remote context sub-menu for table object actions such as inserting and deleting rows or cells.



rtOWriteJS

Main OWrite document manager remote task.







rtGetPDF

Remote task used for ultra-thin requests to download the PDF file that was produced for the current document.

External Component Reference

Introduction

This chapter lists all the OWrite constants, properties, methods and events that are available within desktop and/or server implementations. The xcomp OWrite component library supplies the following external objects

-  Window Object for editing and viewing OWrite documents in an Omnis desktop window.
-  Report object for printing OWrite documents in an Omnis report.
-  Non-visual object “OWriteDoc” for manipulating, searching and printing OWrite documents without the need for a window.
-  Non-visual object “OWriteSearch” for searching OWrite documents for text and/or styles.

Each object provides its own set of properties, methods, constants and events.

Contents

Constants - OWrite provides a large number of constants that are used with OWrite properties or methods. The constants are organised into functional groups and can be accessed from the Omnis Catalogue.

Static Methods - Static methods are not associated with any OWrite object and can be used anywhere in an Omnis method. You can select static methods from Functions tab in the Omnis Catalogue.

Note: Static methods are not supported in methods that are executed in the web client. However some static methods are also implemented as methods of the OWrite form/window/non-visual object thus making them available for use on web clients. Some methods such as the \$getfontlist() may differ in behaviour when used with OWrite object instances.

Window/NV Object Properties - The window/non-visual object properties control the behaviour and appearance of the objects and document content. All listed properties are common between window, remote form and non-visual OWrite objects.

Note: Properties shown in red are read-only and cannot be assigned.

Window/NV Object Methods - In general the object methods listed in this table apply to OWrite window objects, OWrite remote form objects and OWrite non-visual objects. Some methods may not exist on web-client (see individual method descriptions).

Window Object Events - The events listed in this table are generated for both the OWrite window and remote form objects. Events are usually generated as a result of user actions.

Report Object Properties - The OWrite report object can be used in Omnis reports for printing OWrite documents. The properties listed in this table are used to specify the OWrite data and printing options.

The example library implements a report that prints OWrite documents with page headers, footers and page numbers.

OWriteSearch - NV Object Properties - The OWriteSearch object's main function is to record format settings and text for find and replace. The object implements a small subset of properties taken from the OWrite document objects that relate to formatting as well as properties related to search options.

OWriteSearch - NV Object methods - The OWriteSearch object implements methods related to multi-selection searches.

Constants

kWriBord...

Constants for use with the \$scurobjborderstyle property.

Name	Value	Description
kWriBordNone	0	Text box has no border.
kWriBordLine	1	Single line border.
kWriBordLine2	2	Double line border of equal width.
kWriBordLine2TT	3	Double line border with thick and thin lines.
kWriBordLine2TTR	4	Reversed double line border with thin and thick lines.
kWriBordLine3	5	Three line border with thin, thick, and thin lines.

kWriChar...

Special character constants for use with \$::insert().

Name	Value	Description
kWriCharPageBreak	3	Page break character.
kWriCharTab	9	Tab character.
kWriCharLine (v2.0)	10	Line break.
kWriCharReturn	13	Paragraph break.

kWriErr...

Constants returned by OWrite method calls.

Name	Value	Description
kWriErrBadData	-2	Returned by \$loaddata() and \$mergedocs() when the specified data is of an unrecognised format. Returned by \$loadstrings() when the specified table has not been loaded.
kWriErrBadMethod	-12	Indicates an internal error. Contact Brainy Data.
kWriErrBadParams	-1	Returned by all methods when the incorrect number or type of parameters have been passed to a method.
kWriErrBadVersion	-6	Returned by \$loaddata() when the OWrite data is of a newer version than the installed component can load.

kWriErrInsert	-4	Returned by \$::insert() when the method failed. This could be because the format and data type where not compatible or another unknown error occurred.
kWriErrInvalidSelRange	-11	Returned by \$setselection() when the mode is kWriSRAddress and the specified addresses where not valid.
kWriErrNone	0	No error.
kWriErrPrint	-3	Returned by \$print() when a print manager error occurred.
kWriErrSpell	-5	Returned by \$spell() when an invalid action was specified. For valid actions see kWriSppl...
kWriErrStyleExists	-7	Returned by \$addstyle() when attempting to add a style name that already exists.
kWriErrStyleInUse	-10	Returned by \$removestyle() when attempting to remove a style that is in use.
kWriErrStyleNotFound	-8	Returned by \$addstyle() if the specified based on style does not exist, and by \$removestyle() if the style is not found.
kWriErrStyleTooMany	-9	Returned by \$addstyle() when the maximum number of styles (256) has been reached.
kWriErrWarning (v2.4.1)	-13	Returned by some methods when a non-fatal warning error has occurred. The property \$docwarnings will return a list of warnings.
kWriErrNoTextMap (v3.0)	-14	Attempt to use kWriSRPlainText with no text map. See also Plain Text Analyzes.
kWriErrEndOfTextMap (v3.0)	-15	OWrite has reached the end of the text map. Document content may be out of sync. See also Plain Text Analyzes.
kWriErrNotMultiSelect (v3.0)	-16	Search object has not been initialised with \$multiselection enabled. See also Multi-Selection Find.
kWriFmt...		
Constants for use with the \$savedata() and \$loaddata() methods for specifying the format of the data.		
Name	Value	Description

kWriFmtDefault	0	OWrite binary format. Must use binary field for storage.
kWriFmtText	1	Plain text format. When exporting to a text file on disk you must use a binary field for intermediate storage. From version 5.0 onwards, the additional option kWriOutputHF can be used to output a single header and footer at the beginning and end of the text document, i.e. ...kWriFmtText,kTrue,kWriOutputHF,kTrue)
kWriFmtRTF	2	Rich text format. When exporting to a RTF file on disk you must use a binary field for intermediate storage.
kWriFmtHTML (v1.50)	3	HTML format. This option can only be used with \$savedata() (Fat-client only). When exporting to a HTML file on disk you must use a binary field for intermediate storage. From version 5.0 onwards, the additional option kWriOutputHF can be used to output a single header and footer at the beginning and end of the text document, i.e. ...kWriFmtHTML,kTrue,kWriOutputHF,kTrue)
kWriFmtUnassigned (v2.2.2)	255	Used with \$dataname to disable the use of \$dataname.

kWriFrame...

Constants for use with \$curobjframeoptions. These constants can be used to turn off edges of an objects border, and specify special border edging for table cells. These constants do not override the border style and other border settings, they merely specify on which edges the border is drawn.

Name	Value	Description
kWriFrameNone	0	Draw no border.
kWriFrameLeft	1	Draw the left border.
kWriFrameTop	2	Draw the top border.
kWriFrameRight	4	Draw the right border.
kWriFrameBottom	8	Draw the bottom border.

The constants that follow specify special framing modes for table fields. If these values are specified, they will override the above constants and the table will calculate the border edges that are to be drawn.

kWriFrameCustom	0	Draw custom edge arrangement as specified by left/top/right/bottom above. No special framing mode.
kWriFrameBox	16	Draw box around selected cells, no lines in between.
kWriFrameCols	32	Draw column lines for all selected cells, and top lines for the top most selected cells and bottom lines for the bottom most selected cells.
kWriFrameRows	48	Draw row lines for all selected cells and left lines for the left most selected cells and right lines for right most selected cells.
kWriFrameGrid	64	Draw borders on all four sides (same as left+top+right+bottom)

kWriGrammar...

Version: 5.4

Scope: Control

Set of constants for assigning the property \$grammaroptions that specifies which grammar options are to be applied when editing documents.

Name	Value	Description
kWriGrammarNone	0	Grammar options are disabled
kWriGrammarCapFstWrd	1	Capitalize the first word of a sentence
kWriGrammarDelDblSpace	2	Remove double spaces when deleting selected text

kWriHtml...

Version: 1.50

Custom export parameter constants for use with \$savedata() when saving to HTML.

Name	Value	Description
kWriHtmlNoMargins	1	If specified as kTrue, the page margins are ignored.
kWriHtmlSizeAdjust	2	If specified as kTrue, font sizes are scaled down by a factor of 72 over 96. This is to counteract the effect of browsers displaying fonts at a rate of 96 DPI. This parameter is only meaningful when used on the Macintosh platform to create on screen likeness. It will however distort likeness when the html file is printed.

kWriHtmlOmnisCompatible	3	FOR FUTURE USE. NOT FULLY IMPLEMENTED.
kWriHtmlBookmarks	4	This parameter allows you to specify a comma separated list of style names for generating HTML bookmarks within your document. It gives you the capability to link together HTML documents produced by OWrite.
kWriHtmlRawSupport (v2.1.2)	7	When executing \$savedata() with kWriFmtHtml, and the custom parameter kWriHtmlRawSupport is specified with kTrue, OWrite will export the content as RAW html if the content starts with the text "<HTML".
kWriHtmlNoAutoSize (v2.6.0)	12	When specified with kTrue, the HTML produced will use the DIV tag to prevent document content from resizing horizontally to fit the width of the browser window.
kWriHtmlBgColor (v2.6.0)	13	This constant can be used to specify the background colour for the HTML document.
kWriHtmlTitle (v3.0.0)	14	Used to specify the text for the html <title> tag.
kWriInsert... (v4.5) Constants for use with the method \$docinsert(). These constants are flags and can be specified in combination.		
Name	Value	Description
kWriInsertOver	0	insert will replace current selection. (currently only applies to jsOWrite)
kWriInsertAfter	1	content will be inserted after current selection. (currently only applies to jsOWrite)
kWriInsertKeepStyles	2	if specified this will keep the styles of the current selection and apply it to the inserted text
kWriInsertSelect	4	inserted text will be selected following the insert. This replaces the original kTrue value for this parameter, although existing code which passes kFalse or kTrue will behave as before.
kWriInsertApplyMaxImageSize (v5.1)	8	apply the \$maximagesize property when inserting pictures.

kWriLine...		
Constants for use with the \$curobjlinestyle property.		
Name	Value	Description
kWriLineSolid	0	Solid line.
kWriLineSolid	1	Dotted line.
kWriLineDash	2	Line with dashes.
kWriLineDashDot	3	Line with repeating dashes and dots.
kWriLineDashDotDot	4	Line with repeating dashes and two dots.
kWriLineLongDash	5	Line with repeating long dashes.
kWriLineLongDashDot	6	Line with repeating long dashes and dots.
kWriLineLongDashDotDot	7	Line with repeating long dashes and two dots.
kWriList...		
Constants for use with the \$curlisttype property.		
Name	Value	Description
kWriListNone	0	Paragraph has no bullets and is not numbered.
kWriListBullet	1	Paragraph has bullets.
kWriListDecimalD	2	Paragraph is numbered using decimal numbers and terminated with a dot.
kWriListDecimalP	3	Paragraph is numbered using decimal numbers and terminated with a parenthesis.
kWriListAlphaUD	4	Paragraph is numbered using upper case alpha characters and terminated with a dot.
kWriListAlphaUP	5	Paragraph is numbered using upper case alpha characters and terminated with a parenthesis.
kWriListAlphaLD	6	Paragraph is numbered using lower case alpha characters and terminated with a dot.
kWriListAlphaLP	7	Paragraph is numbered using lower case alpha characters and terminated with a parenthesis.
kWriListRomanUD	8	Paragraph is numbered using upper case Roman numerals and terminated with a dot.
kWriListRomanLD	9	Paragraph is numbered using lower case Roman numerals and terminated with a dot.

kWriLoad... Custom parameter constants for use with \$loaddata() when importing or loading data.		
Name	Value	Description
kWriLoadRawPicts (v3.5)	16	If specified as a custom parameter with \$loaddata() during import (currently only RTF), PNG and JPEG images are not converted to Omnis pictures and are stored as raw PNG or JPEG images with the document. Example: <pre>...\$loaddata (rtf, kWriFmtRtf, kFalse, kWriLoadRawPicts, kTrue)</pre>
kWriLoadMetaQuality (v4.5)	18	This parameter can be used to specify a multiplication factor (valid values are between 1 and 8) to increase the pixel width and height that is used to convert a windows meta file to a pixel based image. This should improve the final quality of the bitmapped image to some extent. WARNING: using this feature will increase the overall image size by the specified factor squared. For example a meta-file with an intended resolution of 100 by 50 pixels will be scaled to 400 by 200 pixels if a factor of 4 is specified. This results in an image size of 80,000 pixels (400x200), which is 16 times larger than the 5,000 pixels (100x50) if no factor had been specified.
kWriMenuItem... (v2.0) Constants for use with the flags column of menu lists. See \$popupmenu().		
Name	Value	Description
kWriMenuItemNone	0	Menu item is disabled
kWriMenuItemEnabled	1	Menu item is enabled
kWriMenuItemChecked	2	Menu item is checked
kWriMenuItemEdit... (v3.0) Constants for use with the ID column of menu lists. See \$popupmenu().		
Name	Value	Description
kWriMenuItemEditUndo	1	Menu item ID for the edit menu 'Undo' action

kWriMenuItemEditRedo	2	Menu item ID for the edit menu 'Redo' action
kWriMenuItemEditCut	3	Menu item ID for the edit menu 'Cut' action
kWriMenuItemEditCopy	4	Menu item ID for the edit menu 'Copy' action
kWriMenuItemEditPaste	5	Menu item ID for the edit menu 'Paste' action
kWriMenuItemEditClear	6	Menu item ID for the edit menu 'Clear' action
kWriMenuItemEditSelectAll	7	Menu item ID for the edit menu 'Select All' action
kWriMerge... (v3.5.0) Constants for use with \$mergedocs.		
Name	Value	Description
kWriMergePageBreak	1	seperate the two documents with a page break
kWriMergeNoPaginate	2	prevent pagination after merge to save time. IMPORTANT: only valid action after one or more calls to \$mergedocs is a call to \$savedata
kWriMergeUseExistingStyles	4	ignores differences in identical named styles and always uses styles from first document
kWriObjAlign... Constants for use with the \$curobjalign property for specifying the alignment of an in-line object with the surrounding text.		
Name	Value	Description
kWriObjAlignBottom	0	The bottom of the object is aligned with the bottom of the text.
kWriObjAlignBase	1	The bottom of the object is aligned with the baseline of the text.
kWriObjAlignTop	2	The top of the object is aligned with the top of the text.
kWriObjAlignCenter	4	The object is centered with the text.
kWriObjFmt... Constants for use with the \$curobjfmt property for specifying the formatting mode of an object.		
Name	Value	Description
kWriObjFmtInline	0	Object is in-line with text.
kWriObjFmtBehind	1	Object is positioned behind the text
kWriObjFmtInfront	2	Object is positioned in front of the text

kWriObjFmtWrap	3	Text is wrapped around the object
kWriObjFmtSplit	4	Surrounding text is split horizontally.
kWriObjType...		
Constants for use with the \$::insert() for specifying the object type. Also returned by the \$scurobjtype property.		
Name	Value	Description
kWriObjTypeNone	0	No valid object (no object selected).
kWriObjTypeDoc	1	Document object (currently not used).
kWriObjTypePara	2	Paragraph object (currently not used).
kWriObjTypePict	3	Picture object.
kWriObjTypeCalc	4	Calculated field
kWriObjTypeText	5	NOT IMPLEMENTED
kWriObjTypeCalcPict	6	Calculated picture
kWriObjTypeTextbox	7	Text box
kWriObjTypeChar	8	Character object for inserting special characters. See kWriChar...
kWriObjTypeTable (v2.0)	9	Table object. Constant can be used for inserting a table object.
kWriObjTypeTableRow (v2.0)	10	Table row object (currently not used).
kWriObjTypeTableCell (v2.0)	11	Table cell object.
kWriObjTypeHeadFoot (v3.0)	12	The object type of a header or footer object.
kWriObjTypeInfo (v3.0)	13	Document info object such as page number or count. You can insert info objects of the following sub types. See also Header and Footer Object types.
kWriObjTypeInfo... (v3.0)		
Subtype constants for use with the kWriObjTypeInfo object. These constants are used with the \$docinsert() method, i.e. \$docinsert(kWriObjTypeInfo,kWriObjTypeInfoDate)		
See also Header and Footer Object types.		
Name	Value	Description

kWriObjTypeInfoPgCnt	1	Displays the total page count.
kWriObjTypeInfoPgNum	2	Displays the page number.
kWriObjTypeInfoDate	3	Displays the current short date (formatting specified by #FD)
kWriObjTypeInfoTime	4	Displays the current short time (formatting specified by #FT)
kWriObjTypeInfoPgCntFP (v4.5)	5	Displays the total page count excluding first page
kWriObjTypeInfoPgNumFP (v4.5)	6	Displays the page number excluding the first page
kWriObjTypeInfoPgCntLR (v4.5)	7	Displays the total page count using lower-case roman numerals
kWriObjTypeInfoPgCntUR (v4.5)	8	Displays the total page count using upper-case roman numerals
kWriObjTypeInfoPgNumLR (v4.5)	9	Displays the page number using lower-case roman numerals
kWriObjTypeInfoPgNumUR (v4.5)	10	Displays the page number using upper-case roman numerals

kWriOutput...

Additional output options when exporting documents to different formats

See also kWriFmt...

Name	Value	Description
kWriOutputHF (v5.0)	19	Used when exporting documents to single page output formats (currently plain text and html). When this parameter is specified as kTrue, oWrite will export a single default header and a single default footer.

DISPLAY BEHAVIOUR:

When exporting to HTML, OWrite will attempt to make some intelligent positioning choices based on the tabs that are used in the header or footer.

TAB BEHAVIOUR:

Traditionally, the OWrite HTML export translates tabs to three or four simple spaces. This is because HTML does not support tab positions within in-line text flow. However, within headers and footers, OWrite creates separate text spans that take advantage of the HTML flex-box behaviour to simulate left-center-right tab behaviour that are typically used within headers and footers. This will produce a reasonable approximation of the aforementioned tabs. Tab combinations OWrite will look out for are:-

- headers and footers with a center and right tab
this will result in three flex boxes being used (left,center,right aligned)
- headers and footers with just a right tab
this will result in two flex boxes being used (left and right aligned)

LIMITATIONS:

Because only a single header and footer is exported, OWrite info fields which display the current page number or document page count will be nonsensical and should not be used. A typical use for a fixed footer in a html document may be some advisory note or other relevant information such as copyright info, etc.

kWriOverflow...

(v3.6.5)

Constants send as part of the evOverflow event. The constants indicate what action caused the overflow

See also \$checkoverflow and evOverflow.

Name	Value	Description
kWriOverflowChar	1	Overflow was caused by user typing.
kWriOverflowPaste	2	Overflow was caused by the user pasting content.
kWriOverflowInsert	3	Overflow was caused by a programmed call to \$::insert.
kWriOverflowLoad	4	Overflow was caused by a call to \$loaddata.
kWriOverflowReplace	5	Overflow was caused by a spell-checker replace action.

kWriPaste...

(v5.1.0)

The paste options to be used with \$pasteoptions. These constant values are binary based and thus can be added together to specify a set of options.

The constants a set constants and therefore must never be used by adding constant values to the existing value of the property, i.e. DO NOT DO

```
Calculate owrite.$pasteoptions as
    owrite.$pasteoptions+kWriPasteStripEmptyLineNBSP
```

If kWriPasteStripEmptyLineNBSP has already been applied to the property it will accumulate. Instead you CAN DO

```
Calculate owrite.$pasteoptions as
    bitor(owrite.$pasteoptions, kWriPasteStripEmptyLineNBSP)
```

Name	Value	Description
kWriPastePlainNone	0	Use this constant on its own to disable pasting into oWrite.

kWriPastePlainText	1	Allow pasting of plain text
kWriPasteRichText	2	Allow RTF and images as part of RTF if kWriPasteImages is also specified
kWriPasteImages	4	Allow pasting of images direct from clipboard or as part of RTF if kWriPasteRichText is also specified
kWriPasteStripEmptyLineNBSP	8	Strip spaces and non-breaking spaces from otherwise empty lines during a paste
kWriPasteRTFnoParaStyles	16	When pasting RTF, document and paragraph styles are ignored.
kWriProgress... (v3.5.0) Specifies the type of action being performed during progress event messages. See evProgress.		
Name	Value	Description
kWriProgressLoad	1	progress during loading of documents
kWriProgressSave	2	progress during saving of documents
kWriProgressImport	3	progress during exporting of document
kWriSave... Custom export parameter constants for use with \$savedata().		
Name	Value	Description
kWriSaveNonUnicode (v2.2)	8	Can be used when saving to HTML or Plain text. When used with HTML, the HTML is encoded using the ISO-8859-1 character set. With plain text, the operating system character set is used.
kWriSaveSelection (v1.62)	5	If set to kTrue, only the current selection is saved.
kWriScrDPI... Constants for use with the \$screendpi property. Other values can be specified. The valid range is 36 to 192.		
Name	Value	Description
kWriScrDPIDefault	0	Use default DPI for platform (72 on Macintosh and 96 on Windows and Linux).
kWriScrDPI72	72	Render documents at 72 DPI on screen regardless of platform.

kWriScrDPI96	96	Render documents at 96 DPI on screen regardless of platform.
kWriSelect... (v2.0) Constants for use with the \$setselection(), \$getselection and \$convselection().		
Name	Value	Description
kWriSelectCurrent	-1	Constant that specifies the current selection when used for the iSelStart, iSelEnd or iSelObjID parameters of \$setselection() and \$convselection().
kWriSelectStart	-2	Constant that specifies the beginning of the document, text box or table cell when used with the iSelStart or iSelEnd parameters of \$setselection() and \$convselection().
kWriSelectEnd	-3	Constant that specifies the end of the document, text box or table cell when used with the iSelStart or iSelEnd parameters of \$setselection() and \$convselection().
kWriSelectHeadEvenDef (v3.0)	-20	The object ID for the default page header or page header for even pages if \$headfootoddeven is true. See also Headers & Footers.
kWriSelectFootEvenDef (v3.0)	-19	The object ID for the default page footer or page footer for even pages if \$headfootoddeven is true. See also Headers & Footers.
kWriSelectHeadOdd (v3.0)	-18	The object ID for the page header for odd pages if \$headfootoddeven is true. See also Headers & Footers.
kWriSelectFootOdd (v3.0)	-17	The object ID for the page footer for odd pages if \$headfootoddeven is true. See also Headers & Footers.
kWriSelectHeadFirst (v3.0)	-16	The object ID for the page header for the first page if \$headfootfirstpage is true. See also Headers & Footers.
kWriSelectFootFirst (v3.0)	-15	The object ID for the page footer for the first page if \$headfootfirstpage is true. See also Headers & Footers.

kWriSelectHeadFoodNext (v3.0)	-4	The object ID for the next page header. Useful for moving through all headers and footers sequentially. See also Headers & Footers.
kWriSelectHeadFoodPrev (v3.0)	-5	The object ID for the previous page header. Useful for moving through all headers and footers sequentially. See also Headers & Footers.
kWriSpIlL...		
Constants for use with the \$spell() method. (Fat-client only)		
Name	Value	Description
kWriSpIlLCancel	1	Cancel the interactive spell check.
kWriSpIlLCheckSelection	2	Check selected text or all text if no text is selected.
kWriSpIlLCheckAll	3	Check all text.
kWriSpIlLIgnoreWord	4	Ignore the selected word.
kWriSpIlLIgnoreAll	5	Ignore all occurrences of the selected word.
kWriSpIlLInvalSelection (v2.2.2)	6	Forces OWrite to spell-check the current selection in the background.
kWriSpIlLInvalAll (v2.2.2)	7	Forces OWrite to spell-check the entire document in the background.
kWriSR...		
(v2.0)		
Constants for use with the \$getselection(), \$setselection() and \$convselection().		
Name	Value	Description
kWriSRDefault	0	Standard selection range, characters including white space and object place holders.
kWriSRChars	1	Selection range that only counts non-white space characters.
kWriSRWords	2	Selection range that counts words.
kWriSRRows	3	Selection range that counts paragraph and table rows.
kWriSRSentences	4	Selection range that counts sentences. Empty paragraphs do not return a sentence count.
kWriSRParagraphs	5	Selection range that counts paragraphs.

kWriSRPages	6	Selection range that counts the number of intersecting pages.
kWriSRAddress	7	<p>The constant specifies a dot notation address that does not rely on runtime information to re-create a path to the selection that is specified. In other words, the selection values returned remain valid between closing and opening a document. A selection address may contain 2 or more parts separated by a dot.</p> <p>It is an internal persistent addressing format that allows the exact locating of a cursor position within any OWrite object or the document.</p> <p>Example: \$getselection may return the two strings "2.25" and "4.25". These two strings specify that the characters between position 2 and 4 of the object starting at overall position 25 are selected.</p> <p>Note: Modifying the document may invalidate the address.</p>
kWriSRMouse	8	This constant can only be used with \$getselection(). It returns the standard selection range (kWriSRDefault) under the mouse pointer.
kWriSRPlainText (v3.0)	9	<p>If specified, the selection range will use the plain text map that was created during the last save with kWriTextCreateMap.</p> <p>See also Plain Text Analyzes.</p>
kWriSRBookmark (v3.0)	10	<p>Selection range for specifying a bookmark name in parameter 1, parameter 2 and 3 are ignored.</p> <p>See also Bookmarks.</p>

<p>kWriTblAct... (v2.0) Constants for use with the \$tableaction() method.</p>		
Name	Value	Description
kWriTblActInsertRowBelow	1	Insert a row below the selected cells.
kWriTblActInsertRowAbove	2	Insert a row above the selected cells.
kWriTblActInsertColRight	3	Insert a column to the right of the selected cells.
kWriTblActInsertColLeft	4	Insert a column to the left of the selected cells.
kWriTblActInsertCellRight	5	Insert a cell to the right of the selected cells.

www.brainydata.com

kWriTblActInsertCellLeft	6	Insert a cell to the left of the selected cells
kWriTblActDeleteCells	7	Delete the selected cells.
kWriTblActMergeCells	8	Merge the selected cells. The new cell will take on the content and width of all the selected cells. Note: It is currently not possible to merge cells that belong to different rows. This feature will be added in a future release.
kWriTblActSplitCellsHorz	9	Split the selected cell horizontally. The two new cells will be of equal width occupying no more space than the original cell.
kWriTblActSplitCellsVert	10	CURRENTLY NOT SUPPORTED
kWriTblActAlignLeft	11	Align the left edge of all selected cells.
kWriTblActAlignRight	12	Align the right edge of all selected cells.
kWriTblActPrevRow	13	Select the row above the current selected row.
kWriTblActNextRow	14	Select the row below the current selected row.
kWriTblActPrevCol	15	Select the column to the left of the current column.
kWriTblActNextCol	16	Select the column to the right of the current column.
kWriTblActPrevCell	17	Select the cell to the left of the current cell.
kWriTblActNextCell	18	Select the cell to the right of the current cell.
kWriTblActAboveCell	19	Select the cell above the current cell.
kWriTblActBelowCell	20	Select the cell below the current cell.
kWriTblApply... (v2.0) Constants for use with \$curtblapplymode. These constants change the way table and cell properties are read and assigned.		
Name	Value	Description
kWriTblApplyCells	0	All property assignments and reads are applied to the selected cells. This is the default behavior.
kWriTblApplyRows	1	All property assignments and reads are applied to entire table rows that contain selected cells.
kWriTblApplyCols	2	All property assignments and reads are applied to entire table columns that contain selected cells.

kWriTblApplyTable	3	All property assignments and reads are applied to the entire table regardless of selection.
kWriTblRow... (v2.0) Constants for use with \$curtblrowtype. These constants specify the type of the selected row(s).		
Name	Value	Description
kWriTblRowNormal	0	Default row type.
kWriTblRowHeader	1	Header row type. Header rows are repeated at the top of each page that the table occupies. Header rows are useful for displaying column headers and other header information.
kWriTblRowFooter	2	Footer row type. Footer rows are repeated at the bottom of each page that the table occupies. Footer rows are useful for displaying column totals and other footer information.
kWriText... Constants for use with the \$savedata() and \$loaddata() method when loading or saving plain text.		
Name	Value	Description
kWriTextFmtUTF16 (v2.2.0)	9	If specified, text is exported/imported as 16bit unicode characters. This format includes a leading BOM character.
kWriTextFmtUTF8 (v2.2.0)	10	If specified, text is exported/imported as 8bit unicode characters.
kWriTextSaveSpecialChars	6	If specified, special chars such as internal object place holders, are not stripped when exporting to plain text
kWriTextCreateMap (v3.0)	11	If true, OWrite will create a plain text map for use with kWriSRPlainText. See also Plain Text Analyzes.
kWriView... Version: 1.61 Constants for use with the \$pageview property.		
Name	Value	Description
kWriViewNormal	0	Display document in normal view.
kWriViewPageLayout	1	Display document in page layout view.

kWriViewField	2	Display document in field view. Text is wrapped to the width of the field.
kWriWarn... Version: 2.4.1 Constants specified in a list of warnings returned by \$docwarnings.		
Name	Value	Description
kWriWarnBadObjSize	1	Specified when a OWrite object was found with a negative width or height. The object will have been restored to the original width and height if possible. The column pData1 will specify the objects ident (\$curobjid). This can be used to select the object.
kWriWarnNone	0	No warning (not used)



Static Methods

\$disablescreenupdates()

Syntax: `OWrite.$disablescreenupdates()`
 or `OWriteObjectRef.$disablescreenupdates()`

Version/Platform: 2.1.2

This method is used together with `$enablescreenupdates()`. It disables screen updates to allow Omnis methods to do large amounts of changes to OWrite documents without flashy redraws on screen. These methods effect OWrite controls on all open windows.

Parameter	Description
returns	0 (no error) if successful.

\$docversion()

Syntax: `OWrite.$docversion([xDocData])`
 or `OWriteObjectRef.$docversion([xDocData])`

Version/Platform: v3.0

This method returns the document version number for the specified binary document data.

Parameter	Description
xDocData	The binary OWrite document data for which the version number is returned. If this parameter is omitted, OWrite will return the current version number.

returns the internal document version number which may be one of the following.

Document Version	OWrite Version
1	v1b2
2	v1.0
3	v1.6
4	v1.7
6	v2.0
7	v2.2
8	v3a7
9	v3a8
10	v3a10
11	v3a11 or better

As can be seen from the table, the document version number is not the same as the OWrite product version number.

www.brainydata.com

\$enablescreenupdates()

Syntax: `OWrite.$enablescreenupdates()`
 or `OWriteObjectRef.$enablescreenupdates()`

Version/Platform: 2.1.2

This method is used together with `$disablescreenupdates` which must be called first and must always be perfectly balanced. It enables screen updates after they have been disabled by a call to `$disablescreenupdates()`. It allows Omnis methods to do large amounts of changes to OWrite documents without flashy redraws on screen. These methods can be used with all Omnis windows.

Parameter	Description
returns	0 (no error) if successful.

\$getfontlist()

Syntax: `OWrite.$getfontlist(&IList,cCurFont,bCurBold,bCurItalic,bApplyMap,bSortInUse)`
 or `OWriteObjectRef`

`.$getfontlist(&IList,cCurFont,bCurBold,bCurItalic,bApplyMap,bSortInUse)`

Returns a list of font family names and supported typefaces as provided by the system. The specified Omnis font name and bold/italic styles will select the appropriate lines in the list.

Note: When this method is called using an OWrite object instance, the list will show all font family names that are in use in the current document, at the beginning of the list and separated by an empty line from the remaining family names.

See also: Advanced Font Handling.

Parameter	Description
IFamilyList	<p>The list variable that is to receive the list of family names. The returned list consists of three columns.</p> <ul style="list-style-type: none"> - Family: The family name - Typefaces: Sub-list of supported typeface names - Used: Boolean indicating if the font is used in the document <p>The typeface sub-list for each font family consists of four columns.</p> <ul style="list-style-type: none"> - Typeface: The typeface name, i.e. Bold - OmnisFontName: The Omnis equivalent font name - OmnisStyleBold: The Omnis bold style - OmnisStyleItalic: The Omnis italic style <p>When a user picks a typeface, the Omnis font name, bold style and italic style settings should be used to set <code>\$curfont</code>, <code>\$curbold</code> and <code>\$curitalic</code>.</p>
cCurFont	The Omnis font name setting (<code>\$curfont</code>) to be used to map the current selection to the appropriate family and typeface.

bCurBold	The Omnis bold style setting (\$curbold) to be used to map the current selection to the appropriate family and typeface.
bCurItalic	The Omnis italic style setting (\$curitalic) to be used to map the current selection to the appropriate family and typeface.
bApplyMap	If true, Omnis will limit the fonts to the fonts specified in the OWrite Font Map that was previously loaded by calling \$loadfontmap().
bSortInUse (v3.0.3)	If true (default), the returned list is sorted according to if the font is in use. Fonts in use are separated by an empty line from fonts that are not in use within an OWrite document.
lSizeList (v4.0.1)	Returns list font heights for downloading to the JS-Client. IMPORTANT: It is recommended that the OWrite font-mapping feature is used to limit the fonts that are available on the client. Please see the section on Advanced Font Handling for further details.
returns	0 (no error) if successful.
\$listfrombin() Syntax: OWrite.\$listfrombin(bBinaryListData) or <i>OWriteObjectRef</i> .\$listfrombin(bBinaryListData) This method converts a List in binary format back to a list and returns the result.	
Parameter	Description
bBinaryListData	The binary representation of a list. See OWrite.\$listtobinary.
returns	An Omnis list.
\$listtobin() Syntax: OWrite.\$listtobin(ICollection) or <i>OWriteObjectRef</i> .\$listtobin(ICollection) This method converts an Omnis List into binary data suitable for storing in a binary file on disk.	
Parameter	Description
ICollection	The Omnis list to be converted
returns	Binary representation of an Omnis list.

\$loadfontmap()

Syntax: `OWrite.$loadfontmap(IFontmap[,bRestrictFonts])`
 or `OWriteObjectRef.$loadfontmap(IFontmap[,bRestrictFonts])`

This method should be called once during startup to load the font mapping table. When used with the web-client, the font-mapping table can be loaded once into memory on the server, but the table data must be copied to every client via an instance variable where it can be loaded using the OWrite remote form control. The font map is used to map fonts that do not exist on a platform to fonts that do, and to limit the fonts that can be used.

See also [Advanced Font Handling](#) and [The OWrite-Font-Map](#).

Parameter	Description
IFontmap	An Omnis list containing variable number of columns for mapping unknown font names between systems.
bRestrictFonts (v3.0)	If true, the font map will limit the fonts that can be used with OWrite to the fonts that are specified by the font map.

\$loadstrings()

Syntax: `OWrite.$loadstrings(cTableName)`

You call this method if you need to translate OWrite into another language. You must load the Omnis string table and set the appropriate column prior to calling this method. When finished, you may unload the string table.

Parameter	Description
cTableName	Name of the table as specified when calling <code>StringTable.\$loadstringtable()</code> .

\$mergedocs()

Syntax: `OWrite.$mergedocs(Data1,Data2,iMergeOptions,cPropertyList)`
 or `OWriteObjectRef.$mergedocs(Data1,Data2,bNewPage,cPropertyList)`

This method merges the binary data of two OWrite documents and returns the result.

Parameter	Description
-----------	-------------

Data1	Binary data of first document. (v2.12) When used with the window field or non-visual object, this parameter can be #NULL and the document data passed in the second parameter is merged with the loaded document. Example: <pre>Do OWriteObj.\$loaddata (doc1) Do OWriteObj.\$mergedocs (#NULL, doc2) Do OWriteObj.\$mergedocs (#NULL, doc3) ;; etc Do OWriteObj.\$savedata (mergedDocData)</pre> When calling \$mergedocs in this way, instead of returning the merged document data, OWrite will return 1 if successful.
Data2	Binary data of the document to be added to the end of the first document.
iMergeOptions (v3.5.0)	Specifies one or more of the kWriMerge... constants. Multiple options are specified by adding them together. Example: <pre>...\$mergedocs (doc1, doc2, kWriMergePageBreak+kWriMergeNoPaginate)</pre>
cPropertyList (v1.62)	Comma separated list of properties that are to be merged from the second document, i.e. “\$docpaper,\$docpaperlength,\$docpaperwidth”. Only properties that are listed in the Custom tab of the property inspector and are document properties can be merged. Properties that begin with \$cur or are not stored with the document and cannot be merged.
<p>\$pictconvto()</p> <p>Syntax: OWrite.\$pictconvto(SourceFormat,SourceData,DestFormat[,ConvDPI or IconID, FillColor])</p> <p>This method implements image conversion that works very much like the standard Omnis function pictconvto(). In addition to the functionality the standard Omnis function provides, OWrite’s \$pictconvto can handle the conversion of non-shared picture formats such as Mac PICTs and Windows enhanced meta files and Icons from #ICONS or the icon data files. When converting images received from OWrite you should always use this function to ensure that images will be converted correctly, regardless of source type.</p> <p>Please note: It is not possible to convert Mac image types on Windows and vice versa.</p>	
Parameter	Description
SourceFormat	The source format of the image to be converted, i.e. “CS24”. Use the standard Omnis function pictformat() to find out the format of the image.

SourceData	<p>The picture data.</p> <p>When SourceFormat is “ICON”, the supplied variable is used as temporary storage during conversion.</p>
DestFormat	This can be any of the picture formats supported by Omnis.
ConvDPI or IconID	<p>ConvDPI is used when converting Mac PICTs and Windows meta files. These image types can contain drawing commands that are usually embedded at screen resolution, but can scale to higher resolutions without loss of quality and sharpness. Generally this quality is lost when Omnis converts these images to another format as Omnis uses the images default resolution. By specifying a higher resolution OWrite will be able to retain the images quality at the given DPI. However, the images memory requirement will increase.</p> <p>IconID ^(v2.0) must be specified if SourceFormat is “ICON”. The specified Icon will be converted to a picture of the requested format. The IconID can include the Omnis Icon Size constants.</p>
FillColor	If SourceFormat is “ICON”, the fill color is used to set all transparent pixels of the icon.
<p>Example:</p> <pre> ;convert an OWrite image to JPEG ;get the image data from the current OWrite object Calculate sourceData as ivEdit.\$curobjresult ;get the images format Calculate sourceFormat as pictformat(sourceData) ;now convert the image Calculate destData as OWrite.\$pictconvto (sourceFormat, sourceData, "JPEG", 150) </pre>	
<p>\$popupmenulist()</p> <p>Syntax: OWrite.\$popupmenulist(iHwnd,lMenuList,bBelowControl,iHorzAdjust,iVertAdjust)</p> <p>Version: 1.63</p> <p>This method is identical to \$popupmenu except it does not require an OWrite window control.</p>	
Parameter	Description
iHwnd	The \$framehwnd or \$hwnd value of any Omnis window or window field.
<p><i>for all remaining parameters please see the method \$popupmenu() below</i></p>	

Window/NV Object Properties		
Name	Type	Description
\$backcolor (v2.0)	Integer	Standard Omnis field property for specifying the color of all clear pixels in the background pattern of the OWrite gray area around a document page. See also \$forecolor and \$backpattern.
\$backpattern	Integer	Standard Omnis field property for specifying the background pattern of the OWrite gray area around a document page. See also \$forecolor and \$backcolor.
\$checkoverflow (v3.6.5)	Boolean	If kTrue, OWrite prevents users from adding more content than can be displayed in the field by executing an undo when content was caused to overflow the fields boundary. See also: evOverflow and kWriOverflow...
\$sconvtoshared	Boolean	If kTrue, images imported via RTF will be converted to CS24 format. If kFalse, the images original format is retained.
\$scuralign	Integer	Specifies the alignment of the current selection. If the selected text has different alignment, #NULL is returned. Use the constants kWriAlign...
\$scurbold	Boolean	Specifies the bold state of the current selection. If the selected text has different states, #NULL is returned.
\$scurbookmark (v3.0)	Character	Specifies the bookmark name for the selected text. See also Bookmarks and \$getbookmarks().
\$scurfirstindent	Number 2dpt	Specifies the first line indent of the current selection. If the selected text has different indents, #NULL is returned. The first line indent is specified as an offset from \$scurleftindent.
\$scurfont	Character	Specifies the font name of the current selection. If the selected text has two or more fonts, #NULL is returned.
\$scurfontcolor	Integer	obsolete see \$scurtextcolor.
\$scurfontsize	Integer	Specifies the font size of the current selection. If the selected text has two or more font sizes, #NULL is returned.

<code>\$curlhighlight</code> (v3.0)	Integer	sets or clears the current highlight colour. It takes standard <code>rgb()</code> values or any of the Omnis colour constants. To clear the highlight, you can assign <code>#NULL</code> or the <code>kColorDefault</code> option from the Omnis colour picker.
<code>\$curlitalic</code>	Boolean	Specifies the italic state of the current selection. If the selected text has different states, <code>#NULL</code> is returned.
<code>\$curlleftindent</code>	Number 2dp	Specifies the left indent of the current selection. If the selected text has different indents, <code>#NULL</code> is returned.
<code>\$curlspacing</code>	Integer	Specifies the line spacing of the current selection. If the selected text has different line spacing, <code>#NULL</code> is returned. Use the constants <code>kWriLSpace...</code>
<code>\$curlistisbullet</code>	Boolean	Returns <code>kTrue</code> if the current paragraph has a bullet. This property is provided to support the bullet list check button in the OWrite interface. Setting this boolean will change <code>\$curlisttype</code> to <code>kWriListBullet</code> . Clearing this boolean will change <code>\$curlisttype</code> to <code>kWriListNone</code> .
<code>\$curlistisnum</code>	Boolean	Returns <code>kTrue</code> if the current paragraph is numbered. This property is provided to support the numbered list check button in the OWrite interface. Setting this boolean will change <code>\$curlisttype</code> to <code>kWriListDecimal</code> . Clearing this boolean will change <code>\$curlisttype</code> to <code>kWriListNone</code> . See also <code>\$curlistnumstart</code>

<p>\$curlistlevel (v5.0)</p>	<p>Integer</p>	<p>Specifies the indent level of the list item. By default this is zero for all list items in a single level list. Increasing this value will create a sub-list with its own list counter. Certain key presses will manipulate the list:</p> <ul style="list-style-type: none"> • "Return" key: pressing the Return key on an empty list entry will reduce the list level by one or turn off the list if the list level is already zero. • "Tab" key: pressing the tab key at the beginning of a list item will increase the list level by one. • "Backspace" key: pressing the backspace key at the beginning of the list item will reduce the list level by one or turn off the list if the list level is already zero. <p>When increasing a list level, oWrite will search for previous list entries of the same level in the document and use the same list type for the newly indented list entry. If no other list entry of the same level is found, oWrite will choose some default similar to choices observed by the latest MS-Word. Of course, after oWrite picks the initial list type, the user or your code can change this to another type by assigning \$curlisttype.</p>
<p>\$curlistnumstart (v3.0)</p>	<p>Integer</p>	<p>sets or clears the starting number for numbered lists. Setting \$curlistnumstart to anything other than zero sets the start number for the selected paragraph. Subsequent paragraphs that do not have \$curlistnumstart set, will follow on from the previous paragraph. Assigning zero to \$curlistnumstart clears the start number.</p>
<p>\$curlisttype</p>	<p>Integer</p>	<p>Specifies the paragraphs list type. Use the constants kWriList...</p>
<p>\$curobjalign</p>	<p>Integer</p>	<p>Specifies the current objects alignment. This property is only applicable for in-line objects. One of the kWriObjAlign... constants.</p>

\$curobjautosize	Boolean	If kTrue, calculated text boxes will be sized to fit text. The property can also be set for table cells. By default this is set to kTrue for all cells. Setting it to kFalse will prevent the content from resizing a table cell/row. The content is clipped instead.
\$curobjborderstyle	Integer	Specifies the border style of a text box. One of the kWriBord... constants.
\$curobjbottommargin	Number 2dp	Distance between the text box's bottom border and its text.
\$curobjcalc (v3.0)	Character	The data calculation for calculated fields, text-boxes, table cells and pictures. The specified calculation can be any valid Omnis calculation or notation. The calculation or notation must return the appropriate data for the field. See also \$curtblcalc, \$sevalcalcs and \$sevallocal.
\$curobjclickcalc (v3.0)	Character	The calculation that is executed when \$curobjclicks is enabled and the user clicks the object. This calculation can be empty in which case the event evObjClick is send to the \$event method.
\$curobjclicks	Boolean	If kTrue, the object can be clicked by the user like a hyper-link. The objects \$curobjclickcalc calculation is evaluated when a click is recorded on the field. This calculation would typically call one of your notation methods to execute the click. If \$curobjclickcalc is empty, the evObjClick event is triggered instead.
\$curobjcontainer (v3.0)	Boolean	Container objects (headers, footers, text boxes and table cells) may contain objects such as in-line calculations, pictures and info objects. Traditionally when the selection is adjacent to one of the in-line objects, all \$curobj... properties will apply to the in-line object. Sometimes it is desirable to address the container object instead. Setting the property \$curobjcontainer to kTrue, redirects all \$curobj... properties to the container object containing the selection, regardless of the selection within the container object.
\$curobjdata	Varied	obsolete in v3.0 , see \$curobjname and \$curobjcalc.

\$curobjdatadpi (v3.0)	Integer	returns the current DPI of the image as determined by the image resolution and the current width and height of the image box in the document. When assigning this property, an image can be scaled down to a desired quality, based on the current bounding box. This property will give end users some control over image quality and memory requirements if this feature is exposed by the interface. See also \$maximagesize
\$curobjdatasrc (v3.0)	Character	Source (Omnis calculation) for picture data. If not empty, the picture data is not saved with the document but fetched when required by evaluating this calculation. See also Picture alternative data
\$curobjdisplay (v3.0)	Varied	The display text for calculated fields, text-boxes, and table cells, or the display icon id for calculated pictures. If empty, the fields, text-boxes, and cells will display \$curobjname and calculated pictures will display the typical place holder image.
\$curobjevalmaxheight (v5.0)	Number 2dp	This property applies to text boxes and table cells. Setting \$curobjevalmaxheight to a non-zero positive value, limits the box or cells vertical growth to accommodate data to the specified centimetres or inches. If the assigned value is negative, the height is limited to the number of specified lines of text based on the font line height that is used inside the cell. The inner maximum cell height is calculated assuming single paragraph of text that may or may not wrap. In other words the inner cell height is based on the following formula: $(\text{font ascent} + \text{font descent}) * (-\$curobjevalmaxheight) + \$spacebefore + \$spaceafter$ This feature is useful to limit the height of pictures to perhaps conform to the expected number of lines of text in the other cells of the row. Equally, cells containing text can be limited to the same number of lines by assigning the same max value to all cells in a row.
\$curobjfmt	Integer	Specifies the current objects formatting. One of the kWriObjFmt... constants.
\$curobjframeoptions	Integer	Specifies the framing options for the selected cells. See constants kWriFrame... for more details.

\$curobjheight	Number 2dp	The height of the current object in centimeters or inches.
\$curobjhorzoffset	Number 2dp	Horizontal offset from objects anchor point in centimeters or inches.
\$curobjid	Integer (read-only)	Returns the unique ID of the currently selected object.
\$curobjfillcolor	Integer	The objects background fill color. (24 bit RGB)
\$curobjleftmargin	Number 2dp	Distance between the text box's left border and its text.
\$curobjlinecolor	Integer	The border line color of a text box. (24 bit RGB)
\$curobjlinesize	Number 2dp	The border line size of a text box. The size is specified in Points (1 Point = 1/72 inch). The overall size of the border will depend on the border style (i.e. single line, double line, etc).
\$curobjlinestyle	Integer	Specifies the border line style of a text box. One of the kWriLine... constants.
\$curobjlockaspect	Boolean	If kTrue, when object is sized, the aspect ratio is maintained. (Picture objects only)
\$curobjname (v3.0)	Character	The internal name of the OWrite object. Applies to the calculated field, picture and text box. See also \$curobjdisplay.
\$curobjnoenter (v3.0)	Boolean	Setting this property to true for text boxes or table cells, prevents the user from editing the content of that text box or cell.
\$curobjorigheight	Number 2dp	The current objects original height in centimeters or inches. Mainly used for picture objects.
\$curobjorigwidth	Number 2dp	The current objects original width in centimeters or inches. Mainly used for picture objects.
\$curobjresult (v1.50)	Varied	For all objects, returns the objects data. See also \$curobjdata. From version 1.63 this property can be assigned to force the object to show the given result as if it had been evaluated.
\$curobjrightmargin	Number 2dp	Distance between the text box's right border and its text.

\$scurobjshowrtf (v1.63)	Boolean	If kTrue, a calculated field will display the unevaluated RTF if the objects calculation contains simple RTF (the calculation string begins with “{\rtf”)
\$scurobjstripempty (v1.62)	Boolean	If kTrue, the object will strip empty lines from calculation results. (calculated fields, table cells)
\$scurobjtooltip (v1.62)	Character	Tool-tip text for the current object.
\$scurobjtopmargin	Number 2dp	Distance between the text box's top border and its text.
\$scurobjtype	Integer (read-only)	Returns the type of the current object. It returns one of the kWriObjType... constants. Possible return values are; kWriObjTypeNone, kWriObjTypePict, kWriObjTypeCalc, kWriObjTypeCalcPict, kWriObjTypeTextbox and kWriObjTypeTableCell
\$scurobjuserdata (v2.2.4)	Character Date Number List Row	This property supports a limited set of data types: Character, Date, Number, List or Row (we recommend you use a character, or better still a row variable to allow for future expansion of the data). It can contain any data that you wish and is useful for storing your own custom information with your OWrite document object. The custom data is saved with the OWrite document and is maintained in RTF. Every OWrite object, i.e. Table cell, Text box, Picture and Calculated field, has it's own storage. The OWrite table object has a separate property \$scurtbluserdata.
\$scurobjvertoffset	Number 2dp	Vertical offset from objects anchor point in centimeters or inches.
\$scurobjwdbottom	Number 2dp	The distance between object and wrapped text at the bottom of the object.
\$scurobjwdleft	Number 2dp	The distance between object and wrapped text to the left of the object.
\$scurobjwdright	Number 2dp	The distance between object and wrapped text to the right of the object.
\$scurobjwdtop	Number 2dp	The distance between object and wrapped text at the top of the object.
\$scurobjwidth	Number 2dp	The width of the current object in centimeters or inches.

\$currighthindent	Number 2dp	Specifies the right indent of the current selection. If the selected text has different indents, #NULL is returned.
\$scurspaceafter	Integer	Specifies the paragraph spacing at the end of the paragraph. The spacing is specified in Points (1 Point = 1/72 of an inch)
\$scurspacebefore	Integer	Specifies the paragraph spacing at the beginning of the paragraph. The spacing is specified in Points (1 Point = 1/72 of an inch)
\$scurstrikethrough (v2.0)	Integer	Specifies the strike-through state of the current selection. If the selected text has different states, #NULL is returned. Valid range is 0 to 2.
\$scurstylename	Character	Specifies the style name of the current selection. If the selected text has two or more styles, #NULL is returned.
\$scursubscript	Boolean	Specifies the subscript state of the current selection. If the selected text has different states, #NULL is returned.
\$scursuperscript	Boolean	Specifies the superscript state of the current selection. If the selected text has different states, #NULL is returned.
\$scurtabs	Character	<p>Specifies the tabs of the current selection. If the selected text has different tabs, #NULL is returned. Tabs are specified as a single character indicating the tab type followed by the position as a number with 2 decimal places. Tabs must be separated by spaces.</p> <p>L = left aligned tab C = center aligned tab R = right aligned tab D = decimal point tab</p> <p>Example: "L2.50 C5.00 R7.50 D10.25"</p>
\$scurtblalign	Integer	Specifies the horizontal alignment of the table. Same as \$curalign.
\$scurtblapplymode	Constant	Changes the way property changes are applied to the table. The default is that properties changes are assigned to the current selection. See kWriTblApply... for more details.

\$scrtblcalc (v3.0)	Character	The data calculation for a table object. The specified calculation can be any valid Omnis calculation or notation. The calculation or notation must return an Omnis list or the name of an Omnis list variable which must be a task or instance variable. See also \$scurobjcalc, \$sevalcalcs and \$sevallocal.
\$scrtblcellspacing	Number 2dp	The spacing between table cells in centimeters or inches.
\$scrtblcolumnwidth	Number 2dp	The width of the selected table column(s) in centimeters or inches.
\$scrtbldata	Character	obsolete in v3.0 , see \$scrtblname and \$scrtblcalc.
\$scrtblextendable	Boolean	If kTrue, a new row is added every time a user tabs out of the last cell of the table.
\$scrtblid	Integer	Unique ID of the current table object.
\$scrtblindent	Number 2dp	Specifies the amount in centimeters or inches by which the table is indented from the left of document margin. Same as \$scurleftindent.
\$scrtblmultidatarows (v2.2.2)	Boolean	If true, all table rows of the type kWriTblRowNormal are mapped to a single Omnis list row.
\$scrtblname (v3.0)	Character	The internal name of the table object. See also \$scrtblcalc.
\$scrtblpagefooters	Boolean	If kTrue, the current table field will repeat footer rows on each page that the table crosses.
\$scrtblpageheaders	Boolean	If kTrue, the current table field will repeat header rows on each page that the table crosses.
\$scrtblresult	List	The result of calculation specified by \$scrtbldata. This property can also be directly assigned to display an Omnis list in an OWrite table object.
\$scrtblrowevalcansplit (v3.0)	Bool	If kTrue, table row can be split accross pages during evaluation (display/printing only). See also Split Table Rows and Editing Evaluated Documents.
\$scrtblrowheight	Number 2dp	The height of the selected table row(s) in centimeters or inches.

\$scurtblrowid	Integer	Unique ID of the current table row
\$scurtblrowtype	Constant	The type of the current table row. One of the kWriTblRow... constants.
\$scurtbluserdata (v2.2.4)	Character Date Number List Row	This property supports a limited set of data types: Character, Date, Number, List or Row (we recommend you use a character, or better still a row variable to allow for future expansion of the data). It can contain any data that you wish and is useful for storing your own custom information with your OWrite table object. The custom data is saved with the OWrite document and is maintained in RTF. Other OWrite objects such as a Table cell, Text box, Picture and Calculated field, has it's own storage too, see \$scurobjuserdata.
\$scurtextcolor (v3.0 - was \$scurfontcolor)	Integer	Specifies the font color of the current selection. If the selected text has two or more font colors, #NULL is returned. (24 bit RGB)
\$scurunderline	Boolean	Specifies the underline state of the current selection. If the selected text has different states, #NULL is returned.
\$dataname (fat-client v2.2.2)	Character	Specifies the name of the Omnis field that supplies or receives the document data. Using this property makes \$savedata() and \$loaddata() obsolete. This property must be used with web-client but should only be used in fat-client for simple edit fields. You must also specify the appropriate format type in \$dataname type.
\$dataname type (v2.2.2)	Constant	Specifies the format type for storing document data in the field specified by \$dataname. One of the kWriFmt... constants.
\$deftab	Number 2dp	Specifies the default tab width in centimeters or inches.
\$docbottommargin	Number 2dp	The document bottom margin in centimeters or inches.
\$docbulletchar (v5.0)	Character	Can be assigned to specify a different character for bullet lists. Note for jsOWrite: Custom bullet chars are not supported by all browsers (i.e. Safari, IE and some other minor browsers do not support it)

<code>\$docdecimaltabchar</code> (v.3.8.5)	Character	Specifies a alternative decimal tab character, such as a comma. By default this property is set to the english standard decimal point. The decimal tab character is a property of the document and is saved with the document data. Thus the property need only be assigned when creating new documents or when converting documents that were created in prior versions.
<code>\$docleftmargin</code>	Number 2dp	The document left margin in centimeters or inches.
<code>\$docorientation</code>	Integer	Specifies the document's paper orientation using the standard Omnis orientation constants.
<code>\$docpagecount</code> (v1.50)	Integer (read-only)	Returns the documents number of pages.
<code>\$docpagenumber</code> (v1.50)	Integer (read-only)	Returns the page number of the current selection
<code>\$docpaper</code>	Integer	Specifies the document paper size using the standard Omnis paper size constants.
<code>\$docpaperlength</code>	Number 2dp	Specifies the paper length in centimetres or inches.
<code>\$docpaperwidth</code>	Number 2dp	Specifies the paper width in centimetres or inches.
<code>\$docrightmargin</code>	Number 2dp	The document right margin in centimetres or inches.
<code>\$docscale</code>	Integer	Scales the document on screen. The valid range is 25% to 400%. Default is 100%. The document scaling is stored with the document when saved.
<code>\$doctopmargin</code>	Number 2dp	The document top margin in centimeters or inches.
<code>\$docuserdata</code>	(any)	This property can be set to any type of data, i.e. binary, row variables, list or simply text. The custom data is saved with the OWrite document data.
<code>\$docwarnings</code> (v2.4.1)	List (read-only)	Returns a list of warnings that occurred when loading a document. The list is defined as Code, Text, Data1, Data2, Data3, Data4. The column Code specifies on of the kWriWarn... constants.

\$editstylename	Character	If this property is not empty, any reads and writes will apply to specified style. If it is empty, reads and writes apply to the current selection.
\$evalcalcs	Boolean	Setting this to kTrue will evaluate calculated objects and display the result. Setting it back to kFalse will remove the results and display the objects in their unevaluated state. See also \$evallocal, \$curobjcalc and \$curtblcalc.
\$evalkeeplf (v1.8.0)	Boolean	When true, both CR and LF characters are imported during evaluation with the meaning that CR characters denote a paragraph break and LF characters denote a new line within a paragraph (soft-return). As a consequence, calculations that return plain text for insertion may not include CR-LF character combinations to denote end of paragraphs. Single CR characters must be used instead, regardless of the platform.
\$evallocal (v1.5.0)	Boolean	If set to kTrue documents are evaluated using the context local to the calling method. The calling method or the method's class must provide the variables or methods required by the documents calculated fields. If this property is kFalse, it is the encapsulating object or window class that must provide the required variables or methods. See \$evalcalcs, \$curobjcalc and \$curtblcalc.
\$evalpermanent (v5.3.0)	Boolean (read only)	When saving data with the parameter bMakeDataPermanent and \$evalcalcs set to kTrue, \$evalpermanent of the saved document will be set and \$evalcalcs cleared. See \$savedata().
\$firstsel	Integer	Start position of the current selection. Note: \$firstsel may be greater than \$lastsel if the selection was made from right to left.
\$firstselcol	Integer	First selected column. A column is deemed selected even if only one cell in the column is selected. See also \$lastselcol.
\$firstselrow	Integer	First selected row. A row is deemed selected even if only a single cell in the row is selected. See also \$lastselrow.

\$firsttabiconid (v3.0)	Integer	If non-zero, this property tells OWrite to use alternative icons from #ICONS or the Omnis icon files for painting the tab symbols in the horizontal ruler. The property \$firsttabiconid must be set to the ID of the first tab icon. Your tab icons must have consecutive IDs and you must specify four icons for the four possible tabs in the following order: left, centre, right, decimal. The OWrite Plus examples include sample icons in #ICONS.OWriteButtons IDs 12012 to 12015.
\$footermargin (v3.0)	Number 2dp	The distance between the bottom edge of the paper and the bottom of the page footer in centimetres or inches. See also Headers & Footers.
\$forecolor (v2.0)	Integer	Standard Omnis field property for specifying the colour of all set pixels in the background pattern of the OWrite gray area around a document page. See also \$backcolor and \$backpattern.
\$grammaroptions (v5.4)	Integer	Specifies the grammar options for oWrite. This property can be assigned one or more of the kWriGrammar... constants. The constants a bit constants and multiple constants can be assigned at the same time.
\$headermargin (v3.0)	Number 2dp	The distance between the top edge of the paper and the top of the page header in centimetres or inches. See also Headers & Footers.
\$headfootenabled (v3.0)	Boolean	If true, the control will show headers and footers and allow the editing of their content and document flow. See also Headers & Footers and \$headfootnoedit.
\$headfootfirstpage (v3.0)	Boolean	If true, document has different header and footer on the first page. See also Headers & Footers.
\$headfootnoedit (v5.0)	Boolean	If set to kTrue, the content of headers and footers cannot be edited, although they are displayed if \$headfootenabled is kTrue.

\$headfootoddeven (v3.0)	Boolean	If true, document has different headers for odd and even pages. See also Headers & Footers.
\$holdupdates	Boolean	Assigning this property to kTrue will stop all redraws. It prevents flashing and increases performance when inserting and formatting large amounts of text from an Omnis method. Restoring this property to kFalse will update the contents
\$horzscroll	Boolean	If kTrue, the horizontal scroll bar is shown.
\$hscroll	Integer	Horizontal scroll position in multiples of 8 screen units.
\$isplaintext (v1.61)	Boolean	This property can be used to check if formatting changes have been made that require the storage of rich text. When loading a plain text document you can set this property to kTrue, and before a save you can query it to check if the user has applied any formatting. Pasting RTF from other word processors or inserting objects will also clear this flag. Note: If \$isplaintext is set to kTrue, OWrite copy operations will only place plain text on the clipboard.
\$lastsel	Integer	End position of the current selection. Note: \$lastsel may be smaller than \$firstsel if the selection was made from right to left.
\$lastselcol	Integer	Last selected column. A column is deemed selected even if only one cell in the column is selected. See also \$firstselcol.
\$lastselrow	Integer	Last selected row. A row is deemed selected even if only a single cell in the row is selected. See also \$firstselrow.
\$linktextcolor (v3.0)	Integer	Specifies the text colour that is used for calculated fields that have \$curobjclicks set to true.
\$linktextstyle (v3.0)	Integer	Specifies one or more of the Omnis style constants kNormal, kBold, kItalic and kUnderline. It is used for calculated field that have \$curobjclicks set to true.

<code>\$maximagesize</code> (v3.0)	Integer	This is a property of the OWrite component. It allows one to specify the maximum size in pixels of a pasted or imported image. If non-zero, images are scaled down if their height or width exceeds the specified maximum. The aspect ratio of the image is maintained. See also <code>\$curobjdatadpi</code> .
<code>\$modified</code>	Boolean	If <code>kTrue</code> , the document has been modified since the last call to <code>\$savedata()</code> . When calling <code>\$savedata()</code> this flag is cleared.
<code>\$newprimeasure</code> (v3.0)	Integer	if non-zero, text during printing is measured using more accurate 1000 point measurements to position individual characters for more cross-platform accuracy (true-type fonts only). See also Important Note on Printing
<code>\$nocursor</code>	Boolean	If <code>kTrue</code> , the flashing input cursor is hidden, but text can still be selected.
<code>\$nohilite</code>	Boolean	If <code>kTrue</code> , selected content is not highlighted and text cannot be copied.
<code>\$nospellcheck</code> (v3.0)	Boolean	Can be set to true to temporarily disable spell checking as you type. Any modifications made while true, will not be spell checked, even when this property is set to false again. If you need to spell check some of the changes that were made, you can use the <code>\$spell</code> method to invalidate the selection.
<code>\$notableoutline</code> (v3.0)	Boolean	If set to <code>kTrue</code> , table cell's will not display an outline if they have not been given a border.
<code>\$notabs</code>	Boolean	If <code>kTrue</code> , OWrite ignores the tab key preventing users from entering tabs in the document.
<code>\$nouserscroll</code> (v3.8.0)	Boolean	If set to <code>kTrue</code> , it will disable all scrolling initiated by the keyboard or the mouse. However, it will still be possible to scroll the content by assigning the notation <code>\$hscroll</code> and <code>\$vscroll</code> .
<code>\$owriteobj</code>	Integer (read-only)	Returns ID of the control or object. This ID is used to pass to other OWrite objects.
<code>\$pageview</code>	Integer	Specifies the view of the document. It can be set to normal, page layout or field view. See <code>kWriView...</code> constants.

<p><code>\$papercolor</code> (v1.51)</p>	<p>Integer</p>	<p>Specifies the document background paper color. This setting is used for display purposes only and does not affect the printed output.</p>
<p><code>\$papercontinuous</code> (v1.60, changed in v3.0)</p>	<p>Boolean</p>	<p>If <code>kTrue</code>, the document is formatted as a continuous sheet of paper. The document height will grow as content is added to the document. When the user causes the document height to change an <code>evPaperChanged</code> event is generated.</p> <p>This property can also be used to with evaluated documents to allow users to edit evaluated table objects.</p> <p>See also <code>Editing Evaluated Documents</code>.</p>
<p><code>\$pasteoptions</code> (v5.1)</p>	<p>Integer</p>	<p>This property encapsulates a control based set of options that control handling of clipboard data when users paste document content from other applications or <code>owrite</code> (see <code>kWriPaste...</code> constants). By default this property is set to</p> <p style="text-align: center;"><code>kWriPastePlainText+kWriPasteRichText+kWriPasteImages</code></p>
<p><code>\$pasterawpicts</code> (v3.5.0)</p>	<p>Boolean</p>	<p>If <code>kTrue</code>, images pasted from the clipboard are converted and stored as a raw PNG.</p>
<p><code>\$printdpi</code></p>	<p>Integer</p>	<p>Specifies the resolution that is used to measure and render text for display on screen. Valid ranges are 72 to 2400 DPI. The default value is 600 DPI. This property does not affect <code>OWrite</code> on the Macintosh when used with <code>Omnis Studio</code> version 4.0 or better. There is currently no point in setting the value to anything greater than 600 DPI, as this is the maximum resolution used by <code>Omnis</code> during printing. When printing to lower resolution printers, it may be worthwhile to change the resolution to match the printers resolution for a more accurate representation on screen.</p>
<p><code>\$readonly</code> (v2.0)</p>	<p>Boolean</p>	<p>Specifies the current read-only state of the document. If <code>kTrue</code>, the <code>OWrite</code> field can be viewed, scrolled, and content can be selected for copying. In read-only mode the flashing caret will appear a little fatter and smaller. Documents can also be loaded as read-only in which this property will also be set to <code>kTrue</code>. See <code>\$loaddata()</code>.</p>

\$redotext	Character (read-only)	Returns text that can be displayed in the tool tip of your redo button.
\$screndpi	Integer	Specifies the DPI at which documents are rendered on screen. If it is set to anything other than the default, documents are rendered in a cross platform manner at the specified resolution. For example setting this property to 96 (the Windows screen DPI), documents will be scaled on the Macintosh to a size comparable to that on Windows whereas documents will not be scaled on Windows. See constants kWriScrDPI...
\$showcms	Boolean	If kTrue, rulers will display centimetres.
\$showinvisibles (v3.0)	Boolean	This property enables or disables the display of invisible characters and other markers, i.e. spaces, tabs, paragraph breaks and bookmarks. When enabled, these characters are displayed using light blue circle, chevron and paragraph symbols.
\$showpaperulers	Boolean	If kTrue, the document rulers are shown.
\$spellinterval	Integer	Interval of background spell check in milliseconds. Valid range is between 1 and 10. At each interval, OWrite will check up to two words. If this property is set to 10, OWrite will be able to check a maximum of 200 words per second, with a setting of 1 this is increased to 2000 words a second. The background spell check is used when a document is first opened and every time you change text in the document.
\$styleafter	Character	Specifies the style of the next paragraph when return is pressed at the end of the current paragraph. This property can only be assigned if \$editstyle name is set.
\$undoenabled	Boolean	If kFalse, any changes made to properties or document content will not be placed on the undo stack.
\$undotext	Character (read-only)	Returns text that can be displayed in the tool tip of your undo button.
\$vertscroll	Boolean	If kTrue, the vertical scroll bar is shown.
\$vscroll	Integer	Vertical scroll position in multiples of 8 screen units.

\$watermarks (v3.8.5)	List	This property can be assigned with the name of a list (instance or task variable) or list data directly. Please see the section Printing Watermarks in the chapter “Designing OWrite”.
--------------------------	------	--

Window/NV Object Methods

\$addstyle()

Syntax: *OWriteObjectRef*.\$addstyle(cStyleName[,cBasedOn])

Version: 2.0

Adds a new style to the document. This action cannot be undone. See also \$removestyle.

Parameter	Description
cStyleName	The name of the new style. A error is returned if the name is already in use.
cBasedOn	The name of an existing style. The new style will copy the style properties of the specified style. An error is returned if the style does not exist.
returns	0 (no error) if successful

\$convselection()

Syntax: *OWriteObjectRef*

.\$convselection(&iFirstSel,&iLastSel,&iObjID,iSelRange,iSelRangeTo[,bTotalCount])

Version: 2.0

Converts the given selection range to another selection range. For example one could convert the current selection to get the count of the selected words.

```
Do ref.$getselection(iFirstSel,iLastSel,iObjID,kWriSRDefault)
Do ref.$convselection(iFirstSel,iLastSel,iObjID,kWriSRDefault,kWriSRWords,1)
Calculate word_count as iLastSel-iFirstSel
```

or one could get the word count of the entire document

```
Calculate iFirstSel as kWriSelectStart
Calculate iLastSel as kWriSelectEnd
Calculate iObjID as 0
Do ref.$convselection(iFirstSel,iLastSel,iObjID,kWriSRDefault,kWriSRWords,1)
Calculate word_count as iLastSel-iFirstSel
```

See also \$getselection(), \$setselection(), kWriSR... and kWriSelect...

Parameter	Description
iFirstSel	Start of selection range to be converted
iLastSel	End of selection range to be converted
iObjID	The object to which the selection belongs
iSelRange	The selection range type of the given selection, one of the kWriSR... constants.
iSelRangeTo	The selection range to which to convert to, one of the kWriSR... constants.

bTotalCount	Specifies if we are interested in a total count that should also include text in table cells and text boxes.
\$docinsert() Syntax: <i>OWriteObjectRef</i> .\$docinsert(iType,data and/or additional info...) Inserts text or an object at the current position replacing the current selection.	
Parameter	Description
iType	Specifies the type of the object or data, one of the kWriObjType... constants. See the following method descriptions for the allowed types and their parameters
data or additional info	The parameters that follow the type depend on the object type being inserted. Please see the additional descriptions below
returns	0 (no error) if successful
Syntax: <i>OWriteObjectRef</i> .\$docinsert(kWriObjTypeChar,cChar) Inserts an individual character.	
Parameter	Description
cChar	The character to be inserted. You can use one of the kWriChar... constants.
Syntax: <i>OWriteObjectRef</i> .\$docinsert(kWriObjTypeText,cText[,iInsertOptions,custom parameters]) Inserts a string of text.	
Parameter	Description
cText	Specifies the text to be inserted. If the text begins with "{rtf" and ends with "}", the text is imported using the RTF parser. In addition, any plain text that starts with the keyword "{red}" will be inserted as red text. Calculated fields may return the red color keyword as part of their result.
iInsertOptions	Specifies the additional kWriInsert... options, i.e. ...,kWriInsertSelect+kWriInsertKeepStyles,... Default is kWriInsertOver. Note: In oWrite, this parameter was traditionally a boolean parameter that indicated if the inserted text was to be selected and therefore does not support kWriInsertAfter. Only jsoWrite supports kWriInsertAfter. To achieve the same result in oWrite desktop, use \$setselection prior to executing \$docinsert.

custom parameters	Name value pairs specifying additional parameters when inserting text. Example ...,kWriInsertKeepStyles, kWriLoadRawPicts, kTrue) the above is currently the only supported custom parameter.
Syntax: <i>OWriteObjectRef.\$docinsert(kWriObjTypePict,pPicture[,iInsertOptions])</i> Inserts a Picture.	
Parameter	Description
pPicture	Specifies the picture data. The picture data must be converted to an Omnis shared picture (CS24 or CS32) or must be specified as raw PNG or JPEG data* via a binary variable, if documents are to remain cross platform. IMPORTANT NOTE: PNG images with transparency (alpha pixels) will display correctly on screen, but when printed the transparency is lost as Omnis currently does not support the printing of transparent pixel data within images. * OWrite currently does not support any other raw formats
iInsertOptions	Specifies the additional kWriInsert... options, i.e. ...,kWriInsertSelect+kWriInsertApplyMaxImageSize) Default is kWriInsertSelect.
Syntax: <i>OWriteObjectRef.\$docinsert(kWriObjTypeCalc,cCalculation[,iInsertOptions])</i> Inserts a calculated picture object.	
Parameter	Description
cCalculation	Specifies the calculation in the following format "Name;Omnis Calculation". The calculation must evaluate to text or RTF.
iInsertOptions	Specifies the additional kWriInsert... options, i.e. ...,kWriInsertSelect+kWriInsertKeepStyles)
Syntax: <i>OWriteObjectRef.\$docinsert(kWriObjTypeCalcPict,cCalculation)</i> Inserts a calculated picture object.	
Parameter	Description
cCalculation	Specifies the calculation in the following format "Name;Omnis Calculation". The calculation must evaluate to a picture in the format as specified by "\$docinsert(kWriObjTypePict..." above.

iInsertOptions	Specifies the additional kWriInsert... options, i.e. ...,kWriInsertSelect)
<p>Syntax: <i>OWriteObjectRef</i>.\$docinsert(kWriObjTypeTextbox, cCalculation, iBorderStyle, iBorderLineStyle, iBorderColor,nBorderLineStyle, nTextBoxWidth, nRowHeight)</p> <p>Inserts a text box object.</p>	
Parameter	Description
cCalculation	Specifies the calculation in the following format "Name;Omnis Calculation" or an empty string if no calculation is required. The calculation must evaluate to text or RTF.
iBorderStyle	The border style, one of the kWriBord... constants.
iBorderLineStyle	The border line style, on of the kWriLine... constants.
iBorderColor	The border color, 24bit RGB.
nBorderLineStyle	The border line size in points.
nTextBoxWidth	The text box width in centimeters or inches.
nRowHeight	The text box height in centimeters or inches.
<p>Syntax: <i>OWriteObjectRef</i>.\$docinsert(kWriObjTypeTable, cCalculation, iBorderStyle, iBorderLineStyle, iBorderColor, nBorderLineStyle, iColumnCount, iRowCount, nColumnWidth, nRowHeight)</p> <p>Version: v2.0</p> <p>Inserts a table object.</p>	
Parameter	Description
cCalculation	Specifies the calculation in the following format "Name;Omnis Calculation" or an empty string if no calculation is required. The calculation must evaluate to a list.
iBorderStyle	The border style, one of the kWriBord... constants.
iBorderLineStyle	The border line style, on of the kWriLine... constants.
iBorderColor	The border color, 24bit RGB.
nBorderLineStyle	The border line size in points.
iColumnCount	The number of columns
iRowCount	The number of rows
nColumnWidth	Default column width in centimeters or inches.
nRowHeight	Default row height in centimeters or inches.

\$docprint()

Syntax: *OWriteObjectRef*.\$docprint(cDocumentName,bShowJobsetup,bKeepJobOpen)

Platforms: Fat-client only

Prints the loaded document to the current Omnis report destination.

Parameter	Description
cDocumentName	The name to be displayed during printing.
bShowJobsetup	If kTrue, the job setup dialog will be shown.
bKeepJobOpen (v2.0)	If kTrue, the job is kept open so subsequent calls to \$sprint will print the document to the same print job. The last call to \$sprint must pass kFalse to close the job.
bBorderless (v3.8.6)	If kTrue, the document is printed as if printing to a border-less printer, allowing content to fill the entire paper. This is especially useful when printing to PDFDevice.
returns	0 (no error) if successful

\$editclear()

Syntax: *OWriteObjectRef*.\$editclear()

Clears the selection.

Parameter	Description
returns	0 (no error) if successful

\$editcopy()

Syntax: *OWriteObjectRef*.\$editcopy()

Copies the selected text or object to the clipboard.

Parameter	Description
returns	0 (no error) if successful

\$editcut()

Syntax: *OWriteObjectRef*.\$editcut()

Copies the selected text or object to the clipboard and then clears it from the document.

Parameter	Description
returns	0 (no error) if successful

\$editpaste()

Syntax: *OWriteObjectRef*.\$editpaste()

Paste the contents of the clipboard at the current position, replacing the current selection.

Parameter	Description
returns	0 (no error) if successful
\$editredo() Syntax: <i>OWriteObjectRef</i> .\$editredo() Redo the last undo operation.	
Parameter	Description
returns	0 (no error) if successful
\$editselectall() Syntax: <i>OWriteObjectRef</i> .\$editselectall() Selects the entire document.	
Parameter	Description
returns	0 (no error) if successful
\$editundo() Syntax: <i>OWriteObjectRef</i> .\$editundo() Undo the last edit operation.	
Parameter	Description
returns	0 (no error) if successful
\$sendundo() Syntax: <i>OWriteObjectRef</i> .\$sendundo(cText) This method must be called after a call to \$startundo(). It groups the last set of changes into one undo operation.	
Parameter	Description
cText	Specifies the text for the undo item. Do not include the word Undo as part of the text. This is provided by OWrite.
returns	0 (no error) if successful
\$findinit() Syntax: <i>OWriteObjectRef</i> .\$findinit(&iFindOWriteObjRef[,bEntireDocument]) Platform: Fat-client only Before you can call \$findnext() or \$replace(), you must call this function to build an array of ranges to be searched. This array will include, either the current selection if there was one, or all searchable text of the document including text inside text boxes.	
Parameter	Description

iFindOWriteObjRef	The \$owriteobj value of the OWriteSearch object that will be initialised.
bEntireDocument	If kTrue, the entire document is searched, otherwise the current selection is searched.
returns	1 if successful or 0 if initialisation failed or a negative error code

\$findnext()

Syntax: *OWriteObjectRef*.\$findnext(&iFindOWriteObjRef)

Platform: Fat-client only

Find the text or formatting as specified by the given object. Before you can call this method, you must have called \$findinit() to initialise the find object with the text ranges to be searched.

Parameter	Description
iFindOWriteObjRef	The \$owriteobj value of the OWriteSearch object that contains the search criteria.
returns	1 if the search was successful or 0 if it failed or a negative error code

\$getbookmarks()

Syntax: *OWriteObjectRef*.\$getbookmarks(&lList)

Returns a single column list of bookmark names.

See also Bookmarks and \$curbookmark.

Parameter	Description
lList	The list variable that is to receive the bookmark names.
returns	0 (no error) if successful

\$getobjslst()

Syntax: *OWriteObjectRef*.\$getobjslst(lList,iObjType)

Populates the list with information of objects in the current document.

Parameter	Description
-----------	-------------

IList	The list that is to receive the object information. It must be formatted prior to calling this method. The columns of the list are; <ul style="list-style-type: none"> • Type - One of the kWriObj... constants. • Ident - The objects ID. • FirstSel - The starting position of the object. • LastSel - The end position of the object. • Name - The display name of the object. • Calculation (v1.62, optional) - The objects calculation. • CalcResult (v1.62, optional) - The calculation result. • DocResult (v2.0, optional) - The result taken from the document The information from the FirstSel, LastSel and Ident columns can be used to select the object by calling \$setselection().
iObjType	The type of the objects for which to return information. One of the kWriObjType... constants. If kWriObjTypeNone is specified, all objects are returned.
returns	0 (no error) if successful
\$getselbounds()	
Syntax: <i>OWriteObjectRef</i> .\$getselbounds(iTop,iLeft,iBottom,iRight)	
Returns the co-ordinates of the current selection in screen co-ordinates. This method is useful if you need to move an obscuring window to show the current selection.	
Parameter	Description
iTop	The top co-ordinate.
iLeft	The left co-ordinate.
iBottom	The bottom co-ordinate.
iRight	The right co-ordinate.
returns	0 (no error) if successful
\$getselpageoffset()	
Syntax: <i>OWriteObjectRef</i> .\$getselpageoffset(iPage,nHorzOffset,nVertOffset,bEndSel)	
Returns the offset from the top-left of the page and the page number of the start selection or end selection in cms or inches.	
Parameter	Description
iPage	Contains the page number on return
nHorzOffset	Returns the horizontal offset of the selection from the left edge of the page.

nVertOffset	Returns the vertical offset of the selection from to the top edge of the page.
bEndSel	Specifies if the start or the end of the current selection is required.
returns	0 (no error) if successful

\$getselection()

Syntax: *OWriteObjectRef*.\$getselection(&iFirstSel,&iLastSel,&iObjID,iSelRange)

Version: 2.0

Returns the current selection range in the specified format. It can also be used to count the selected words or rows or other counts supported by this method.

```
Do ref.$getselection(iFirstSel,iLastSel,iObjID,kWriSRRows)
Calculate row_count as iLastSel-iFirstSel
```

See also \$setselection() and \$convselection().

Parameter	Description
iFirstSel	The variable that is to receive the start of the current selection.
iLastSel	The variable that is to receive the end of the current selection.
iObjID	The variable that is to receive the ID of the selected object.
iSelRange	Specifies the type of the required range, one of the kWriSR... constants.
returns	0 (no error) if successful

\$getstylelist()

Syntax: *OWriteObjectRef*.\$getstylelist(&lList)

Populates the list with the style names from the loaded document.

Parameter	Description
lList	The list to receive the style names.
returns	0 (no error) if successful

\$globalpos()

Syntax: *OWriteObjectRef*.\$globalpos(iLocalPos)

Converts a selection local to the current object to a global document position. This is useful if one needs to know where a position is in relation to other objects in the document.

Parameter	Description
iLocalPos	The character position inside a text box or table cell.
returns	The character position global to the document.

\$::insert()

Renamed to \$docinsert(). Existing code will continue to work. See case 1965 in version 5.3 [release notes](#) for more details.

\$loaddata()

Syntax: *OWriteObjectRef*.\$loaddata(xDocData[,iFormat,bReadOnly])

Sets the document data.

Parameter	Description
xDocData	The document data
iFormat	One of the kWriFmt... constants specifying the format of the data. Default is kWriFmtDefault.
bReadOnly	If kTrue, the document cannot be modified but you can still select and copy text or objects if \$enabled is set to kTrue. Default is kFalse. See also \$readonly.
returns	0 (no error) if successful

\$picturefrompage()

Syntax: *OWriteObjectRef*.\$picturefrompage(iPage,nMaxWidth,nMaxHeight,&pPicture)

Version: 2.0

Creates an Omnis picture of the specified page in the specified resolution. Ideal for creating thumb-images of your documents or document templates.

Parameter	Description
iPage	The page number.
nMaxWidth	The width of the image in screen units (pixels)
nMaxHeight	The height of the image in screen units (pixels)
pPicture	The omnis picture variable that receives the image
returns	0 (no error) if successful

\$popupmenu()

Syntax: *OWriteObjectRef*.\$popupmenu(lMenuItem,bBelowControl,iHorzAdjust,iVertAdjust)

Pops-up a menu from the given list at the current cursor position, or below the control the mouse is hovering over. *Also see OWrite.\$popupmenulist()*

Parameter	Description
-----------	-------------

IMenuList	<p>A four column list specifying the menu items. The columns are as follows; Text, ID, Flags, SubmenuItems. IDs 1 to 100 are reserved and must not be used by custom menu items. The following special IDs can be used to implement edit menu item that are managed appropriately by OWrite; 1 = Undo, 2 = Redo, 3 = Cut, 4 = Copy, 5 = Paste, 6 = Clear, 7 = Select All. See kWriMenuItemEdit... constants.</p> <p>An ID of -1 indicates that the fourth column contains menu items for the hierarchical menu.</p> <p>The flags column specifies one or more status flags, see kWriMenuItem... constants.</p>
bBelowControl	Position the menu at the bottom edge of the control under the mouse.
iHorzAdjust	Additional horizontal offset in screen units.
iVertAdjust	Additional vertical offset in screen units.
returns	the ID of the menu item that was selected or zero if the user canceled the menu.

\$print()

\$::print()

Renamed to \$docprint(). Existing code will continue to work. See case 1965 in version 5.3 [release notes](#) for more details.

\$removestyle()

Syntax: *OWriteObjectRef*.\$removestyle(*cStyleName*)

Version: v2.0

Removes the specified style from the document. This action cannot be undone. See also \$addstyle().

Parameter	Description
cStyleName	The style to be removed. If the style is in use or the style does not exist, an error is returned
returns	0 (no error) if successful

\$replace()

Syntax: *OWriteObjectRef*.\$replace(&iFindOWriteObjRef,&iReplaceOWriteObjRef,bAll)

Platforms: (Fat-client only)

Replaces the text or styles using the given search data with the text or styles supplied by the replace data. Before you can call this method, you must have called \$findinit() to initialise the find object with the text ranges to be searched.

Parameter	Description
iFindOWriteObjRef	The \$owriteobj value of the OWriteSearch object that contains the search criteria.
iReplaceOWriteObjRef	The \$owriteobj value of the OWriteSearch object that contains the replace data.
bAll	If kFalse, only the next occurrence of the search criteria is replaced. If kTrue, all occurrences are replaced.
returns	1 if text was found to be replaced
\$savedata() Syntax: <i>OWriteObjectRef</i> .\$savedata(&xDocData[,iFormat,bMakeDataPermanent,custom]) Returns the document data. See also \$dataname and \$datanameype.	
Parameter	Description
xDocData	The binary or text variable to receive the document data.
iFormat	The desired format of the data. One of the kWriFmt... constants specifying the format of the data. Default is kWriFmtDefault.
bMakeDataPermanent (updated in v5.3.0)	If kTrue and \$evalcalcs is kTrue, embedded calculations are replaced by their result data. Further evaluation of the saved document will not be possible. In version 5.3 or later, saving data with this flag will clear the \$evalcalcs flag and set the \$evalpermanent flag in the saved document data when saving to kWriFmtDefault.
[custom] (v1.50)	Some export formats may require you to specify additional information specific to the chosen format. Custom parameters are identified by OWrite constants followed by the value. That means for every custom parameter you pass two additional parameters to \$savedata. For example: <pre>Do OWriteRef.\$savedata(var, kWriFmtHTML, kFalse, kWriHtmlBgColor, rgb(255,200,200), kWriHtmlNoAutoSize, kTrue,...)</pre> See kWriHtml..., kWriSave... and kWriText for details of custom parameters for this method.
returns	0 (no error) if successful

\$setdatafromsrc()

Syntax: *OWriteObjectRef*.\$setdatafromsrc(pPicture)

This function is used in the web-client to set the picture data after an evGetDataFromSrc event was received and the picture data has been fetched from the server.

Parameter	Description
pPicture	The picture data for the picture object that generated the evGetDataSrc event.
returns	returns 1 if successful, 0 otherwise

\$setselection()

Syntax: *OWriteObjectRef*.\$setselection(iFirstSel,iLastSel,iObjID[,iSelRange])

Changes the current selection. See also \$getselection(), \$convselection(), kWriSR... and kWriSelect...

Note: When selecting rows and cells within a table object, the HIWORD in a selection range specifies the row number and the LOWORD specifies the column number. To select column two in row three one would specify 3*65536+2.

The code `$setselection(3*65536+2,3*65536+3,table_id)` would select columns 2 and 3 in row 3 in the table specified by table_id.

Parameter	Description
iFirstSel	The start of the selection range.
iLastSel	The end of the selection range.
iObjID	The ID of the object to be selected. If the specified object is a text box or table cell, the selection range applies to the text inside the text box.
iSelRange (v2.0)	Specifies the type of the given range, one of the kWriSR... constants. By default the range is assumed to be a standard selection range, character based including white space and object place holders. Example: The following call <code>\$setselection(0,5,0,kWriSRWords)</code> would select words 1 to 5.
returns	0 (no error) if successful

\$spell()

Syntax: *OWriteObjectRef*.\$spell(iAction)

Platforms: Fat-client only

Performs the specified spell checker action.

Parameter	Description
iAction	The action to perform. One of the kWriSppl... constants.

returns	0 (no error) if successful
\$startundo() Syntax: <i>OWriteObjectRef</i> .\$startundo() Call this method if you need to combine a number of changes into one undo operation. When you have made your changes you must call \$sendundo().	
Parameter	Description
returns	0 (no error) if successful
\$tableaction() Syntax: <i>OWriteObjectRef</i> .\$tableaction(iAction,bExecute) Version: 2.0 Executes the specified table action, or tests if the specified action can be executed.	
Parameter	Description
iAction	The table action to execute or test. One of the kWriTblAct... constants.
bExecute	If kTrue, execute the action, otherwise merely test if the action can be executed.
returns	1 if successful or the action can be executed, 0 otherwise.

Window Object Events

evOverflow

Version: 3.6.5

This event is generated when the user caused an overflow to occur and \$checkoverflow is set to kTrue. See also \$checkoverflow and kWriOverflow...

Event Parameter	Description
pOverflowType	One of the kWriOverflow... constants, indicating what caused the overflow to occur.

evClick

Version: 2.0

This event is generated when a click occurs on an object in the document

no event parameters

evContextObject

This event is generated when the user right/ctrl clicks while over an object. You should present the user with the appropriate context menu.

Event Parameter	Description
pX	The horizontal screen co-ordinate.
pY	The vertical screen co-ordinate.

evContextSpell

Platforms: (Fat-client only)

This event is generated when the user right/ctrl clicks while over an incorrectly spelled word. You should present the user with a menu allowing the user to choose a word from the list of suggestions.

Event Parameter	Description
pX	The horizontal screen co-ordinate.
pY	The vertical screen co-ordinate.
pList	List containing suggestions.

evContextText

This event is generated when the user right/ctrl clicks while over some text. You should present the user with the appropriate context menu.

Event Parameter	Description
pX	The horizontal screen co-ordinate.
pY	The vertical screen co-ordinate.

evDoubleClick

Version: 2.0

no event parameters

evFormatChanged

This event is generated when the current formatting has changed. You should update any interface items that display formatting information to the user.

no event parameters

evGetDataFromSrc

This event is generated when a picture field has a data source calculation that requires evaluating. After fetching the image data from the server, the method \$setdatafromsrc must be used to set the image data.

Version/Platform: v3.0 - Web-Client only

Event Parameter	Description
pObjID	The \$curobjid value of the picture object
pObjName	The \$curobjname value of the picture object
pObjCalc	The \$curobjdatasrc value of the picture object

evIndentChanged

Version: 1.63

This event is generated when the user changes the paragraph indents via the OWrite ruler controls. See also evMarginChanged and evTabChanged.

no event parameters

evKillFocus

Version: 1.63

This event is generated when the OWrite window control or remote form control loses the focus. The evSetFocus and evKillFocus events are very similar to the standard evBefore and evAfter events, but are more reliable as these events are always generated when OWrite receives and loses the focus. See also evSetFocus.

no event parameters

evMarginChanged

Version: 1.63

This event is generated when the user changes the document margins via the OWrite ruler controls. See also evIndentChanged and evTabChanged.

no event parameters

evModified

Version: 1.62

This event is generated when the user modifies the document and \$modified was previously false. If the \$modified flag is subsequently cleared a new event is generated when the user modifies the document again.

*no event parameters***evMultiFind**

Version: v3.0

This event is generated when the multi-selection-find state changes from shown to hidden or vice versa.

See also Multi-Selection Find.

Event Parameter	Description
pSelectionShown	If true, the current multi-find is displaying all matching words or phrases in the window object. If false, the highlights are removed.

evObjClick

Version: v3.0

This event is generated when the user clicks an OWrite object for which \$curobjclicks is true and \$curobjclickcalc is empty, except in the web-client when evObjClick is always generated when \$curobjclicks is true.

Event Parameter	Description
pObjID	The internal ID of the object that was clicked
pObjName	The object's name as specified by \$curobjname.
pObjCalc	The object's click calculation as specified by \$curobjclickcalc. This will always be empty for fat-client, but may contain a calculation to be executed on the server in web-client implementations. The web-client cannot evaluate custom calculations, so evObjClick is always generated regardless of the value of \$curobjclickcalc.

evPasteFF

This event is generated when the user selects the Paste from file option from the edit menu. Typically you should present a file selection dialog and convert the loaded data to a format compatible with OWrite for inserting at the current position. The example library implements a method that supports a number of picture formats, plain text and RTF files.

no event parameters

<p>evPaperChanged Version: 1.60</p> <p>This event is generated when \$papercontinuous is kTrue, and the paper height has changed because the user has added or removed content from the document.</p>	
Event Parameter	Description
pY	The amount in centimetres or inches by which the height has been altered.
pDocHeight (v2.0)	The absolute graphical document height in centimetres or inches. This measurement is the same that is used by OWrite to set the scroll ranges of the vertical scroll-bar.
<p>evProgress Version: 3.5.0</p> <p>This event is generated during loading, saving, exporting, importing or printing of large documents where any of these actions may result in a 2 second or longer delay.</p>	
Event Parameter	Description
ProgressType	Indicates the action being performed. One of the kWriProgress... constants.
ProgressPercent	Value between 1% and 100%, indicating the progress so far.
<p>evSelChanged This event is generated when the current selection changes.</p> <p><i>no event parameters</i></p>	
<p>evSetEditMenu Version: 1.62</p> <p>This event is only relevant when the standard edit menu is replaced. It is generated when the undo and redo menu item text requires updating.</p> <p><i>no event parameters</i></p>	
<p>evSetFocus Version: 1.63</p> <p>This event is generated when the OWrite window control or remote form control receives the focus. The evSetFocus and evKillFocus events are very similar to the standard evBefore and evAfter events, but are more reliable as these events are always generated when OWrite receives and loses the focus. Sell also evKillFocus.</p> <p><i>no event parameters</i></p>	

www.brainydata.com

evStylesChanged

This event is generated when a style has been added. When you receive this event you can call the method `$getstylelist()` to update your list of styles. You can entirely rely on this event to maintain your list of styles for your interface.

no event parameters

evTabChanged

Version: 1.63

This event is generated when the user changes the paragraph's tabs via the OWrite ruler controls. See also `evMarginChanged` and `evIndentChanged`.

no event parameters

Report Object Properties		
Name	Type	Description
\$dataname	Binary	Specifies the Omnis field that contains the binary document data. See also \$dataname and \$dataname type for window/form objects.
\$fitimages	Boolean	If set to kTrue, images are moved to the next page if they don't fit on the current page.
\$headfootenabled (v3.0)	Boolean	If set to kTrue and \$ignorepos is true, OWrite will print headers and footers from the document data.
\$ignorepagebreaks	Boolean	If set to kTrue, page breaks in OWrite documents are ignored and will not generate new pages when encountered during printing.
\$ignorepos	Boolean	If set to kTrue, the report object position is ignored and the document data is laid out according to the Page Layout view of the OWrite control.
\$watermarks (v3.8.5)	List	This property can be assigned with the name of a list (instance or task variable) or list data directly. Please see the section Printing Watermarks in the chapter "Designing OWrite".

OWriteSearch - NV Object Properties

Name	Type	Description
\$completeword	Boolean	If kTrue, only complete words are included in the search.
\$ignorecase	Boolean	If kFalse, the search is case sensitive.
\$multiselectbackcolor (v3.0)	Integer	The back colour for highlighting all matching text within the document. See also Multi-Selection Find.
\$multiselection (v3.0)	Boolean	If true, searches will highlight all matching text within the document. See also Multi-Selection Find.
\$multiselectlist (v3.0)	List	The list of selection ranges for all text that matches the search criteria. See also Multi-Selection Find.
\$multiselecttextcolor (v3.0)	Integer	The text colour for highlighting all matching text within the document. See also Multi-Selection Find.
\$multiwordfind (v3.0)	Boolean	If true, the search phrase is treated as individual words and all matching words are included in the list of matches. See also Multi-Selection Find.
\$owriteobj	Integer	Unique object ID. Use this property when calling the methods \$findinit(), \$findnext() and \$replace(). Do not pass the Omnis object.
\$selectbackcolor (v3.0)	Integer	The back colour for highlighting the current matching text. See also Multi-Selection Find.
\$selecttextcolor (v3.0)	Integer	The text colour for highlighting the current matching text. See also Multi-Selection Find.
\$text	Character	The search or replace text.

(other)	varied	<p>Properties for specifying formatting options for search and replace actions.</p> <p>\$curstylename, \$curfont, \$curfontsize, \$curfontcolor, \$curitalic, \$curbold, \$curunderline, \$curstrikethrough, \$cursuperscript, \$cursubscript, \$curalign, \$curlinespacing, \$curleftindent, \$currightindent, \$curfirstindent, \$curtabs, \$curspacebefore, \$curspaceafter, and \$curlisttype.</p> <p>Any of these properties that are not NULL, will be included in the search.</p>
---------	--------	--



OWriteSearch - NV Object Methods

\$getresultlist()

Syntax: *OWriteSearchRef*.\$getresultlist(&lList,iSelRange,bIncludeRowText)

Version: v3.0

Returns a list of search matches. Can only be used when \$multiselection is true, and after \$findinit() has been called.

See also Multi-Selection Find.

Parameter	Description
lList	The list variable that will receive the list of found matches. The returned list has the following columns. - FirstSel: The start of the selection range. - LastSel: The end of the selection range. - ObjID: ID of the object containing the match. - ObjPos: Global pos of the object. - PageNumber: The page number for the match. - RowText: The text of the entire row, containing the match.
iSelRange	Specifies the selection range mode for the FirstSel and LastSel columns. One of the kWriSR... constants.
bIncludeRowText	If true, each match will include the entire text of the row in the column RowText.
returns	0 (no error) if successful

\$setresultlist()

Syntax: *OWriteSearchRef*.\$setresultlist(&lList,iSelRange)

Version: v3.0

Sets the result list and positions the current selection to the match in the current line of the list,

See also Multi-Selection Find.

Parameter	Description
lList	The updated list
iSelRange	Specifies the selection range mode for the FirstSel and LastSel columns. One of the kWriSR... constants.
returns	0 (no error) if successful

Using the OWrite Ruler

The OWrite ruler can be used to manipulate a paragraphs tabs and indents and the top, left, right and bottom document margins.

Tabs

Without adding any custom tabs to a paragraph, tab characters can be inserted into the document by pressing the tab key and text will be positioned according to the OWrite default tab position.

Using the ruler you can place, remove and move tabs. Supported tabs are left, centre, right and decimal. Once a custom tab is placed the OWrite default tab position is ignored up to the last custom tab in the paragraph. If more tab characters are inserted than available custom tabs, the OWrite default tab position will be used again after the last custom tab.

Developer Note:
The default tab position can be changed via the `$deftab` property.

Placing a Tab

Repeatedly click the tab button to the left of the ruler until the desired tab style is shown.



Now click in the ruler to place the tab.



Removing a Tab

Click and drag the tab away from the ruler and let go. A trash can will appear while dragging outside the ruler.



Moving a Tab

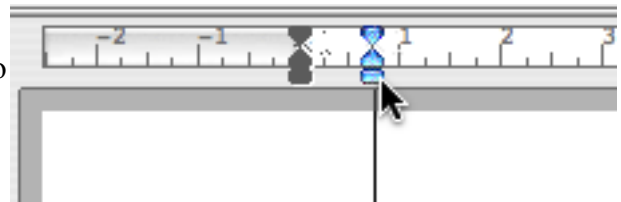
Click and drag the tab to the left or right to move a tab.

Indents

Using the ruler you can change a paragraph's left, right and first line indents.

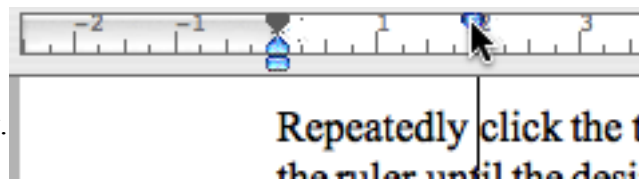
Changing both left and first line indent

Click and hold the hanging indent button and drag to the desired location in the ruler.



Changing first line indent

Click and hold the first line indent button (top button) and drag to the desired location in the ruler.



Repeatedly click the 1
the ruler until the desi

Changing left indent

Click and hold the left indent button and drag to the desired location in the ruler.

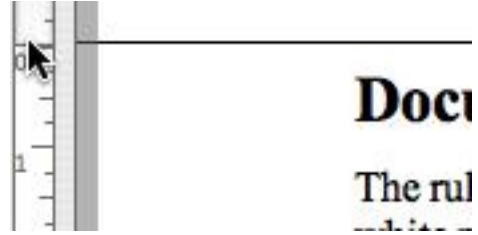


Now click in the rul

Document Margins

The ruler can also be used to change the overall document margins by grabbing the edge between the white ruler area and the gray margin area.

For example, to change the top margin click and drag the edge as shown in the image.



OWrite Cursor Keys

Standard Key Behaviour

The table below shows cursor key combinations and the OWrite behaviour. There are two types of actions, those that move the input cursor and those that scroll the document window.

OWrite Action	Macintosh Keys	Windows/Linux Keys
Previous character	left-arrow	left-arrow
Next character	right-arrow	right-arrow
Previous word	Option & left-arrow	Ctrl & left-arrow
Next word	Option & right-arrow	Ctrl & right-arrow
Previous paragraph	Option & up-arrow	Ctrl & up-arrow
Next paragraph	Option & down-arrow	Ctrl & down-arrow
Beginning of line	Command & left-arrow	Home
End of line	Command & right-arrow	End
Beginning of document	Home	Ctrl & Home
End of document	End	Ctrl & End
Scroll left	Command & Option & left-arrow	Ctrl & Option and left-arrow
Scroll right	Command & Option & right-arrow	Ctrl & Option and right-arrow
Scroll up	Command & Option & up-arrow	Ctrl & Option and up-arrow
Scroll down	Command & Option & down-arrow	Ctrl & Option and down-arrow
Scroll page up	Page Up	Page Up
Scroll page down	Page Down	Page Down

Special Key Behaviour

When OWrite is put in a special display only mode the key behaviours are as follows.

Developer Note:

The special display only mode is achieved by setting the properties \$nocursor and \$nohilito to kTrue.

OWrite Action	Macintosh Keys	Windows/Linux Keys
Beginning of document	Home	Home
End of document	End	End
Scroll left	left-arrow	left-arrow
Scroll right	right-arrow	right-arrow
Scroll up	up-arrow	up-arrow
Scroll down	down-arrow	down-arrow
Scroll page up	Page Up	Page Up
Scroll page down	Page Down	Page Down

Paragraph List Key Behaviour

Version 5.0

When the input cursor is inside a bullet or numbered list, the following keys can be used to manipulate the list indent level.

Key Press	Action
“Return” key	pressing the Return key on an empty list entry will reduce the list level by one or turn off the list if the list level is already zero.
“Tab” key	pressing the tab key at the beginning of a list item will increase the list level by one.
“Backspace” key	pressing the backspace key at the beginning of the list item will reduce the list level by one or turn off the list if the list level is already zero.