

Useful Macro Information For OpenOffice.org By Andrew Pitonyak

**This is not the same as my book
OpenOffice.org Macros Explained (OOME).**

You can download OOME free

**This document, is primarily a random collection of
thoughts and examples (brain dump).**

Last Modified

Wednesday, June 11, 2025 at 04:39:49 PM

Document Revision: 1141

Thank You

Thanks to my wife Michelle Pitonyak for supporting and encouraging me to write this document. Thank you Laurent Godard (and the French translation team) for your good ideas and hard work. Thanks to Hermann-Josef Beckers for his German translation. Kelvin Eldridge, you have great understanding and have helped me understand numerous bugs. Jean Hollis Weber and Solveig Haugland, thank you for the personal replies when I had specific document usage problems. Sasa Kelecevic, and Hermann Kienlein, you have provided many working examples to help me. Andreas Bregas, thank you for the quick replies, descriptions, and fixes. Mathias Bauer, thank you for the explanations and examples revealing the deep mysteries of the internals. I owe a large thank you to the entire open-source community and the mailing lists for providing helpful information and support.

Disclaimer

The material in this document carries no guarantee of applicability, accuracy, or safety. Use the information and macros in this document at your own risk. In the unlikely event that they cause the loss of some or all of your data and/or computer hardware, that neither I, nor any of the contributors, will be held responsible.

Contact Information

Andrew Pitonyak • 6888 Bowerman Street West • Worthington, OH 43085 • USA
home: andrew@pitonyak.org • work:

home telephone: 614-438-0190 • cell telephone: 614-937-4641

Credentials

I have two Bachelors of Science degrees, one in Computer Science and one in Mathematics. I also have two Masters of Science degrees, one in Applied Industrial Mathematics and one in Computer Science. I have spent time at Oakland University in Rochester Michigan, The Ohio State University in Columbus Ohio, and at The Technical University Of Dresden in Dresden Germany.

Public Documentation License Notice

The contents of this Documentation are subject to the Public Documentation License Version 1.0 (the "License"); you may only use this Documentation if you comply with the terms of this License.

A copy of the License is available at <http://www.openoffice.org/licenses/pdl.pdf>

The Original Documentation is <http://www.pitonyak.org/AndrewMacro.odt>

<i>Contribution</i>	<i>Contributor</i>	<i>Contact</i>	<i>Copyright</i>
Original author	Andrew Pitonyak	andrew@pitonyak.org	2002-2014
French translation	The French Native-lang documentation project	doc@fr.openoffice.org	2003
German Translation	Hermann-Josef Beckers	hjb-rheine@t-online.de	2003
Macros	Hermann Kienlein	info@kienlein.com	2002-2003
Macros	Sasa Kelecevic	scat@teol.net	2002-2003
Editing and additions	Christian Junker	christianjunker@gmail.com	2004

Table of Contents

Thank You	2
Disclaimer	2
Contact Information	2
Credentials	2
Public Documentation License Notice	iii
Table of Contents	iv
1. Introduction	1
1.1. Special thanks to Bartosz	1
1.2. Abbreviations and definitions	1
2. Available Resources	3
2.1. Included Material	3
2.2. On line Resources	3
2.3. Translations of this document	5
3. Getting started: concepts	7
3.1. My first macro: “Hello World”	7
3.2. Grouping Code	7
3.3. Debugging	9
3.4. Variables	9
3.5. Accessing And Creating Objects In OpenOffice	9
3.6. Everyone keeps talking about UNO, what is it?	10
3.6.1. Simple types	10
3.6.2. Object	11
3.6.3. Structures	11
3.6.4. Interfaces	11
3.6.5. Services	12
3.6.6. Interfaces and services	13
3.6.7. What type is this object?	13
3.6.8. What methods, properties, interfaces, and services are supported?	16
3.6.9. Languages other than Basic	16
3.6.9.1. CreateUnoService	16
3.6.9.2. ThisComponent	17
3.6.9.3. StarDesktop	17
3.6.10. Accessing methods and properties	18
3.7. Summary	19
4. Examples	21
4.1. Debugging And Inspecting Macros	21
4.1.1. Determine Document Type	21
4.2. X-Ray	23
4.3. Dispatch: Using Universal Network Objects (UNO)	23
4.3.1. The dispatcher requires a user interface	24
4.3.1.1. Modifying the menu – a dispatcher example	24
4.4. Intercept menu commands using Basic	25

5. Miscellaneous Examples	29
5.1. Display Text In Status Bar	29
5.2. Display All Styles In The Current Document	29
5.3. Iterate Through All Open Documents	30
5.4. List Fonts And Other Screen Information	30
5.4.1. Display supported fonts	32
5.5. Set the default font using the ConfigurationProvider	32
5.6. Print Current Document	32
5.6.1. Print Current Page	34
5.6.2. Other Printing Arguments	34
5.6.3. Landscape	34
5.7. Configuration information	34
5.7.1. OOo version	34
5.7.2. OOo Locale	35
5.8. Open And Close Documents (And The Desktop)	36
5.8.1. Close OpenOffice And/Or Documents	36
5.8.1.1. What if the file is modified?	37
5.8.2. Load A Document From A URL	37
5.8.2.1. A complete example	39
5.8.3. Save a document with a password	40
5.8.4. Create a new document from a template	40
5.8.5. How do I Enable Macros With LoadComponentFromURL	41
5.8.6. Error handling on load	42
5.8.7. Mail Merge example, merge all documents in a directory	42
5.9. Loading/Inserting an image into your document	44
5.9.1. Danny Brewer embeds an image	45
5.9.2. Embed an image using a dispatch	46
5.9.3. Embed an image directly	47
5.9.3.1. Guess image size	47
5.9.3.2. Embed image	48
5.9.3.3. Embed multiple images	49
5.9.4. Duplicate an existing image	50
5.9.5. Convert all linked images	51
5.9.6. Access OOo's internal storage system	53
5.10. Setting Margins	53
5.10.1. Setting the paper size	54
5.11. Calling an external program (Internet Explorer) using OLE	54
5.12. Use the Shell command for files containing spaces	55
5.13. Read And Write Number In File	55
5.14. Create Number Format Style	56
5.14.1. View Supported Number Format Styles	57
5.15. Return the Fibonacci array	58
5.16. Insert Text At Bookmark	59
5.17. Saving And Exporting A Document	59

5.17.1. Filter names	60
5.18. User Fields	61
5.18.1. Document Information	61
5.18.2. Text Fields	62
5.18.3. Master Fields	63
5.18.4. Removing Text Fields	68
5.18.5. Insert a URL into a Calc cell	68
5.18.6. Find a URL in a Calc cell	69
5.18.7. Adding a SetExpression TextField	70
5.19. User Defined Data Types	71
5.20. Spell Check, Hyphenation, and Thesaurus	71
5.21. Changing The Mouse Cursor	73
5.22. Setting The Page Background	74
5.23. Manipulating the clipboard	74
5.23.1. Copy Spreadsheet Cells With The Clipboard	74
5.23.2. Copy Spreadsheet Cells Without The Clipboard	75
5.23.3. Getting the content-type of the Clipboard	76
5.23.4. Storing a string to the clipboard	77
5.23.5. View the clipboard as text	78
5.23.6. An alternative to the clipboard – transferable content	79
5.24. Setting The Locale (Language)	79
5.25. Setting the locale for selected text	80
5.26. Auto Text	82
5.27. Decimal Feet To Fraction	85
5.27.1. Convert number to words	91
5.28. Sending Email	97
5.29. Macro libraries	98
5.29.1. The vocabulary	99
5.29.1.1. Library container	99
5.29.1.2. Libraries	99
5.29.1.3. Modules	99
5.29.2. Where are libraries stored?	99
5.29.3. The library container	100
5.29.4. Warning about unpublished services	101
5.29.5. What does it means to load a Library?	101
5.29.6. Distribute/deploy a library	101
5.30. Setting Bitmap Size	102
5.30.1. Insert, size, and position a graphic in a Calc document	104
5.30.2. Insert image into a text table cell	105
5.30.3. Export an image at a specified size	108
5.30.4. Draw a Line in a Calc Document	109
5.31. Extracting a Zip File	110
5.31.1. Another Zip File Example	111
5.31.2. Zip Entire Directories	113

5.32. Run a macro by string name	116
5.32.1. Run a macro from the command line	117
5.32.2. Run a named macro in a document	117
5.33. Using a “default application” to open a file	118
5.34. Listing Fonts	118
5.35. Get the document URL, filename, and directory	119
5.36. Get and set the current directory	119
5.37. Writing to a file	121
5.38. Parsing XML	122
5.39. Manipulating Dates	125
5.40. Is OpenOffice embedded into a web browser?	127
5.41. Focus (bring to the front) a new document	127
5.42. What is the document type (based on URL)	128
5.43. Connect to a remote OOO server using Basic	128
5.44. Toolbars	129
5.44.1. Create a toolbar for a component type	131
5.44.1.1. My first toolbar	132
5.45. Load hidden then setVisible	136
5.46. Extension Manager	136
5.47. Embed data in a document	137
5.48. Toggle design mode	137
6. Calc macros	141
6.1. Is this a spreadsheet document?	141
6.2. Display cell value, string, or formula	141
6.3. Set cell value, format, string, or formula	142
6.3.1. Reference a cell in another document	142
6.4. Clear a cell	142
6.5. Selected text, what is it?	143
6.5.1. Simple example processing selected cells	144
6.5.2. Get the active cell and ignore the rest	146
6.5.3. Select a Cell	147
6.6. Human readable address of cell	148
6.7. Insert formatted date into cell	149
6.7.1. A shorter way to do it	150
6.8. Display selected range in message box	150
6.9. Fill selected range with text	151
6.10. Some stats on a selected range	151
6.11. Database range	152
6.11.1. Set selected cells to a database range	153
6.11.2. Delete database range	153
6.12. Table borders	154
6.13. Sort range	155
6.14. Display all data in a column	156
6.15. Using Outline (Grouping) Methods	157

6.16.	Protecting your data	157
6.17.	Setting header and footer text	158
6.18.	Copying spreadsheet cells	159
6.18.1.	Copy entire sheet to a new document	159
6.19.	Select a named range	160
6.19.1.	Select an entire column	162
6.19.2.	Select an entire row	162
6.20.	Convert data in column format into rows	162
6.21.	Toggle Automatic Calculation	164
6.22.	Which cells are used in a sheet?	164
6.23.	Searching a Calc document	165
6.24.	Print a Calc range	168
6.25.	Is a cell merged?	169
6.26.	Write your own Calc functions	170
6.26.1.	User defined Calc functions	170
6.26.2.	Evaluating the argument	171
6.26.3.	What is the return type	172
6.26.4.	Do not modify other cells in the sheet	172
6.26.5.	Add digits in a number	173
6.27.	Add a chart	173
6.28.	Use FunctionAccess to call Calc Functions	174
7.	Writer Macros	177
7.1.	Selected Text, What Is It?	177
7.1.1.	Is the cursor in a text table?	177
7.1.2.	Can I check the current selection for a TextTable or Cell?	178
7.2.	Text Cursors, What Are They?	178
7.2.1.	You can not move a cursor to a TextTable anchor	179
7.2.2.	Inserting something before (or after) a text table	181
7.2.3.	You can move a cursor to a Bookmark anchor	181
7.2.4.	Insert Text At Bookmark	182
7.3.	Andrew's Selected Text Framework	182
7.3.1.	Is Text Selected?	183
7.3.2.	How To Get A Selection	184
7.3.3.	Selected Text, Which End Is Which	184
7.3.4.	The Selected Text Framework Macro	186
7.3.4.1.	The Rejected Framework	186
7.3.4.2.	The Accepted Framework	187
7.3.4.3.	The Main Worker	188
7.3.5.	Counting Sentences	189
7.3.6.	Remove Empty Spaces And Lines, A Larger Example	189
7.3.6.1.	Define "White Space"	189
7.3.6.2.	Rank Characters For Deletion	190
7.3.6.3.	The Standard Selected Text Iterator	191
7.3.6.4.	The Worker Macro	191

7.3.7. Removing Empty Paragraphs, Yet Another Example	192
7.3.8. Selected Text, Timing Considerations And Counting Words	193
7.3.8.1. Searching Selected Text To Count Words	193
7.3.8.2. Using Strings To Count Words	193
7.3.8.3. Using A Character Cursor To Count Words	196
7.3.8.5. Using A Word Cursor To Count Words	197
7.3.8.6. Final Thoughts On Counting Words And Timing	198
7.3.9. Counting Words, You Should Use This Macro!	198
7.4. Replacing Selected Space Using Strings	202
7.4.1. Compare Cursors And String Examples	204
7.5. Setting Text Attributes	215
7.6. Insert text	216
7.6.1. Insert new paragraph	216
7.7. Fields	217
7.7.1. Insert a formatted date field into a Write document	217
7.7.2. Inserting a Note (Annotation)	218
7.8. Inserting A New Page	218
7.8.1. Removing Page Breaks	219
7.9. Set the document page style	220
7.10. Toggle a header or footer on or off	220
7.11. Insert An OLE Object	220
7.12. Setting Paragraph Style	221
7.13. Create Your Own Style	222
7.14. Search And Replace	222
7.14.1. Replacing Text	222
7.14.2. Searching Selected Text	223
7.14.3. Complicated Search And Replace	224
7.14.4. Search and Replace with Attributes and Regular Expressions	226
7.14.5. Search only the first text table	227
7.15. Changing The Case Of Words	228
7.16. Traverse paragraphs (text cursor behavior)	230
7.16.1. Formatting macro paragraphs (an example)	231
7.16.2. Is the cursor in the last paragraph	234
7.16.3. What does it mean to enumerate text content?	234
7.16.4. Enumerating text and finding text content	237
7.16.5. But I only want to find the graphics objects	239
7.16.6. Find a text field contained in the current paragraph?	240
7.17. Where is the Display Cursor?	242
7.17.1. Which cursor should I use to delete a line or a paragraph	245
7.17.2. Delete the current page	246
7.18. Insert an index or table of contents	247
7.19. Inserting a URL into a Write document	248
7.20. Sorting Text	248
7.21. Outline numbering	249

7.22.	Configure Outline numbering	251
7.23.	Numbering paragraphs – not outline numbering	252
7.24.	Insert a table of contents (TOC) or other index	253
7.25.	Text sections	254
7.25.1.	Insert a text section, setting columns and widths	255
7.26.	Footnotes and Endnotes	256
7.27.	Redlines	257
7.28.	Formulas	258
7.29.	Cross references	262
7.29.1.	Enumerate text fields	262
7.29.2.	Enumerate text content	264
7.29.3.	Removing references	268
7.29.4.	Adding references	268
8.	Text tables	269
8.1.	Finding text tables	269
8.1.1.	Where is the text table	269
8.1.2.	Enumerating text tables	271
8.2.	Enumerating cells in a text table	272
8.2.1.	Simple text tables	272
8.2.2.	Formatting a simple text table	273
8.2.3.	What is a complex text table	275
8.2.4.	Enumerating cells in any text table	276
8.3.	Getting data from a simple text table	277
8.4.	Table cursors and cell ranges	278
8.5.	Cell ranges	278
8.5.1.	Using a cell range to clear cells	278
8.6.	Chart data	278
8.7.	Column Widths	279
8.8.	Setting the optimal column width	279
8.9.	How wide is a text table?	280
8.10.	The cursor in a text table	280
8.10.1.	Move the cursor after a text table	282
8.11.	Creating a table	283
8.12.	A table with no borders	284
9.	MacroFormatterADP - Colorize code and XML	285
9.1.	Strings Module	285
9.1.1.	Special characters and numbers in strings	288
9.1.2.	Arrays of strings	293
9.2.	Utilities Module	296
9.2.1.	Where does the code start?	296
9.2.2.	Stacks	297
9.2.3.	Set a character style	298
9.2.4.	Create a property	299
9.2.5.	Find a text document	300

9.3. Styles Module	301
9.3.1. Create character styles	301
9.3.2. Create paragraph styles	305
9.4. Basic Module	310
9.4.1. Use the macros	312
9.4.2. The worker macro	314
9.5. Java Module	317
9.6. Cpp Module	323
9.7. XML Module	326
10. Forms	337
10.1. Introduction	337
10.2. Dialogs	337
10.2.1. Controls	338
10.2.2. Control Label	339
10.2.3. Control Button	339
10.2.4. Text Box	339
10.2.5. List Box	339
10.2.6. Combo Box	340
10.2.7. Check Box	340
10.2.8. Option/Radio Button	340
10.2.9. Progress Bar	340
10.3. Obtaining Controls	341
10.3.1. Size and location of a Control by name	341
10.3.2. Which control called a handler and where is it located?	342
10.4. Choosing a File Using the File Dialog	343
10.5. Center a dialog on the screen	344
10.5.1. DisplayAccess tells you all about the screen	345
10.6. Set the event listener for a control	346
10.7. Controlling a dialog I did not create	347
10.7.1. Inserting a formula	348
10.7.2. Discovering the accessible content (by Andrew)	350
10.7.3. Manipulating the Options dialog	353
10.7.4. Listing the supported printers	355
10.7.5. Finding an open window	357
10.7.6. Inspecting accessible content (by ms777)	358
11. XForms	363
12. Database	365
13. Investment example	367
13.1. Internal Rate of Return (IRR)	367
13.1.1. Using only simple interest	367
13.1.2. Compound the interest	369
14. Handlers and Listeners	371
14.1. Warning, your handler may disappear	371
14.2. xKeyHandler example	371

14.3.	Listener Write-Up by Paolo Mantovani	374
14.3.1.	The CreateUnoListener function	374
14.3.2.	Nice, but what does it do?	374
14.3.3.	How do I know what methods to create?	375
14.3.4.	Example 1: com.sun.star.view.XSelectionChangeListener	376
14.3.5.	Example 2: com.sun.star.view.XPrintJobListener	377
14.3.6.	Example 3: com.sun.star.awt.XKeyHandler	379
14.3.6.1.	Andrew has a little something to add	380
14.3.6.2.	A note about key modifiers (Ctrl and Alt keys)	381
14.3.7.	Example 4: com.sun.star.awt.XMouseClickedHandler	381
14.3.8.	Example 5: Manual binding of events	382
14.4.	What happened to my ActiveSheet listener?	383
15.	Impress	385
15.1.	Slide background color	385
16.	Language	387
16.1.	Comments	387
16.2.	Variables	387
16.2.1.	Names	387
16.2.2.	Declaration	387
16.2.3.	Evil Global Variables And Statics	388
16.2.4.	Types	389
16.2.4.1.	Boolean Variables	391
16.2.4.2.	Integer Variables	391
16.2.4.3.	Long Integer Variables	391
16.2.4.4.	Currency Variables	392
16.2.4.5.	Single Variables	392
16.2.4.6.	Double Variables	392
16.2.4.7.	String Variables	392
16.2.5.	Object, Variant, Empty, and Null	392
16.2.6.	Should I Use Object Or Variant	393
16.2.7.	Constants	393
16.2.8.	Arrays	394
16.2.8.1.	Option Base	394
16.2.8.2.	LBound(arrayname[,Dimension])	394
16.2.8.3.	UBound(arrayname[,Dimension])	394
16.2.8.4.	Is This Array Defined	395
16.2.9.	DimArray, Changing The Dimension	395
16.2.10.	ReDim, Changing The Number Of Elements	395
16.2.11.	Testing Objects	397
16.2.12.	Comparison Operators	397
16.3.	Functions and SubProcedures	398
16.3.1.	Optional Parameters	398
16.3.2.	Parameters By Reference Or Value	399
16.3.3.	Recursion	400

16.4. Flow Control	401
16.4.1. If Then Else	401
16.4.2. IIF	401
16.4.3. Choose	401
16.4.4. For...Next	402
16.4.5. Do Loop	403
16.4.6. Select Case	404
16.4.6.1. Case Expressions	404
16.4.6.2. Incorrect Simple Example	404
16.4.6.3. Incorrect Range Example	405
16.4.6.4. Incorrect Range Example	405
16.4.6.5. Ranges, The Correct Way	405
16.4.7. While...Wend	406
16.4.8. GoSub	406
16.4.9. GoTo	407
16.4.10. On GoTo	407
16.4.11. Exit	408
16.4.12. Error Handling	409
16.4.12.1. Specify How To Handle The Error	409
16.4.12.2. Write The Error Handler	409
16.4.12.3. An Example	410
16.5. Miscellaneous	411
17. Compatibility With Visual BASIC	413
17.1. Data types	413
17.2. Variables	414
17.3. Arrays	414
17.4. Subroutine and Function Constructs	415
17.5. Operators	415
17.6. Subroutines and Functions	416
17.6.1. Numerical Subroutines and Functions	416
17.7. Compatibility mode and private variables	417
18. Operators and Precedence	419
19. String Manipulations	421
19.1. Remove Characters From String	422
19.2. Replace Text In String	422
19.3. Printing The ASCII Values Of A String	423
19.4. Remove All Occurrences Of A String	423
20. Numeric Manipulations	425
21. Date Manipulations	427
22. File Manipulations	429
23. Operators, Statements, and Functions	431
23.1. Subtraction operator (-)	431
23.2. Multiplication operator (*)	431
23.3. Addition operator (+)	431

23.4. Power operator (^)	432
23.5. Division operator (/)	432
23.6. AND Operator	433
23.7. Abs Function	434
23.8. Array Function	434
23.9. Asc Function	435
23.10. ATN Function	436
23.11. Beep Statement	436
23.12. Blue Function	437
23.13. ByVal Keyword	437
23.14. Call Keyword	438
23.15. CBool Function	438
23.16. CByte Function	439
23.17. CDate Function	440
23.18. CDateFromIso Function	440
23.19. CDateToIso Function	441
23.20. CDbl Function	441
23.21. ChDir statement is deprecated	441
23.22. ChDrive statement is deprecated	442
23.23. Choose Function	442
23.24. Chr Function	443
23.25. CInt Function	444
23.26. CLng Function	444
23.27. Close Statement	445
23.28. Const Statement	446
23.29. ConvertFromURL Function	446
23.30. ConvertToURL Function	447
23.31. Cos Function	447
23.32. CreateUnoDialog Function	448
23.33. CreateUnoService Function	449
23.34. CreateUnoStruct Function	450
23.35. CSng Function	450
23.36. CStr Function	451
23.37. CurDir Function	451
23.38. Date Function	452
23.39. DateSerial Function	452
23.40. DateValue Function	453
23.41. Day Function	454
23.42. Declare Statement	455
23.43. DefBool Statement	455
23.44. DefDate Statement	456
23.45. DefDbl Statement	456
23.46. DefInt Statement	457
23.47. DefLng Statement	457

23.48.	<u>DefObj Statement</u>	458
23.49.	<u>DefVar Statement</u>	458
23.50.	<u>Dim Statement</u>	458
23.51.	<u>DimArray Function</u>	459
23.52.	<u>Dir Function</u>	460
23.53.	<u>Do...Loop Statement</u>	462
23.54.	<u>End Statement</u>	462
23.55.	<u>Environ Function</u>	463
23.56.	<u>EOF Function</u>	464
23.57.	<u>EqualUnoObjects Function</u>	465
23.58.	<u>EQV Operator</u>	465
23.59.	<u>Erl Function</u>	466
23.60.	<u>Err Function</u>	467
23.61.	<u>Error statement does not work as indicated</u>	467
23.62.	<u>Error Function</u>	468
23.63.	<u>Exit Statement</u>	468
23.64.	<u>Exp Function</u>	469
23.65.	<u>FileAttr Function</u>	470
23.66.	<u>FileCopy Statement</u>	471
23.67.	<u>FileDateTime Function</u>	471
23.68.	<u>FileExists Function</u>	472
23.69.	<u>FileLen Function</u>	472
23.70.	<u>FindObject Function</u>	473
23.71.	<u>FindPropertyObject Function</u>	474
23.72.	<u>Fix Function</u>	475
23.73.	<u>For...Next Statement</u>	475
23.74.	<u>Format Function</u>	476
23.75.	<u>FreeFile Function</u>	478
23.76.	<u>FreeLibrary Function</u>	478
23.77.	<u>Function Statement</u>	479
23.78.	<u>Get Statement</u>	479
23.79.	<u>GetAttr Function</u>	481
23.80.	<u>GetProcessServiceManager Function</u>	482
23.81.	<u>GetSolarVersion Function</u>	482
23.82.	<u>GetSystemTicks Function</u>	483
23.83.	<u>GlobalScope Statement</u>	484
23.84.	<u>GoSub Statement</u>	484
23.85.	<u>GoTo Statement</u>	485
23.86.	<u>Green Function</u>	486
23.87.	<u>HasUnoInterfaces Function</u>	486
23.88.	<u>Hex Function</u>	487
23.89.	<u>Hour Function</u>	488
23.90.	<u>If Statement</u>	488
23.91.	<u>IIF Statement</u>	489

23.92.	Imp Operator	489
23.93.	Input Statement	490
23.94.	InputBox Function	491
23.95.	InStr Function	492
23.96.	Int Function	493
23.97.	IsArray Function	493
23.98.	IsDate Function	494
23.99.	IsEmpty Function	494
23.100.	IsMissing Function	495
23.101.	IsNull Function	495
23.102.	IsNumeric Function	496
23.103.	IsObject Function	497
23.104.	IsUnoStruct Function	497
23.105.	Kill Function	498
23.106.	LBound Function	498
23.107.	LCase Function	499
23.108.	Left Function	499
23.109.	Len Function	500
23.110.	Let Function	500
23.111.	Line Input Statement	501
23.112.	Loc Function	501
23.113.	Lof Function	501
23.114.	Log Function	502
23.115.	Loop Statement	503
23.116.	LSet Statement	504
23.117.	LTrim Function	505
23.118.	Private Keyword	505
23.119.	Public Keyword	506
23.120.	Red Function	506
23.121.	RSet Statement	507
23.122.	Shell Function	507
23.123.	UBound Function	510
23.124.	UCase Function	510
23.125.	URL Notation And Filenames	511
23.125.1.	URL Notation	511
23.125.2.	Paths With Spaces And Other Special Characters	511
23.125.3.	Anchoring To The Home Directory On Unix	511
24.	Other languages	513
24.1.	C#	513
24.2.	Visual Basic Programmers	514
24.2.1.	ActiveWorkbook	515
24.2.2.	ActiveSheet and ActiveCell	515
25.	Index	517

1. Introduction

This is “Andrew's Macro Document”, the free document that started before I wrote my published book. The book contains excellent material for beginners, with complete tested examples, reference material, and figures. The book is a reference for the supported Basic commands. This document, on the other hand, covers fewer commands and is not as accurate or detailed. In other words, consider obtaining a copy of my published book “OpenOffice.org Macros Explained” (see <http://www.pitonyak.org/book/>). Then again, I suppose that you could also just send me money by PayPal (andrew@pitonyak.org) to support the development of this document!

When I wrote my first macro for OpenOffice, I was overwhelmed by the complexity. I started a list of macros that accomplished simple tasks. I started creating macros requested by the user community. My quest to understand OOo macros became this document.

This document is frequently updated and available from my web site:
<http://www.pitonyak.org/AndrewMacro.odt>

The Last Modified time is found on the title page. My web page also indicates the last uploaded date and time. Although the template used to create this document is not required, it is available on my web site.

1.1. Special thanks to Bartosz

Bartosz, who lives in Poland, fixed a long standing bug that crashed large documents (such as this document). The fix was put in place around OOo version 3.4. If you use an earlier version of OOo, then OOo will crash when you close this document. Please note that the fix was not trivial and a great deal of skill and effort went into finding a suitable fix for this bug. Thank you for your work!

1.2. Abbreviations and definitions

OpenOffice.org is frequently abbreviated as OOo. OOo Basic is the name of the macro language included with OpenOffice. OOo Basic is similar to Visual Basic so knowledge of Visual Basic is a great advantage.

Table 1.1. Common abbreviations and definitions.

Abbreviation	Definition
GUI	Graphical User Interface
IDE	Integrated Development Environment
UNO	Universal Network Objects
IDL	Interface Definition Language
SDK	Software Development Kit
OLE	Object Linking and Embedding
COM	Component Object Model
OOo	OpenOffice.org

2. Available Resources

2.1. Included Material

Use **Help | OpenOffice.org Help** to open the OOo help pages. OOo contains help for Writer, Calc, Base, Basic, Draw, Math, and Impress. The upper left corner of the OOo help system contains a drop down list that determines which help set is displayed. To view the help for Basic, the drop down must display “Help about OpenOffice.org Basic”.



Figure 2.1: OOo help pages for Basic.

Many excellent macros are included with OOo. For example, I found a macro that prints the property and method names for an object. I used these methods before I wrote my own object inspector. Use **Tools | Macros | Organize Macros | OpenOffice.org Basic** to open the Macro dialog. Expand the Tools library in the OpenOffice.org library container. Look at the Debug module – some good examples include `WritedbInfo (document)` and `printdbgInfo (sheet)`.

TIP Before running a macro, the library that contains the macro must be loaded. The first chapter of my book, which discusses libraries in depth, is available as a free download (see <http://www.pitonyak.org/book/>). This is a great introduction for a beginner and it is free. This same chapter is available in updated form from the OOo Authors web site.

2.2. On line Resources

The following links and references help decrypt the initially difficult paradigm:

- <http://www.openoffice.org> (Apache OpenOffice [AOO], main link)
- <http://www.libreoffice.org/> (LibreOffice [LO], main link).
- <https://forum.openoffice.org/> (a well supported forum with help).
- <http://www.oooforum.org> (This used to be well supported forum but lately it has been filled with spam if it is up at all).
- <http://www.openoffice.org/api/docs/common/ref/com/sun/star/module-ix.html> (this is the official IDL reference, here you'll find almost every command with a description)

- https://wiki.openoffice.org/wiki/Documentation/DevGuide/OpenOffice.org_Developers_Guide (The second official Documentation, explains everything in detail on almost 1000 pages!)
- <http://www.pitonyak.org/oo.php> (My web site).
- <http://www.pitonyak.org/AndrewMacro.odt> (Latest copy of this document)
- http://www.pitonyak.org/OOME_3_0.odt (copy of my book)
- <http://docs.sun.com/app/docs> (Sun wrote a book on macro programming. Very well written and laid out, search for StarOffice)
- <http://api.openoffice.org/basic/man/tutorial/tutorial.pdf> (Excellent)
- <http://www.openoffice.org/api/> (This site takes some getting used to but it is very complete)
- <http://www.openoffice.org/documentation/> Download the “How To” document referenced below. http://documentation.openoffice.org/HOW_TO/various_topics/How_to_use_basic_macros.sxw
- <http://www.openoffice.org/udk/> (Here you will find advanced information about UNO)
- http://udk.openoffice.org/common/man/tutorial/office_automation.html (OLE)
- <http://kienlein.com/pages/oo.html> (Examples)
- <http://www.kargs.net/openoffice.html> (Examples)
- <http://homepages.paradise.net.nz/hillview/OOo/> (numerous excellent macros here including reveal codes macros, key macros, and, information on converting from MS Office)
- <http://api.libreoffice.org/> API docs for LO.
- <http://api.libreoffice.org/docs/idl/ref/annotated.html> See LO IDL references.

To find detailed specific information, search the Developer's Guide or perform a web search such as “cursor OpenOffice”. To limit the search, use the following search: “site:api.openoffice.org cursor”.

If I know the package name, I can usually guess the web location. I inspect the API web site at least as often as I inspect objects.

<http://api.openoffice.org/docs/common/ref/com/sun/module-ix.html>

2.3. Translations of this document

Table 2.1. Translations of this document.

Link	Language
http://www.pitonyak.org/AndrewMacro.odt	English

<i>Link</i>	<i>Language</i>
http://fr.openoffice.org/Documentation/Guides/Indexguide.html *	French
http://www.pitonyak.org/AndrewMacro_rus.odt *	Russian
http://www.pitonyak.org/AndrewMacroGerman.sxw *	German

* The translations are not up to date.

3. Getting started: concepts

The first chapter of my book, which is available as a free download, is a good “getting started” chapter. The chapter introduces your first macro and the IDE; I consider it a better place to start (see <http://www.pitonyak.org/book/>).

A Macro is used to automate task in OpenOffice.org. A macro can automate actions that otherwise require error-prone manual labor. Currently, the automatic actions are most easily created by writing Macros in OOo Basic. The scripting framework can ease the use of other languages, but Basic is still the easiest to use. Here are some of the advantages of using the OOo Basic language to control OOo:

- easy to learn,
- supports COM (ActiveX) and advanced GUI features in OpenOffice,
- on line user-community, and
- cross platform solution.

Tip

OpenOffice.org Basic is also known as StarBasic.

3.1. My first macro: “Hello World”

Open a new OOo document. Use **Use Tools | Macros | Organize Macros | OpenOffice.org Basic** to open the Macro dialog. On the left hand side of the dialog, find the document that you just opened. It is probably called “untitled1”. Single click underneath “untitled1” where it says “standard”. Click the “new” button on the far right to create a new module. Using the default name “Module1” is probably not the best choice. when you have multiple documents open and they all have a module named “Module1”, it may become difficult to tell them apart. For now, name your first module “MyFirstModule”. The OOo Basic IDE will open. Enter the code shown in Listing 3.1.

***Listing 3.1:** Your first simple macro, “Hello world”.*

```
Sub Main
  Print "Hello World"
End Sub
```

Click the run button in the tool-bar to run your first OOo Basic macro.

3.2. Grouping Code

OOo Basic is based on subroutines and functions, which are defined using the key words Sub and Function – I interchangeably refer to these as procedures, routines, subroutines, or functions. A routine can call other routines. The difference between a Sub and a Function, is that a Function returns a value and a Sub does not. In Listing 3.2, the subroutine HelloWorld calls the function HelloWorldString to obtains the text “Hello World”, which is printed.

Listing 3.2: “Hello world” using a subroutine and a function.

```
Sub HelloWorld
    Dim s As String
    s = HelloWorldString()
    MsgBox s
End Sub

Function HelloWorldString() As String
    HelloWorldString = "Hello World"
End Function
```

Macros are stored as shown in Figure 3.1.

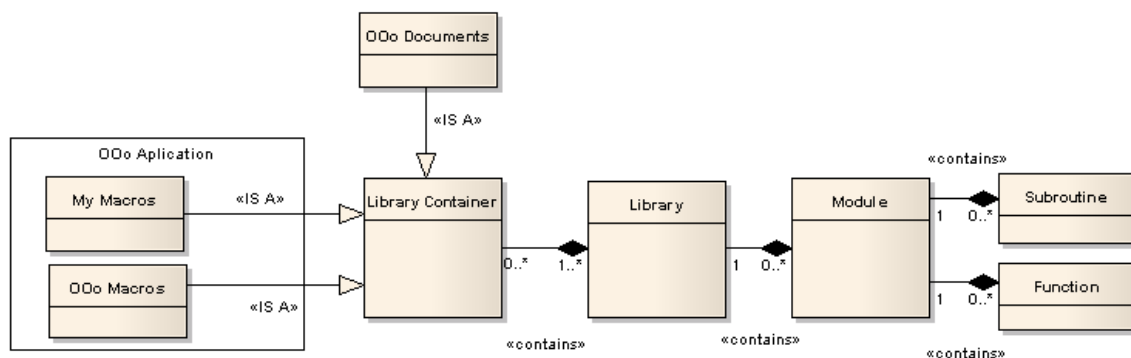


Figure 3.1. How macros are stored.

Do not be fooled by Figure 3.1, it is really very simple. Reading figure from right to left:

- Modules contain (or are composed of) subroutines and functions. The 1 indicates that each routine is contained in exactly one module. The 0..* indicates that each module contains zero or more routines.
- Each library contains zero or more modules, and each module can live in only one library.
- Each library container contains one or more libraries. Every library contains the Standard library, which cannot be removed, cannot be over-written, and is always loaded so the contained macros are available when the library container is available. A library can be contained in zero or more library containers. A library can be stored in a directory on disk by itself, so it is possible that it is not contained in any library container. When a library is imported into a library container, you can choose to copy the library, or link to the library. It is possible for multiple library containers to link to the same library.
- OOo documents are library containers, so documents can contain libraries. Libraries contained in documents are only available from those documents.

- The OOo application has two library containers. One library container contains “my macros” that I write. There are also OOo macros that are included with OOo; these are stored in the OOo library container.

3.3. Debugging

The IDE contains debugging capabilities, such as setting break points and watch variables. You can also single step through your code. It is useful to set the breakpoint before the suspected error and then single step through the code to see how the error happens.

3.4. Variables

A variable is similar to a box that contains something. Like a box, some variables are more suited to contain certain type of data. A variable's type determines what it can store. Attempting to store the wrong type of data in a variable, will frequently cause an error. Use the DIM statement to “dimension” (declare) a variable before you use it.

Listing 3.3: Declare a simple variable.

```
Dim <variablename> As <Type>
```

A variable whose content can not be changed is called a Constant. A constant is declared using the Const keyword.

Listing 3.4: Declare a constant.

```
Const <constantname> = <constantvalue>
```

OOo Basic supports many data types. Strings are simple text values delimited by double quotation marks. For numbers, both integer and floating point data types are supported. To learn more about variables, refer to section 16 Language on page 403 and also refer to the OOo Help. You can also refer to my book “OpenOffice.org Macros Explained”.

3.5. Accessing And Creating Objects In OpenOffice

OpenOffice.org implements a large number of Services (Objects); usually, services are easily available. Access the current document and the desktop using the global variables ThisComponent and StarDesktop respectively – both of these global variables represent an Object. When you have a document, you can access its interface (see Listing 3.5).

Listing 3.5: Declare and use some variables.

```
Sub Example
    Dim oDoc As Variant    ' Reference the active document.
    Dim oText As Variant   ' Reference the document's main Text object.
    oDoc = ThisComponent   ' Get the active document
    oText = oDoc.GetText() ' Get the TextDocument service
End Sub
```

In OOo 2.x, ThisComponent never refers to a Base document and a Base document is not a library container; macros are stored in the contained forms. Starting with version 3.0,

ThisComponent will always be the component that was active when the macro was invoked. This holds if the macro is located in a Base document or in the OOo library container. Also, it holds if the active component is a Base document or one of its sub components; a form, for example.

The variable ThisDatabaseDocument will be introduced for basic macros embedded in a Base document, and always refers to the Base document.

To create an instance of a service, use the global method createUnoService() as shown in Listing 3.6. This also demonstrates how to create a structure.

Listing 3.6: *This is the old way of executing a dispatch.*

```
Sub PerformDispatch(vObj, uno$)
    Dim vParser      ' This will reference a URLTransformer.
    Dim vDisp        'Return value from the dispatch.
    Dim oUrl As New com.sun.star.util.URL    'Create a Structure

    oUrl.Complete = uno$
    vParser = createUnoService("com.sun.star.util.URLTransformer")
    vParser.parseStrict(oUrl)

    vDisp = vObj.queryDispatch(oUrl, "", 0)
    If (Not IsNull(vDisp)) Then vDisp.dispatch(oUrl, noargs())
End Sub
```

Tip

Although you can create a desktop instance as shown below, you should use the global variable StarDesktop instead. You only need to create the desktop in languages other than StarBasic.

```
createUnoService("com.sun.star.frame.Desktop")
```

3.6. Everyone keeps talking about UNO, what is it?

UNO (Universal Network Object) was created to allow different environments to interact. Why? Because different programming languages and environments may have a different representation for the same data. Even integers, as simple as they are, can be represented differently on different computers and programming languages.

3.6.1. Simple types

UNO defines numerous basic types such as strings, integers, etc. (so they are the same in different environments). When you use simple types in OOo Basic, everything is automatic. In other words, don't worry about it, just use it.

3.6.2. Object

An object is a more complicated type that may contain properties and/or methods. A method is similar to a subroutine or a function; some return values, some do not. For example, the following line calls the `getCurrentController` method of `ThisComponent`.

```
ThisComponent.getCurrentController()
```

The current controller object is also available as a property without calling a method

```
ThisComponent.CurrentController
```

3.6.3. Structures

A structure is a composite type that comprising a fixed set of labeled properties. In other words, a structure is a simple object that contains properties and not methods. Structures that are defined by OpenOffice.org are testable using the method `IsUNOStruct()`.

Listing 3.7: Test for an UNO structure.

```
Sub ExamineStructures
  Dim oProperty As New com.sun.star.beans.PropertyValue
  With oProperty
    .name = "Joe"
    .value = 17
  End With
  Print oProperty.Name & " is " & oProperty.Value
  If IsUNOStruct(oProperty) Then
    Print "oProperty is an UNO Structure"
  End If
End Sub
```

You can define your own structures, but they are not UNO structures so `IsUNOStruct()` returns False for them.

3.6.4. Interfaces

In OOO, an interface defines zero or more methods. An interface can inherit from zero or more interfaces. An object that implements an interface, must support the methods defined by the interface and all interfaces that it inherits (see Figure 3.2).

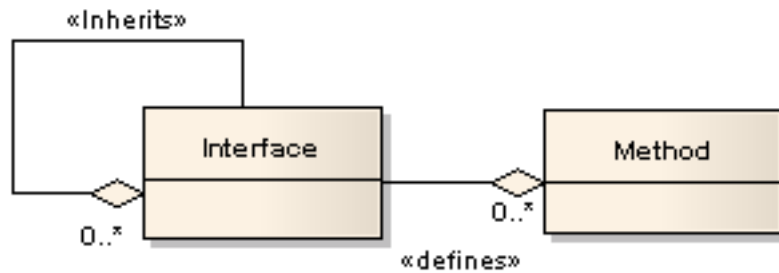


Figure 3.2. Interfaces define methods.

Almost all UNO objects support services and interfaces. When I use the word interface, as it relates to an object, the word always means the set of methods that an object supports. For example, if an object supports the `com.sun.star.frame.XStorable` interface, or `XStorable`, then the object supports the methods in *Table 3.1*.

Table 3.1. Methods defined by the `XStorable` interface.

Method	Summary
<code>hasLocation</code>	Returns true if the object has a know the location, either because it was loaded from there, or because it is stored there.
<code>getLocation</code>	Returns the URL the object was stored to.
<code>isReadOnly</code>	If true, do not call <code>store()</code> .
<code>store</code>	Store the data to the URL from which it was loaded.
<code>storeAsURL</code>	Store the object to a URL. Subsequent calls to <code>store</code> use this URL.
<code>storeToURL</code>	Store the object to a URL, this does not change the document's URL.

The macro in Listing 3.19 checks to see if the component supports the `XStorable` interface. If it does, then the macro uses `hasLocation()` and `getLocation()`. It is possible that a returned component will not support the `XStorable` interface or that the document has not been saved and therefore does not have a location to print.

3.6.5. Services

In OOO, a service defines zero or more properties. A service can inherit from zero or more services, and zero or more interfaces. An object that supports a service, must support the methods and properties defined by service and all inherited interfaces and services (see Figure 3.3).

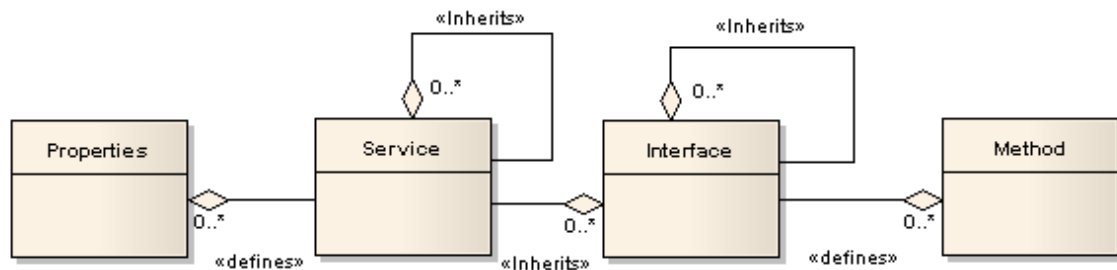


Figure 3.3. Services define properties and methods.

When I use the word service, as it relates to an object, I always mean a set of interfaces, properties, and services that an object supports. Given a specific interface or service, you can find the definition of that interface or service at <http://api.openoffice.org>. The service definition defines only the objects immediately defined by the service. For example, the TextRange service is defined as supporting the CharacterProperties service, which is defined to support individual properties such as the CharFontName. An object that supports the TextRange service, therefore, supports the CharFontName property even though it is not explicitly listed; you need to explore all of the listed interfaces, services, and properties to see what an object really supports.

3.6.6. Interfaces and services

Although the definition of services and interfaces are available, the definition is a poor method for quickly determining everything about an object. You can set break points in the BASIC IDE and inspect variables. This is useful to see the properties. XRay is a popular object inspection library. Download and install Xray from OOo Macros (see <http://bernard.marcelly.perso.sfr.fr/index2.html>). I wrote my own object inspection macros before XRay was available.

Most UNO objects implements the XServiceInfo interface, which allows you to ask the object if it supports a specific service.

Listing 3.8: Verify that a document is a text document.

```

Dim s As String
s = "com.sun.star.text.TextDocument"
If ThisComponent.supportsService(s) Then
    Print "The document is a text document"
Else
    Print "The document is not a text document"
End If
  
```

3.6.7. What type is this object?

It is useful to know the object type so that you know what you can do with it. Here is a brief list of inspection methods:

Table 3.2: Methods used to inspect variables.

Method	Description
IsArray	Is the parameter an array?
IsEmpty	Is the parameter an uninitialized variant type?
IsNull	Does the parameter contain “no” data?
IsObject	Is the parameter an OLE object?
IsUnoStruct	Is the parameter an UNO structure?
TypeName	What is the type name of the parameter?

A variables type name can also provide information about a variables properties.

Table 3.3: Values returned by the `TypeName()` statement.

Type	TypeName()
Variant	“Empty” or name of contained object
Object	“Object” even if it is null. Same for structures.
regular type	regular type name such as “String”
array	name followed by “()”

Listing 3.9 demonstrates the statements in Table 3.2 for different variable types.

Listing 3.9: Inspect variables.

```

Sub TypeTest
    Dim oSFA
    Dim aProperty As New com.sun.star.beans.Property
    oSFA = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
    Dim v, o As Object, s As String, ss$, a(4) As String
    ss = "Empty Variant: " & GetSomeObjInfo(v) & chr(10) & _
        "Empty Object: " & GetSomeObjInfo(o) & chr(10) & _
        "Empty String: " & GetSomeObjInfo(s) & chr(10)
    v = 4
    ss = ss & "int Variant: " & GetSomeObjInfo(v) & chr(10)
    v = o
    ss = ss & "null obj Variant: " & GetSomeObjInfo(v) & chr(10) & _
        "struct: " & GetSomeObjInfo(aProperty) & chr(10) & _
        "service: " & GetSomeObjInfo(oSFA) & chr(10) & _
        "array: " & GetSomeObjInfo(a())
    MsgBox ss, 64, "Type Info"
End Sub

```

REM Returns basic type information for the parameter.
 REM This also returns the dimensions of an array.

```

Function GetSomeObjInfo(vObj) As String
    Dim s As String
    s = "TypeName = " & TypeName(vObj) & CHR$(10) & _
        "VarType = " & VarType(vObj) & CHR$(10)
    If IsNull(vObj) Then
        s = s & "IsNull = True"
    ElseIf IsEmpty(vObj) Then
        s = s & "IsEmpty = True"
    Else
        If IsObject(vObj) Then
            On Local Error GoTo DebugNoSet
            s = s & "Implementation = " & _
                NotSafeGetImplementationName(vObj) & CHR$(10)
            DebugNoSet:
            On Local Error Goto 0
            s = s & "IsObject = True" & CHR$(10)
        End If
        If IsUnoStruct(vObj) Then s = s & "IsUnoStruct = True" & CHR$(10)
        If IsDate(vObj) Then s = s & "IsDate = True" & CHR$(10)
        REM The argument to IsNumeric must be a string.
        If IsNumeric(vObj) Then s = s & "IsNumeric = True" & CHR$(10)
        If IsArray(vObj) Then
            On Local Error Goto DebugBoundsError:
            Dim i%, sTemp$
            s = s & "IsArray = True" & CHR$(10) & "range = ("
            Do While (i% >= 0)
                i% = i% + 1
                sTemp$ = LBound(vObj, i%) & " To " & UBound(vObj, i%)
                If i% > 1 Then s = s & ", "
                s = s & sTemp$
            Loop
            DebugBoundsError:
            On Local Error Goto 0
            s = s & ")" & CHR$(10)
        End If
    End If
    GetSomeObjInfo = s
End Function

REM This places an error handler where it will catch the problem
REM and return something anyway!
Function SafeGetImplementationName(vObj) As String
    On Local Error GoTo ThisErrorHere:
    SafeGetImplementationName = NotSafeGetImplementationName(vObj)
    Exit Function
ThisErrorHere:
    On Local Error GoTo 0

```

```

    SafeGetImplementationName = "*** Unknown ***"
End Function

REM The problem is that if this Function is called and the vObj
REM type does NOT support the getImplementationName() call,
REM then I receive an "Object variable not set" error at
REM the Function definition.
Function NotSafeGetImplementationName(vObj) As String
    NotSafeGetImplementationName = vObj.getImplementationName()
End Function

```

3.6.8. What methods, properties, interfaces, and services are supported?

UNO objects usually support ServiceInfo, which provides information about the service. Use getImplementationName() to obtain the fully qualified object name. Use the fully qualified name to search Google or the Developer's Guide for more information. Listing 3.10 demonstrates how to print the methods, interfaces, and properties for an UNO object.

Listing 3.10: *What can this object do?*

```

MsgBox vObj.dbg_methods           'Methods for this object.
MsgBox vObj.dbg_supportedInterfaces 'Interfaces for by this object.
MsgBox vObj.dbg_properties        'Properties for this object.

```

OOo includes macros to display debug information. The most commonly used included macros are PrintdbgInfo(object) and ShowArray(object). The WritedbgiInfo(object) macro inserts object debug information into an open writer document.

3.6.9. Languages other than Basic

StarBasic provides numerous conveniences not provided by other programming languages. This section touches on only a few of these.

3.6.9.1. CreateUnoService

The CreateUnoService() method is a short cut to obtaining the global service manager and then calling createInstance() on the service manager.

Listing 3.11: *Get the global process service manager.*

```

oManager = GetProcessServiceManager()
oDesk = oManager.createInstance("com.sun.star.frame.Desktop")

```

In StarBasic, this process may be done in a single step – unless you need to use createInstanceWithArguments() that is.

Listing 3.12: *CreateUnoService is less code than using the process service manager.*

```

oDesk = CreateUnoService("com.sun.star.frame.Desktop")

```

Other languages, such as Visual Basic, do not support the CreateUnoService() method.

Listing 3.13: Create a UNO service using Visual Basic.

```
Rem Visual Basic does not support CreateUnoService().
Rem The service manager is always the first thing to create
REM In Visual Basic.
Rem If Oo is not running, it is started.
Set oManager = CreateObject("com.sun.star.ServiceManager")
Rem Create a desktop object.
Set oDesk = oManager.CreateInstance("com.sun.star.frame.Desktop")
```

3.6.9.2. ThisComponent

In StarBasic, ThisComponent references the current the document, or the document that caused a macro to be called. ThisComponent is set when the macro is started, and does not change, even if the macro causes a new document to become current – this includes closing ThisComponent. Even with the pitfalls, ThisComponent is a nice convenience. In languages other than Basic, the common solution is to use the getCurrentComponent() on the desktop object. If the Basic IDE or help window is current, getCurrentComponent() returns these objects, which are not Oo Documents.

3.6.9.3. StarDesktop

StarDesktop references the desktop object which is essentially the primary Oo application. The name originated when the product was named StarOffice and it displayed a main desktop object that contained all of the open components. Examples of components that the desktop object may contain include all supported documents, the BASIC Integrated Development Environment (IDE), and the included help pages (see *Figure 3.4*).

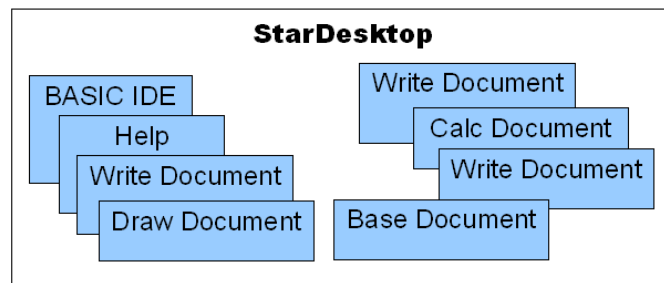


Figure 3.4: The desktop contains components.

Getting back to the macro in *Listing 3.19*, StarDesktop provides access to the currently open components. The method `getCurrentComponent()` returns the currently active component. If the macro is run from the BASIC IDE, then a reference to the BASIC IDE is returned. If the macro is run while a document is displayed, probably by using **Tools > Macros > Run Macro** then `oComp` will reference the current document.

TIP The global variable ThisComponent refers to the currently active document. If a non-

document type component has the focus, then `ThisComponent` refers to the last active document. As of OOO version 2.01, the Basic IDE, help pages, and Base documents do not cause `ThisComponent` to be set to the current component.

3.6.10. Accessing methods and properties

StarBasic automatically makes the methods and properties supported by an object available – StarBasic sometimes makes properties available that are not available using other methods. In other languages, the interface that defines the method that you want to call must be extracted before it can be used (see Listing 3.14).

Listing 3.14: *In Java, you must obtain an interface before you can use it.*

```
XDesktop xDesk;  
xDesk = (XDesktop) UnoRuntime.queryInterface(XDesktop.class, desktop);  
XFrame xFrame = (XFrame) xDesk.getCurrentFrame();  
XDispatchProvider oProvider = (XDispatchProvider)  
UnoRuntime.queryInterface(XDispatchProvider.class, xFrame);
```

If the view cursor is in a text section, the `TextSection` property contains a reference to the text section. If not, the `TextSection` property is null. In StarBasic, I can obtain the text section as follows:

Listing 3.15: *OOo Basic allows you to access properties directly.*

```
If IsNull(oDoc.CurrentController.getViewCursor().TextSection) Then
```

In a language other than StarBasic, the `CurrentController` property and the `TextSection` property are not directly available. The current controller is available using a “get” method, and the text section is available as a property value. Code that uses get and set methods is easier to translate into other languages than code that uses properties.

Listing 3.16: *Access some properties using get methods.*

```
oVCurs = oDoc.getCurrentController().getViewCursor()  
If IsNull(oVCurs.getPropertyValue("TextSection")) Then
```

StarBasic allows some properties to act as an array, even if the property is not an array; specifically, properties that implement an interface for indexed access. Consider the `Sheets` property in a Calc document. In Calc, both of the following accomplish the same task, but only the second example works outside of StarBasic.

Listing 3.17: *OOo Basic allows you to access some properties as an array.*

```
oDoc.sheets(1)           ' Access as an array.  
oDoc.getSheets().getByIndex(1) ' Use a method.
```

3.7. Summary

The StarBasic language refers to the syntax and the commands for general programming. StarBasic is easy to use.

Listing 3.18: Simple macro that does not access the OOO API.

```
Sub SimpleExample()  
    Dim i As Integer  
    i = 4  
    Print "The value of i = " & i  
End Sub
```

Most macros are written to interact with the OOO components. You must, therefore, learn about the methods and properties for each object that you want to use; this is difficult.

Listing 3.19: Simple macro that uses the OOO API to inspect the current component.

```
Sub ExamineCurrentComponent  
    Dim oComp  
    oComp = StarDesktop.getCurrentComponent()  
    If HasUnoInterfaces(oComp, "com.sun.star.frame.XStorable") Then  
        If oComp.hasLocation() Then  
            Print "The current component has URL: " & oComp.getLocation()  
        Else  
            Print "The current component does not have a location."  
        End If  
    Else  
        Print "The current component is not storable"  
    End If  
End Sub
```

Although the macro in *Listing 3.19* is simple, it requires a lot knowledge to write. The macro starts by declaring the variable `oComp`, which defaults to type `Variant` because the type is not explicitly given.

4. Examples

4.1. Debugging And Inspecting Macros

It can be difficult to determine what methods and properties are available for an object. The methods in this section should help.

4.1.1. Determine Document Type

In Oo, most of the functionality is defined by services. To determine the document type, look at the services it supports. The macro shown below uses this method. I assume that this is safer than using `getImplementationName()`.

Listing 4.1: Identify most OpenOffice.org document types.

```
'Author: Included with OpenOffice
'Modified by Andrew Pitonyak
Function GetDocumentType(oDoc)
    Dim sImpress$
    Dim sCalc$
    Dim sDraw$
    Dim sBase$
    Dim sMath$
    Dim sWrite$

    sCalc    = "com.sun.star.sheet.SpreadsheetDocument"
    sImpress = "com.sun.star.presentation.PresentationDocument"
    sDraw     = "com.sun.star.drawing.DrawingDocument"
    sBase     = "com.sun.star.sdb.DatabaseDocument"
    sMath     = "com.sun.star.formula.FormulaProperties"
    sWrite    = "com.sun.star.text.TextDocument"

    On Local Error GoTo NODOCUMENTTYPE
    If oDoc.SupportsService(sCalc) Then
        GetDocumentType() = "scalc"
    ElseIf oDoc.SupportsService(sWrite) Then
        GetDocumentType() = "swriter"
    ElseIf oDoc.SupportsService(sDraw) Then
        GetDocumentType() = "sdraw"
    ElseIf oDoc.SupportsService(sMath) Then
        GetDocumentType() = "smath"
    ElseIf oDoc.SupportsService(sImpress) Then
        GetDocumentType() = "simplode"
    ElseIf oDoc.SupportsService(sBase) Then
        GetDocumentType() = "sbase"
    End If
    NODOCUMENTTYPE:
    If Err <> 0 Then
        GetDocumentType = ""
        Resume GOON
    End If
End Function
```



```

        GOON:
    End If
End Function

```

Listing 4.2 returns the name of the PDF export filter based on the document type.

Listing 4.2: *Use the document type to determine the PDF export filter.*

```

Function GetPDFFilter(oDoc)
    REM Author: Alain Viret [Alain.Viret@bger.admin.ch]
    REM Modified by Andrew Pitonyak
    On Local Error GoTo NODOCUMENTTYPE
    Dim sImpress$
    Dim sCalc$
    Dim sDraw$
    Dim sBase$
    Dim sMath$
    Dim sWrite$

    sCalc      = "com.sun.star.sheet.SpreadsheetDocument"
    sImpress   = "com.sun.star.presentation.PresentationDocument"
    sDraw      = "com.sun.star.drawing.DrawingDocument"
    sBase      = "com.sun.star.sdb.DatabaseDocument"
    sMath      = "com.sun.star.formula.FormulaProperties"
    sWrite     = "com.sun.star.text.TextDocument"

    On Local Error GoTo NODOCUMENTTYPE
    If oDoc.SupportsService(sCalc) Then
        GetPDFFilter() = "calc_pdf_Export"
    ElseIf oDoc.SupportsService(sWrite) Then
        GetPDFFilter() = "writer_pdf_Export"
    ElseIf oDoc.SupportsService(sDraw) Then
        GetPDFFilter() = "draw_pdf_Export"
    ElseIf oDoc.SupportsService(sMath) Then
        GetPDFFilter() = "math_pdf_Export"
    ElseIf oDoc.SupportsService(sImpress) Then
        GetPDFFilter() = "impress_pdf_Export"
    End If
    NODOCUMENTTYPE:
    If Err <> 0 Then
        GetPDFFilter() = ""
        Resume GOON
    GOON:
    End If
End Function

```

4.2. X-Ray

Bernard Marcelly wrote a tool called X-Ray that displays object information in a dialog. X-Ray tool is available for download from <http://bernard.marcelly.perso.sfr.fr/index2.html>, and is highly recommended by many people!

4.3. Dispatch: Using Universal Network Objects (UNO)

<http://udk.openoffice.org> and the Developer's Guide are good references in your quest to understand UNO. UNO is a component model offering interoperability between different programming languages, object models, machine architectures, and processes.

On the Windows platforms, many software packages use the existing COM. OOo, however, has its own multi-platform component object model. By using its own object model, OOo's functionality is not limited to Windows. Secondly, OOo can provide a better error handling system than is provided by COM. Nevertheless, COM can still be used to control OpenOffice (Windows). For more information, see

<http://api.openoffice.org/docs/DevelopersGuide/ProfUNO/ProfUNO.htm#1+4+4+Automation+Bridge>.

This example dispatches an UNO command to perform the “undo” command.

Listing 4.3: Use the new dispatch method to perform the undo operation.

```
Sub Undo
    Dim oDisp
    Dim oFrame
    oFrame = ThisComponent.CurrentController.Frame
    oDisp = createUnoService("com.sun.star.frame.DispatchHelper")
    oDisp.executeDispatch(oFrame, ".uno:Undo", "", 0, Array())
End Sub
```

The difficult part is knowing the UNO interface and the parameters for each. Consider the following which should work with newer versions of OOo.

Listing 4.4: Use the new dispatch method to export a Write document to PDF.

```
Dim a(2) As New com.sun.star.beans.PropertyValue
a(0).Name = "URL" : a(0).Value = "my_file_name_.pdf"
a(1).Name = "FilterName" : a(1).Value = "writer_pdf_Export"
oDisp.executeDispatch(oFrame, ".uno:ExportDirectToPDF", "", 0, a())
```

Listing 4.5: Use the new dispatch method to go to a cell, copy, and paste.

```
Dim a(1) As New com.sun.star.beans.PropertyValue
a(0).Name = "ToPoint" : a(0).Value = "$B$3"
oDisp.executeDispatch(oFrame, ".uno:GoToCell", "", 0, a())
oDisp.executeDispatch(oFrame, ".uno:Copy", "", 0, Array())
oDisp.executeDispatch(oFrame, ".uno:Paste", "", 0, Array())
```

4.3.1. The dispatcher requires a user interface

Hal Vaughan asked questions, Mathias Bauer answered them.

Q: Will Dispatch Names Change?

A: Macros using dispatch names rather than numbers will not change between OOo versions.

Q: Is there any reason to use the regular API calls instead of calling the dispatcher with the function name?

A: Dispatch calls do not work on a document without a UI. If OOo is run in a real "server" mode where documents can be loaded and scripted without any GUI, only macros using the "real" API will work. The real API is also much more powerful and gives you a better insight in the real objects. IMHO you should use the dispatch API only for two reasons:

1. Recording macros
2. As a workaround when a certain task can not be done by any "real" API (because it does not exist or it is broken).

4.3.1.1. Modifying the menu – a dispatcher example

In OOo 2.0, you should be able to assign a macro to a menu item using the API:

http://specs.openoffice.org/ui_in_general/api/ProgrammaticControlOfMenuAndToolbarItems.sxw

The API IDL states that there is an XMenu Interface and also an XMenuListener. Retrieving the Menu ID from the XML file that defines the menu is possible, should be able to assign a macro to that ID ??? test this! You can also use a DispatchProviderInterceptor, which can be compared to a Listener or Handler.

Use a DispatchProviderInterceptor to listen for Dispatch commands. It is possible to intercept almost every command. Using slot ids, we can assign macros to special DispatchCommands that represent menu items. Although we can not prevent a user from selecting a menu item, we can intercept the command when it is dispatched. A DispatchInterceptor must be implemented and registered.

The ToggleToolbarVisibility macro, written by Peter Biela, toggles the tool bar visibility.

Listing 4.6: Toggle the visibility of a tool bar.

```
REM Author: Peter Biela
REM E-Mail: Peter.Biela@planet-interkom.de
REM Modified: Andrew Pitonyak
Sub ToggleToolbarVisibility( )
    Dim oFrame
    Dim oDisp
    Dim a()
    Dim s$
    Dim i As Integer
    oFrame = ThisComponent.CurrentController.Frame
```

```

a() = Array( ".uno:MenuBarVisible", ".uno:ObjectBarVisible", _
            ".uno:OptionBarVisible", ".uno:NavigationBarVisible", _
            ".uno:StatusBarVisible", ".uno:ToolBarVisible", _
            ".uno:MacroBarVisible", ".uno:FunctionBarVisible" )

oDisp = createUnoService("com.sun.star.frame.DispatchHelper")
For i = LBound(a()) to Ubound(a())
    oDisp.executeDispatch(oFrame, a(i), "", 0, Array())
Next

'on CalcFrames
s = ".uno:InputLineVisible"
oDisp.executeDispatch(oFrame, s, "", 0, Array())
End Sub

```

4.4. Intercept menu commands using Basic

Paolo Mantovani discovered that you can intercept menu commands from Basic. It was assumed that you could not because the language does not support the creation of custom UNO objects. I, Andrew Pitonyak, would like to point out that this is not the first time that Paolo Mantovani has managed to accomplish something that was claimed to be impossible.

Listing 4.7: Intercept menu commands.

```

REM Author: Paolo Mantovani
REM Modified: Andrew Pitonyak
Option Explicit

Global oDispatchInterceptor
Global oSlaveDispatchProvider
Global oMasterDispatchProvider
Global oFrame
Global bDebug As Boolean

'
Sub RegisterInterceptor
    Dim oFrame : oFrame = ThisComponent.currentController.Frame
    Dim s$      : s = "com.sun.star.frame.XDispatchProviderInterceptor"
    oDispatchInterceptor = CreateUnoListener("ThisFrame_", s)
    oFrame.registerDispatchProviderInterceptor(oDispatchInterceptor)
End Sub

'
Sub ReleaseInterceptor()
On Error Resume Next
    oFrame.releaseDispatchProviderInterceptor(oDispatchInterceptor)
End Sub

```

```

'
Function ThisFrame_queryDispatch ( oUrl As Object, _
    sTargetFrameName As String, lFlags As Long ) As Variant
    Dim oDisp
    Dim s$

    'the slot protocol causes ooo crash...
    If oUrl.protocol = "slot:" Then
        Exit Function
    End If

    If bDebug Then
        Print oUrl.complete, sTargetFrameName, lFlags
    End If

    s = sTargetFrameName
    oDisp = oSlaveDispatchProvider.queryDispatch( oUrl, s, lFlags )

    'do your management here
    Select Case oUrl.complete
        Case ".undo:Save" 'disable the save command
            Exit Function
        'Case "..."
        '...
        Case Else
            ' do nothing
    End Select

    ThisFrame_queryDispatch = oDisp
End Function

'
Function ThisFrame_queryDispatches ( mDispArray ) As Variant
    ThisFrame_queryDispatches = mDispArray
End Function

'
Function ThisFrame_getSlaveDispatchProvider ( ) As Variant
    ThisFrame_getSlaveDispatchProvider = oSlaveDispatchProvider
End Function

'
Sub ThisFrame_setSlaveDispatchProvider ( oSDP )
    oSlaveDispatchProvider = oSDP
End Sub

'

```

```

Function ThisFrame_getMasterDispatchProvider ( ) As Variant
    ThisFrame_getMasterDispatchProvider = oMasterDispatchProvider
End Function

'


---


Sub ThisFrame_setMasterDispatchProvider ( oMDP )
    oMasterDispatchProvider = oMDP
End Sub

'


---


Sub ToggleDebug()
    'be carefull! you will have a debug message
    ' for each dispatch....
    bDebug = Not bDebug
End Sub

```

5. Miscellaneous Examples

5.1. Display Text In Status Bar

Listing 5.1: Display text in the status bar, but it will not change.

```
'Author: Sasa Kelecevic
'email: scat@teol.net
'Here are two methods that may be used to obtain the
'status indicator
Function ProgressBar
    ProgressBar = ThisComponent.CurrentController.StatusIndicator
End Function

REM display text in status bar
Sub StatusText(sInformation as String)
    Dim iLen As Integer
    Dim iRest as Integer

    iLen = Len(sInformation)
    iRest = 270-iLen
    ProgressBar.start(sInformation+SPACE(iRest),0)
End Sub
```

According to Christian Erpelding [erpelding@ce-data.de], using the above code, you can only change the status bar ONCE and then all changes to the status bar are ignored. Use setText rather than start as shown below.

Listing 5.2: Display text in the status bar.

```
Sub StatusText(sInformation)
    Dim iLen as Integer
    Dim iRest As Integer
    iLen=Len(sInformation)
    iRest=350-iLen
    REM This uses the ProgressBar function shown above!
    ProgressBar.setText(sInformation+SPACE(iRest))
End Sub
```

5.2. Display All Styles In The Current Document

This is not as exciting as it appears. The following styles exist for a text document: CharacterStyles, FrameStyles, NumberingStyles, PageStyles, and ParagraphStyles.

Listing 5.3: Display all of the styles used in the current document.

```
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub DisplayAllStyles
    Dim mFamilyNames As Variant, mStyleNames As Variant
    Dim sMsg As String, n%, i%
    Dim oFamilies As Object, oStyle As Object, oStyles As Object
```

```

oFamilies = ThisComponent.StyleFamilies
mFamilyNames = oFamilies.getElementNames()
For n = LBound(mFamilyNames) To UBound(mFamilyNames)
    sMsg = ""
    oStyles = oFamilies.getByName(mFamilyNames(n))
    mStyleNames = oStyles.getElementNames()
    For i = LBound(mStyleNames) To UBound(mStyleNames)
        sMsg=sMsg + i + " : " + mStyleNames(i) + Chr(13)
        If ((i + 1) Mod 21 = 0) Then
            MsgBox sMsg,0,mFamilyNames(n)
            sMsg = ""
        End If
    Next i
    MsgBox sMsg,0,mFamilyNames(n)
Next n
End Sub

```

5.3. Iterate Through All Open Documents

Listing 5.4: Iterate through all of the open documents.

```

Sub Documents_Iteration( )
    Dim oDesktop As Object, oDocs As Object
    Dim oDoc As Object, oComponents As Object
    Dim i as Integer 'i counts how many windows are open in OOo
    i = 0

    oComponents = StarDesktop.getComponents()
    oDocs = oComponents.createEnumeration()
    Do While oDocs.hasMoreElements()
        oDoc = oDocs.nextElement()
        i = i + 1 'Counter
    Loop
    MsgBox i + "Components are currently open"
End Sub

```

5.4. List Fonts And Other Screen Information

Thanks to Paul Sobolik, I finally have a working example. First you create an abstract window toolkit and then you create a virtual device compatible with the screen. From this device, you can obtain things such as the screen dimensions and font information.

Tip

When designing a font, it is common to generate a version for different display attributes such as bold or italic. When you list the fonts supported by your system, you will frequently find all of the variations. Windows contains “Courier New Regular”, “Courier New Italic”, “Courier New Bold”, and “Courier New Bold Italic”.

I have a font document that is far more in depth than what is displayed here. The document is available on my web site.

Listing 5.5: List available fonts.

```
'Author:  Paul Sobolik
'email:  psobolik@lycos.com
Sub ListFonts
    Dim oToolkit as Object
    oToolkit = CreateUnoService("com.sun.star.awt.Toolkit")

    Dim oDevice as Variant
    oDevice = oToolkit.createScreenCompatibleDevice(0, 0)

    Dim oFontDescriptors As Variant
    oFontDescriptors = oDevice.FontDescriptors

    Dim oFontDescriptor As Object

    Dim sFontList as String
    Dim iIndex as Integer, iStart As Integer
    Dim iTotal As Integer, iAdjust As Integer

    iTotal = UBound(oFontDescriptors) - LBound(oFontDescriptors) + 1
    iStart = 1
    iAdjust = iStart - LBound(oFontDescriptors)
    For iIndex = LBound(oFontDescriptors) To UBound(oFontDescriptors)
        oFontDescriptor = oFontDescriptors(iIndex)
        sFontList = sFontList & iIndex + iAdjust & ": " & _
            oFontDescriptor.Name & " " & _
            oFontDescriptor.StyleName & Chr(10)
        If ((iIndex + iAdjust) Mod 20 = 0) Then
            MsgBox sFontList, 0, "Fonts " & iStart & " to " & _
                iIndex + iAdjust & " of " & iTotal
            iStart = iIndex + iAdjust + 1
            sFontList = ""
        End If
    Next iIndex
    If sFontList <> "" Then
        Dim s$
        s = "Fonts " & iStart & " to " & iIndex & " of " & iTotal
        MsgBox sFontList, 0, s
    End If
End Sub
```

Note that it depends on your Operating System (also on its settings) which Fonts are supported.

5.4.1. Display supported fonts

See my macro font document on my web site.

<http://www.pitonyak.org/AndrewFontMacro.odt>

The document iterates through the fonts and prints a summary of the different fonts with examples for each.

5.5. Set the default font using the ConfigurationProvider

To change the default font, run the macro and then restart OOo.

Listing 5.6: Set the default font using the ConfigurationProvider.

```
Author: Christian Junker
Sub DefaultFont_Change()
    Dim nodeArgs(0) As New com.sun.star.beans.PropertyValue
    Dim s$

    REM Properties
    nodeArgs(0).Name = "nodePath"
    nodeArgs(0).Value = "org.openoffice.Office.Writer/DefaultFont"
    nodeArgs(0).State = com.sun.star.beans.PropertyState.DEFAULT_VALUE
    nodeArgs(0).Handle = -1 'no handle!

    REM the required Config Services
    s = "com.sun.star.comp.configuration.ConfigurationProvider"
    Provider = createUnoService(s)
    s = "com.sun.star.configuration.ConfigurationUpdateAccess"
    UpdateAccess = Provider.CreateInstanceWithArguments(s, nodeArgs())

    REM set your DefaultFont now..
    UpdateAccess.Standard = "Arial"
    UpdateAccess.Heading = "Arial"
    UpdateAccess.List = "Arial"
    UpdateAccess.Caption = "Arial"
    UpdateAccess.Index = "Arial"
    UpdateAccess.commitChanges()
End Sub
```

5.6. Print Current Document

I played with this and I can print. I stopped trying to figure out how to print an A4 document on my Letter printer! I wanted to set this by default but I decided that it is not worth my time for now.

Listing 5.7: Print the current document.

```
'Author: Andrew Pitonyak
```

```

'email: andrew@pitonyak.org
Sub PrintSomePrinterProperties
    Dim mPrintopts1(), x as Variant

    'Dimensioned at 0, if you set any other properties,
    'be certain to set this to a higher value....
    Dim mPrintopts2(0) As New com.sun.star.beans.PropertyValue
    Dim oDoc As Object, oPrinter As Object
    Dim sMsg As String
    Dim n As Integer

    oDoc = ThisComponent

    '*****
    'Do you want to choose a certain printer
    'Dim mPrinter(0) As New com.sun.star.beans.PropertyValue
    'mPrinter(0).Name="Name"
    'mPrinter(0).value="Other printer"
    'oDoc.Printer = mPrinter()

    '*****
    'To simply print the document do the following:
    'oDoc.Print(mPrintopts1())

    '*****
    'To print pages 1-3, 7, and 9
    'mPrintopts2(0).Name="Pages"
    'mPrintopts2(0).Value="1-3; 7; 9"
    'oDoc.Printer.PaperFormat=com.sun.star.view.PaperFormat.LETTER
    'DisplayMethods(oDoc, "propr")
    'DisplayMethods(oDoc, "")
    oPrinter = oDoc.getPrinter()
    MsgBox "Printers " & LBound(oPrinter) & " to " & UBound(oPrinter)
    sMsg = ""
    For n = LBound(oPrinter) To UBound(oPrinter)
        sMsg = sMsg + oPrinter(n).Name + Chr(13)
    Next n
    MsgBox sMsg, 0, "Print Settings"

    'DisplayMethods(oPrinter, "propr")
    'DisplayMethods(oPrinter, "")

    'mPrintopts2(0).Name="PaperFormat"
    'mPrintopts2(0).Value=com.sun.star.view.PaperFormat.LETTER
    'oDoc.Print(mPrintopts2())
End Sub

```

5.6.1. Print Current Page

Listing 5.8: Print only the current page.

```
Dim aPrintOps(0) As New com.sun.star.beans.PropertyValue
oDoc = ThisComponent
oVCurs = oDoc.CurrentController.getViewCursor()
aPrintOps(0).Name = "Pages"
aPrintOps(0).Value = trim(str(oVCurs.getPage()))
oDoc.print(aPrintOps())
```

5.6.2. Other Printing Arguments

Another parameter to consider is the Wait parameter set to True. This causes printing to be synchronous and the call does not return until after printing is finished. This removes the requirement of a listener when printing is finished; assuming that you wanted to use one anyway. Note that your printer name may need to be surrounded by angle brackets “<>” and maybe not. If I remember correctly, this is related to network printers, but this may have changed.

5.6.3. Landscape

Listing 5.9: Print the document in landscape mode.

```
Sub PrintLandscape()
    Dim oOpt(1) as new com.sun.star.beans.PropertyValue

    oOpt(0).Name = "Name"
    oOpt(0).Value = "<insert_your_printername_here>"
    oOpt(1).Name = "PaperOrientation"
    oOpt(1).Value = com.sun.star.view.PaperOrientation.LANDSCAPE
    ThisComponent.Printer = oOpt()
End Sub
```

5.7. Configuration information

5.7.1. OOo version

Unfortunately, the function GetSolarVersion frequently stays the same even when the versions change. Version 1.0.3.1 returns “641”, 1.1RC3 returns 645, and 2.01 RC2 returns 680, but this is not enough granularity. The following macro returns the actual OOo version.

Listing 5.10: Obtain the current OpenOffice.org version.

```
Function OOoVersion() As String
    'Retrieves the running OOo version
    'Author : Laurent Godard
    'e-mail : listes.godard@laposte.net
    '
```

```

Dim oSet, oConfigProvider
Dim oParm(0) As New com.sun.star.beans.PropertyValue
Dim sProvider$, sAccess$
sProvider = "com.sun.star.configuration.ConfigurationProvider"
sAccess   = "com.sun.star.configuration.ConfigurationAccess"
oConfigProvider = createUnoService(sProvider)
oParm(0).Name = "nodepath"
oParm(0).Value = "/org.openoffice.Setup/Product"
oSet = oConfigProvider.CreateInstanceWithArguments(sAccess, oParm())
OOVersion=oSet.GetByName("ooSetupVersion")
End Function

```

5.7.2. OOo Locale

Listing 5.11: Obtain the current locale.

```

Function OOoLang() as string
    'Author : Laurent Godard
    'e-mail : listes.godard@laposte.net

    Dim oSet, oConfigProvider
    Dim oParm(0) As New com.sun.star.beans.PropertyValue
    Dim sProvider$, sAccess$
    sProvider = "com.sun.star.configuration.ConfigurationProvider"
    sAccess   = "com.sun.star.configuration.ConfigurationAccess"
    oConfigProvider = createUnoService(sProvider)
    oParm(0).Name = "nodepath"
    oParm(0).Value = "/org.openoffice.Setup/L10N"
    oSet = oConfigProvider.CreateInstanceWithArguments(sAccess, oParm())

    Dim OOoLangue as string
    OOoLangue= oSet.getbyname("ooLocale")      'en-US
    OOoLang=lcase(Left(trim(OOoLangue),2))      'en
End Function

```

The above macro is reputed to fail on AOO, but, the following should work correctly. The macro does function correctly with LO. The macro shown below should work for both AOO and LO.

Listing 5.12: Obtain the current locale using the tools library.

```

GlobalScope.BasicLibraries.loadLibrary( "Tools" )
Print
GetRegistryKeyContent("org.openoffice.Setup/L10N", FALSE).getByName("ooLocale")

```

5.8. Open And Close Documents (And The Desktop)

5.8.1. Close OpenOffice And/Or Documents

All OpenOffice.org documents and frame objects (services) support the XCloseable interface. To close these objects you must call `close(bForce As Boolean)`. Generally speaking, if `bForce` is false, the object may refuse to close, otherwise it can not refuse. I say generally speaking, because anything that has registered to listen for close events may veto a document close. When you tell OOo to print a document, control is returned before printing is finished. If you were then able to close the document, well, OOo would probably crash. Refer to the Developer's Guide for a more complete description of the XCloseable interface, and if you are uncertain which to use, use `close(True)`.

The desktop object does not support the XCloseable interface for legacy reasons. The `terminate()` method is used for this. This method causes a `queryTermination`-event to be broadcast to all listeners. If no `TerminationVetoException` is thrown, a `notifyTermination`-event is broadcast and true is returned. If not, an `abortTermination`-event is broadcast and false is returned. To quote Mathias Bauer, “the `terminate()` method was already there for a longer time, long before we discovered that it is not the right way to handle closing documents or windows. If this method hadn't been there, we would have used XCloseable for the desktop also.”[Bauer001]

The following macro uses `close` if `close` is supported and `dispose` otherwise.

Listing 5.13: Proper method to close an OpenOffice.org document.

```
Dim oDoc
oDoc = ThisComponent
If HasUnoInterfaces(oDoc, "com.sun.star.util.XCloseable") Then
    oDoc.close(true)
Else
    oDoc.dispose()
End If
```

Christian Junker adds the following:

There are many issues with properly closing a document or killing the soffice process. Using the method `StarDesktop.terminate()` does not kill the process!).

Use `oDoc.close(true)`, unless you have “exceptional” framework issues.

Avoid `oDoc.dispose()`! A disposed document may still be visible, but you can not manipulate it in any way. Even referencing the document using `ThisComponent` can lead to an error, because the Document's Model does not exist.

OOo version 2.0 should provide an improved Framework, especially for the close methods.

5.8.1.1. What if the file is modified?

I always assume that the document supports the close method. In other words, I assume that I am using a newer version of OpenOffice.org. I want to avoid any dialogs, so I check to see if the document has been modified. If I can store the document, then I try to do this.

Listing 5.14: Close a document that is modified

```
oDoc = ThisComponent
If (oDoc.isModified) Then
  If (oDoc.hasLocation AND (Not oDoc.isReadOnly)) Then
    oDoc.store()
  Else
    oDoc.setModified(False)
  End If
End If
oDoc.close(True)
```

5.8.2. Load A Document From A URL

To load a document from a URL, use the LoadComponentFromURL() method from the desktop. This loads a component into either a new or an existing frame.

Syntax:

```
loadComponentFromURL(
  string aURL,
  string aTargetFrameName,
  long nSearchFlags,
  sequence< com::sun::star::beans::PropertyValue > aArgs)
```

Returns:

com::sun::star::lang::XComponent

Parameters:

aURL: URL of the document to load. To create a new document, use "private:factory/scalc", "private:factory/swriter", "private:factory/swriter/web", etc.

aTargetFrameName: Name of the frame that will contain the document in. If a frame with the name exists, it is used, otherwise it is created. "_blank" creates a new frame, "_self" uses the current frame, "_parent" uses the parent of frame, and "_top" uses the top frame of the current path in the tree.

nSearchFlags: Use the values of FrameSearchFlag to specify how to find the specified aTargetFrameName . Normally, simply use 0 .

<http://api.openoffice.org/docs/common/ref/com/sun/star/frame/FrameSearchFlag.html>

Table 5.1: Frame search flags.

#	Name	Description
0	Auto	SELF+CHILDREN

#	Name	Description
1	PARENT	Includes the parent frame
2	SELF	Includes the start frame
4	CHILDREN	Include the child frames of the start frame
8	CREATE	Frame will be created if not found
16	SIBLINGS	Include the other child frames of the parent of the start frame
32	TASKS	Include all frames in all tasks in the current frames hierarchy
23	ALL	Include all frames not in other tasks. $23 = 1+2+4+16 = \text{PARENT} + \text{SELF} + \text{CHILDREN} + \text{SIBLINGS}$.
55	GLOBAL	Search entire hierarchy of frames. $55 = 1+2+4+16+32 = \text{PARENT} + \text{SELF} + \text{CHILDREN} + \text{SIBLINGS} + \text{TASKS}$.
63		GLOBAL + CREATE

aArgs: Specify component or filter specific behavior. "ReadOnly" with a boolean value specifies whether the document is opened read-only. "FilterName" specifies the component type to create and the filter to use, for example: "scal: Text - csv". See <http://api.openoffice.org/docs/common/ref/com/sun/star/document/MediaDescriptor.html>

Listing 5.15: Load a document from a given URL.

```
REM Frame "MyName" will be created if it does not exist
REM because it includes "CREATE" bit.
oDoc1 = StarDesktop.LoadComponentFromUrl(sUrl1, "MyName", 63, Array())
REM Use existing Frame "MyName"
oDoc2 = StarDesktop.LoadComponentFromUrl(sUrl2, "MyName", 55, Array())
```

Tip

In 1.1 the frame implements loadComponentFromURL so you can use:

```
oDoc = oDesk.LoadComponentFromUrl(sUrl_1, "_blank", 0,
Noargs())
oFrame = oDoc.CurrentController.Frame
oDoc = oFrame.LoadComponentFromUrl(sUrl_2, "", 2, Noargs())
```

Note the search flag arguments and the empty frame name argument.

Warning

In 1.1 you can only reuse a frame if you know its name.

Listing 5.16: Insert a document at the current cursor:

```
Sub insertDocumentAtCursor(sFileUrl$, oDoc)
    Dim oCur ' Created cursor
    Dim oVC : oDoc.getCurrentController().getViewCursor()
    Dim oText : oText = oVC.getText()
    oCur=oText.createTextCursorByRange(oVC.getStart())
    oCur.insertDocumentFromURL(sFileURL, Array())
End Sub
```


Be warned that the behavior when there is a problem opening a document, starting in OOo 2.x, an error will occur. Previously, a NULL document was returned.

Listing 5.17: Create a new document.

```
'----- create a new Writer file -----  
Dim s$ : s = "private:factory/swriter"  
oDoc=StarDesktop.loadComponentFromURL(s, "_blank", 0, Array())  
'----- open an existing file -----  
oDoc=StarDesktop.loadComponentFromURL(sUrl, "_blank", 0, Array())
```

5.8.2.1. A complete example

The purpose of this example, is to cycle through three documents titled one, two, and three. Each document is opened into the current frame.

Listing 5.18: Load a document into an existing frame.

```
Sub open_new_doc  
    Dim mArgs(2) as New com.sun.star.beans.PropertyValue  
    Dim oDoc  
    Dim oFrame  
    Dim s As String  
  
    If (ThisComponent.isModified) Then  
        If (ThisComponent.hasLocation AND (Not ThisComponent.isReadOnly)) Then  
            ThisComponent.store()  
        Else  
            ThisComponent.setModified(False)  
        End If  
    End If  
  
    mArgs(0).Name = "ReadOnly"  
    mArgs(0).Value = True  
  
    mArgs(1).Name = "MacroExecutionMode"  
    mArgs(1).Value = 4  
  
    mArgs(2).Name = "AsTemplate"  
    mArgs(2).Value = FALSE  
  
    REM Choose the next document to load  
    If ThisComponent.hasLocation Then  
        s = ThisComponent.getURL()  
        If InStr(s, "one") <> 0 Then  
            s = "file:///C:/tmp/two.oxt"  
        ElseIf InStr(s, "two") <> 0 Then  
            s = "file:///C:/tmp/three.oxt"
```

```

Else
    s = "file:///C:/tmp/one.oxt"
End If
Else
    s = "file:///C:/tmp/one.oxt"
End If

REM Get the document's frame and then load the specified document
REM into the current frame!
oFrame = ThisComponent.getCurrentController().getFrame()
oDoc = oFrame.LoadComponentFromUrl(s, "", 2, mArgs())

If IsNull(oDoc) OR IsEmpty(oDoc) Then
    Print "Unable to load " & s
End If
End Sub

```

5.8.3. Save a document with a password

To save a document with a password, you must set the “Password” attribute.

Listing 5.19: *Save a document using a password.*

```

Sub SaveDocumentWithPassword
    Dim args(0) As New com.sun.star.beans.PropertyValue
    Dim sURL$

    args(0).Name = "Password"
    args(0).Value = "test"

    sURL=ConvertToURL("/andrew0/home/andy/test.odt")
    ThisComponent.storeToURL(sURL, args())
End Sub

```

The argument name is case sensitive, so “password” will not work.

5.8.4. Create a new document from a template

To create a new document based on a template use the following:

Listing 5.20: *Create a new document from a template.*

```

Sub NewDoc
    Dim oDoc
    Dim sPath$
    Dim a(0) As New com.sun.star.beans.PropertyValue
    a(0).Name = "AsTemplate"
    a(0).Value = true

    sPath$ = "file:///~/Documents/DocTemplate.stw"

```

```
oDoc = StarDesktop.LoadComponentFromUrl(sPath$, "_blank" , 0, a())
End Sub
```

If you want to edit the template as a template, set “AsTemplate” to “False”.

Warning

After loading a document as hidden, you should not make the document visible because not all of the required services are initialized. This can cause OOO to crash. Hopefully this will be fixed in OOO version 2.0 .

5.8.5. How do I Enable Macros With LoadComponentFromURL

When a document is loaded by a macro, the contained macros are disabled. This is a security issue. As of version 1.1, you can enable macros when the document is loaded. Set the property “MacroExecutionMode” to either 2 or 4 and it should will work. I base this on an email on the dev mailing list. Thank You Mikhail Voitenko <Mikhail.Voitenko@Sun.COM>

<http://www.openoffice.org/servlets/ReadMsg?msgId=782516&listName=dev>

I condensed his reply:

The MediaDescriptor property MacroExecutionMode, uses values from the com.sun.star.document.MacroExecMode constants. If not specified, the default behavior forbids macro execution. Supported constant values are as follows: NEVER_EXECUTE,

FROM_LIST, ALWAYS_EXECUTE, USE_CONFIG, ALWAYS_EXECUTE_NO_WARN, USE_CONFIG_REJECT_CONFIRMATION, and USE_CONFIG_APPROVE_CONFIRMATION.

There are a few caveats to watch for. If you load a document "AsTemplate" the document is not opened, it is created. You must have events bound to “create document” rather than “open document”. To cover both cases, bind the macro to both events.

Listing 5.21: *Examples setting media descriptor properties.*

```
Dim oProp(1) As New com.sun.star.beans.PropertyValue
oProp(0).Name="AsTemplate"
oProp(0).Value=True
oProp(1).Name="MacroExecutionMode"
oProp(1).Value=4
```

This should work for macros configured to "OnNew" (Create Document), if you load a template or an sxw (but I have not tried it). If you use "OnLoad" (Open Document), you must set "AsTemplate" to *False* (or use an sxw file, because this defaults to *False*, where as templates (stw) default to *True*).

In OOO version 2.0, the Macro security is likely to be expanded.

5.8.6. Error handling on load

When a document fails to load a message is displayed providing information concerning the failed load. When the document is loaded from C++, it is possible that no exceptions will be thrown, so you will not be aware of the error.

Mathias Bauer explained that the XComponentLoader interface is not able to throw arbitrary exceptions so the “Interaction Handler” concept is used. When a document is loaded via `loadComponentFromURL`, an “InteractionHandler” is passed in the arguments array. The GUI provides a UI based Interaction Handler that converts the errors into a user interaction such as displaying an error message or prompting for a password (see the Developer's Guide for a few examples). If an Interaction Handler is not provided, a default handler is used. The default handler catches all exceptions and re-throws the few that may be thrown from `loadComponentFromURL`. Although it is not possible to implement your own interaction handler using Basic, the Developer's Guide has examples in other languages.

5.8.7. Mail Merge example, merge all documents in a directory

A mail merge creates a separate document for each merged record. This utility retrieves all write documents in a directory and creates a single output file that contains all of the documents combined into one. I modified the original macro so that all variables are declared and this works even if the first file found is not a Writer document.

***Listing 5.22:** Merge all documents in a single directory into one.*

```
'author: Laurent Godard
'Modified by: Andrew Pitonyak
Sub MergeDocumentsInDirectory()
' On Error Resume Next
Dim DestDirectory As String
Dim FileName As String
Dim SrcFile As String, DstFile As String
Dim oDesktop, oDoc, oCursor, oText
Dim argsInsert()
Dim args()
'Remove the following comments to do things hidden
'Dim args(0) As New com.sun.star.beans.PropertyValue
'args(0).name="Hidden"
'args(0).value=true

'Which destination directory?
DestDirectory=Trim(GetFolderName())

If DestDirectory = "" Then
MsgBox "No directory selected, exiting",16,"Merging Documents"
Exit Sub
End If
```

```

REM Force a trailing backslash.
REM This is okay because using URL notation
If Right(DestDirectory,1) <> "/" Then
    DestDirectory=DestDirectory & "/"
End If

oDeskTop=CreateUnoService("com.sun.star.frame.Desktop")

REM Read the first file!
FileName=Dir(DestDirectory)
DstFile = ConvertToURL(DestDirectory & "ResultatFusion.sxw")
Do While FileName <> ""
    If lcase(right(FileName,3))="sxw" Then
        SrcFile = ConvertToURL(DestDirectory & FileName)
        If IsNull(oDoc) OR IsEmpty(oDoc) Then
            FileCopy( SrcFile, DstFile )
            oDoc=oDeskTop.Loadcomponentfromurl(DstFile, _
                                                "_blank", 0, Args())

            oText = oDoc.getText
            oCursor = oText.createTextCursor()
        Else
            oCursor.gotoEnd(false)
            oCursor.BreakType = com.sun.star.style.BreakType.PAGE_BEFORE
            oCursor.insertDocumentFromUrl(SrcFile, argsInsert())
        End If
    End If
    FileName=dir()
Loop

If IsNull(oDoc) OR IsEmpty(oDoc) Then
    MsgBox "No documents merged!",16,"Merging Documents"
    Exit Sub
End If

'Save the document
Dim args2()
oDoc.StoreAsURL(DestDirectory & "ResultatFusion.sxw",args2())
If HasUnoInterfaces(oDoc, "com.sun.star.util.XCloseable") Then
    oDoc.close(true)
Else
    oDoc.dispose()
End If

'Reload the document!
oDoc=oDeskTop.Loadcomponentfromurl(DstFile,"_blank",0,Args2())
End Sub

```

5.9. Loading/Inserting an image into your document

This is a simple task that is difficult to figure out until you know that the inserted object must be created using `createInstance("object")` by the document. The image is inserted as a link into the document. The following macro inserts a text graphics object as a link to the existing document. This is a text graphics object, which is inserted at a cursor position. Although you must set the image size, you do not need to set the position.

Listing 5.23: *Insert a GraphicsObject into a document as a link.*

```
Sub InsertGraphicObject(oDoc, sURL$)
    REM Author: Andrew Pitonyak
    Dim oCursor
    Dim oGraph
    Dim oText

    oText = oDoc.getText()
    oCursor = oText.createTextCursor()
    oCursor.goToStart(FALSE)
    oGraph = oDoc.createInstance("com.sun.star.text.GraphicObject")

    With oGraph
        .GraphicURL = sURL
        .AnchorType = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
        .Width = 6000
        .Height = 8000
    End With

    'now insert the image into the text document
    oText.insertTextContent(oCursor, oGraph, False)
End Sub
```

You can also insert a graphics object shape, which is inserted into the draw page rather than at a cursor location. You must, therefore set the location and the size.

Listing 5.24: *Insert a GraphicsObjectShape into the draw page.*

```
Sub InsertGraphicObjectShape(oDoc, sURL$)
    REM Author: Andrew Pitonyak
    Dim oSize As New com.sun.star.awt.Size
    Dim oPos As New com.sun.star.awt.Point
    Dim oGraph

    REM First, create a graphic object shape
    oGraph = oDoc.createInstance("com.sun.star.drawing.GraphicObjectShape")

    REM Size and place the graphic object.
    oSize.Width=6000
    oSize.Height=8000
```

```

oGraph.setSize(oSize)

oPos.X = 2540
oPos.Y = 2540
oGraph.setPosition(oPos)

REM Assuming a text document, add it to the single draw page.
oDoc.getDrawpage().add(oGraph)

REM Set URL to the image.
oGraph.GraphicURL = sURL
End Sub

```

Tip

An inserted image can be contained in the document, or outside of the document. In either case, the GraphicURL always links to the image. If the object is not inserted as a link, then then URL starts with the text “vnd.sun.star.GraphicObject:”. Graphic objects inserted using the API are inserted as links – they are not embedded into the document. Danny Brewer, however, figured out how to get around this, as mentioned shortly.

5.9.1. Danny Brewer embeds an image

Danny Brewer figured out how to load a bitmap into the document and then obtain the URL of the loaded bitmap. The `com.sun.star.drawing.BitmapTable` service loads the image.

Listing 5.25: *Insert the image into the internal bitmap table.*

```

REM Given a URL to an external graphic resource,
REM load that graphic permanently into this drawing document,
REM and return a new URL to the internal resource.
REM The new URL can be used in place of the old URL.
Function LoadGraphicIntoDocument( oDoc As Object, cUrl$, cInternalName$ ) As String
    Dim oBitmaps
    Dim cNewUrl As String

    ' Get the BitmapTable from this drawing document.
    ' It is a service that maintains a list of bitmaps that are internal
    ' to the document.
    oBitmaps = oDoc.CreateInstance( "com.sun.star.drawing.BitmapTable" )

    ' Add an external graphic to the BitmapTable of this document.
    oBitmaps.insertByName( cInternalName, cUrl )

    ' Now ask for it back.
    ' What we get back is a different Url that points to a graphic
    ' which is inside this document, and remains with the document.
    cNewUrl = oBitmaps.getByName( cInternalName )

```

```

LoadGraphicIntoDocument = cNewUrl
End Function

```

Loading the bitmap into the document does not display the bitmap. The internal bitmap can be used with either Listing 5.23 or Listing 5.24 to insert an image that is stored inside of the document. The following macro inserts the same image using two different methods. Either will work, depending upon whether you desire an embedded or a linked image. Note that `oBitmaps.getByName("DBGif")` returns the URL of the embedded image in the form `vnd.sun.star.GraphicObject:<big hex number>`.

Listing 5.26: *You can insert as an external or an internal link.*

```

Dim sURL$
Dim oBitmaps
Dim s$

REM Insert a reference to the external file
sURL = "file:///andrew0/home/andy/db.gif"
InsertGraphicObject(ThisComponent, sURL)

REM Insert a reference to an internally contained graphic.
S = "com.sun.star.drawing.BitmapTable"
oBitmaps = ThisComponent.createInstance( s )
LoadGraphicIntoDocument(ThisComponent, sURL, "DBGif")
InsertGraphicObject(ThisComponent, oBitmaps.getByName("DBGif"))

```

5.9.2. Embed an image using a dispatch

A dispatch can easily embed an image into a document.

Listing 5.27: *Embed an image into a document.*

```

Dim oFrame
Dim oDisp
Dim oProp(1) as new com.sun.star.beans.PropertyValue

oFrame = ThisComponent.CurrentController.Frame
oDisp = createUnoService("com.sun.star.frame.DispatchHelper")

oProp(0).Name = "FileName"
oProp(0).Value = "file:///<YOURPATH>/<YOURFILE>"
oProp(1).Name = "AsLink"
oProp(1).Value = False
oDisp.executeDispatch(oFrame, ".uno:InsertGraphic", "", 0, oProp())

```


5.9.3. Embed an image directly

The following method, which requires OOo version 2.0 or later, was provided by Stephan Wunderlich. Unfortunately, all images were brought in using using a very small size, so I made some serious changes in an attempt to properly size the image after insertion.

5.9.3.1. Guess image size

Sometimes, you must guess the image size because it is not available. The following method assumes that the argument is a service of type `com.sun.star.graphic.GraphicDescriptor`, which optionally provides the image size in 100th mm and in pixels. The method returns a value in 100th mm, which is what is required for the internal display.

I purposely used floating point rather than long integer, because some of the intermediate values while scaling may be large.

Listing 5.28: Guess the image size.

```
Function RecommendGraphSize(oGraph)
    Dim oSize
    Dim lMaxW As Double ' Maximum width in 100th mm
    Dim lMaxH As Double ' Maximum height in 100th mm

    lMaxW = 6.75 * 2540 ' 6.75 inches
    lMaxH = 9.5 * 2540 ' 9.5 inches

    If IsNull(oGraph) OR IsEmpty(oGraph) Then
        Exit Function
    End If
    oSize = oGraph.Size100thMM
    If oSize.Height = 0 OR oSize.Width = 0 Then
        ' 2540 is 25.40 mm in an inch, but I need 100th mm.
        ' There are 1440 twips in an inch
        oSize.Height = oGraph.SizePixel.Height * 2540.0 * TwipsPerPixelY() / 1440
        oSize.Width = oGraph.SizePixel.Width * 2540.0 * TwipsPerPixelX() / 1440
    End If
    If oSize.Height = 0 OR oSize.Width = 0 Then
        'oSize.Height = 2540
        'oSize.Width = 2540
        Exit Function
    End If
    If oSize.Width > lMaxW Then
        oSize.Height = oSize.Height * lMaxW / oSize.Width
        oSize.Width = lMaxW
    End If
    If oSize.Height > lMaxH Then
        oSize.Width = oSize.Width * lMaxH / oSize.Height
        oSize.Height = lMaxH
    End If
```

```

RecommendGraphSize = oSize
End Function

```

If the size is available in 100th mm, then this is used. Next, the size is checked in Pixels. An image has both a size in pixels, and an expected pixel density (Dots Per Inch). We may have the number of pixels, but we do not have the DPI. I guess the pixel density as the pixel density of the computer display. In other words, if the expected size is not available, then assume that this was created for display on the current monitor.

5.9.3.2. *Embed image*

An image is embedded as follows:

1. A shape is created and added to the draw page.
2. The graphic provider service is used to obtain image descriptor from disk before it is loaded.
3. The image descriptor is used to guess the image size. The image size is guessed, because we only know what is in the descriptor, and the descriptor may not really know.
4. The shape is set to image as provided by the graphic provider service. At this point, the image is loaded and known by OOo.
5. A newly created graph object sets its URL to the URL used by the shape object. As such, the graphic and the shape should reference the same image.
6. The graph is anchored as a character and then inserted into the document at the cursor.
7. The shape is no longer required, so it is removed.

For reasons I do not understand, all images inserted as a very small images (less than 1 cm). I use the guessed image size to set the graphic size.

Listing 5.29: Embed an image in a document.

```

' oDoc - document to contain the image.
' oCurs - Cursor where the image is added
' sURL - URL of the image to insert.
' sParStyle - set the paragraph style to this.
Sub EmbedGraphic(oDoc, oCurs, sURL$, sParStyle$)
    Dim oShape
    Dim oGraph      'The graphic object is text content.
    Dim oProvider   'GraphicProvider service.
    Dim oText

    oShape = oDoc.CreateInstance("com.sun.star.drawing.GraphicObjectShape")
    oGraph = oDoc.CreateInstance("com.sun.star.text.GraphicObject")

```

```

oDoc.getDrawPage().add(oShape)

oProvider = createUnoService("com.sun.star.graphic.GraphicProvider")

Dim oProps(0) as new com.sun.star.beans.PropertyValue
oProps(0).Name = "URL"
oProps(0).Value = sURL

REM Save the original size.
Dim oSize100thMM
Dim lHeight As Long
Dim lWidth As Long
oSize100thMM = RecommendGraphSize(oProvider.queryGraphicDescriptor(oProps))
If NOT IsNull(oSize100thMM) AND NOT IsEmpty(oSize100thMM) Then
    lHeight = oSize100thMM.Height
    lWidth = oSize100thMM.Width
End If

oShape.Graphic = oProvider.queryGraphic(oProps())
oGraph.graphicurl = oShape.graphicurl
oGraph.AnchorType = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
oText = oCurs.getText()
oText.insertTextContent(oCurs, oGraph, false)
oDoc.getDrawPage().remove(oShape)

If lHeight > 0 AND lWidth > 0 Then
    Dim oSize
    oSize = oGraph.Size
    oSize.Height = lHeight
    oSize.Width = lWidth
    oGraph.Size = oSize
End If

' Set the paragraph style if it is in the document.
Dim oStyles
oStyles = oDoc.StyleFamilies.getByName("ParagraphStyles")
If oStyles.hasByName(sParStyle) Then
    oCurs.ParaStyleName = sParStyle
End If
End Sub

```

5.9.3.3. *Embed multiple images*

I started with a text file referencing many images. Everytime the list changed, I had to start over and work for a few hours to insert many images. The document format contains single lines of the form:

```
Insert image C:\path\image_name.png
```

The following macro uses a regular expression to match all paragraphs as shown above.

Listing 5.30: Find and embed an images.

```
Sub FindInsertGraphStatements(oDoc)
    Dim oSearch
    Dim oFound
    Dim s$
    Dim sLeading$
    Dim lLeadLen As Long
    Dim sFileURL$

    sLeading = "Insert image "
    lLeadLen = Len(sLeading)

    oSearch = oDoc.createSearchDescriptor()
    'Inspect oSearch
    With oSearch
        .SearchString = sLeading & "(.*)"
        .SearchRegularExpression = True
    End With
    oFound = oDoc.findFirst(oSearch)
    Do While Not IsNull(oFound)
        s = oFound.getString()
        sFileURL$ = ConvertToUrl(Trim(Right(s, Len(s) - lLeadLen)))
        If FileExists(sFileURL) Then
            oFound.setString("")
            EmbedGraphic(oDoc, oFound, sFileURL, "OOoFigure")
            'Exit Sub
        End If
        oFound = ThisComponent.findNext(oFound.End, oSearch)
    Loop
End Sub
```

5.9.4. Duplicate an existing image

With OOo 2.3, you can duplicate an image that is in a document; linked or unlinked. The macro creates a new graphic object from the first image object in the document. (??I added this before the functionality was supported. If you try this, let me know how it works??).

Listing 5.31: Duplicate an existing image.

```
Sub DuplicateFirstGrahpic()
    Dim oDoc, oExistingGraph, oNewGraph
    oDoc = ThisComponent
    oExistingGraph = oDoc.getGraphicObjects().getbyIndex(0).Graphic
    oNewGraph = oDoc.CreateInstance("com.sun.star.text.TextGraphicObject")
    oNewGraph.graphic = oExistingGraph
```

```

        ' Attach it to the start of the document.
        oNewGraph.attach(oDoc.getText().getStart())
    End Sub

```

5.9.5. Convert all linked images

I needed to convert a document full of linked images to embedded images. All of my images were of type TextGraphicObject.

To insert an embedded image into a document, it must first be inserted as a link and then changed to an embedded object. Initially, I figured out how to convert a GraphicsObjectShape and then more than a year later, I figured out how to convert a TextGraphicObject.

Listing 5.32: *Convert all linked images to embedded images.*

```

Sub ConvertAllLinkedGraphics(Optional aDoc)
    Dim oDoc          ' Working document
    Dim oDP           ' Draw page
    Dim i%            ' Index counter
    Dim oGraph        ' Graph object in the draw page
    Dim iLinked%      ' Number of linked images
    Dim iEmbedded%    ' Number of embedded images
    Dim iConverted%   ' Linked images converted to embedded
    Dim s1$           ' Graphic service name
    Dim s2$           ' Graphic service name

    REM Only know how to convert these types
    s1 = "com.sun.star.drawing.GraphicObjectShape"
    s2 = "com.sun.star.text.TextGraphicObject"

    If IsMissing(aDoc) OR IsNull(aDoc) OR IsEmpty(aDoc) Then
        oDoc = ThisComponent
    Else
        oDoc = aDoc
    End If

    REM Get the document draw page and then enumerate the images.
    oDP = oDoc.getDrawPage()
    For i=0 To oDP.getCount()-1
        oGraph = oDP.getByIndex(i)
        If oGraph.supportsService(s1) OR oGraph.supportsService(s2) Then
            If InStr(oGraph.GraphicURL, "vnd.sun") <> 0 Then
                iEmbedded = iEmbedded + 1
            Else
                iLinked = iLinked + 1
                If EmbedLinkedGraphic(oGraph, oDoc) Then
                    iConverted = iConverted + 1
                End If
            End If
        End If
    Next i
End Sub

```

```

        End If
    End If
Next
Print "Found " & iLinked & " linked and " & iEmbedded & _
    " embedded images and converted " & iConverted
End Sub

Function EmbedLinkedGraphic(oGraph, oDoc) As Boolean
    REM Author: Andrew Pitonyak
    Dim sGraphURL$ ' External URL of the image.
    Dim oGraph_2   ' Created image.
    Dim oCurs      ' Cursor where the image is located.
    Dim oText      ' Text object containing image.
    Dim oAnchor    ' Anchor point of the image
    Dim s1$        ' Graphic service name
    Dim s2$        ' Graphic service name

    EmbedLinkedGraphic = False
    If InStr(oGraph.GraphicURL, "vnd.sun") <> 0 Then
        REM Ignore an image that is already embedded
        Exit Function
    End If
    s1 = "com.sun.star.drawing.GraphicObjectShape"
    s2 = "com.sun.star.text.TextGraphicObject"
    If oGraph.supportsService(s1) Then

        REM Convert a GraphicObjectShape.
        oAnchor = oGraph.getAnchor()
        oText = oAnchor.getText()

        oGraph_2 = ThisComponent.createInstance(s)
        oGraph_2.GraphicObjectFillBitmap = oGraph.GraphicObjectFillBitmap
        oGraph_2.Size = oGraph.Size
        oGraph_2.Position = oGraph.Position
        oText.insertTextContent(oAnchor, oGraph_2, False)
        oText.removeTextContent(oGraph)
        EmbedLinkedGraphic = True

    ElseIf oGraph.supportsService(s2) Then

        REM Convert a TextGraphicObject.
        Dim oBitmaps
        Dim sNewURL$
        Dim sName$

        sName$ = oGraph.LinkDisplayName
        oBitmaps = oDoc.createInstance( "com.sun.star.drawing.BitmapTable" )

```

```

If oBitMaps.hasByName(sName) Then
    Print "Link display name " & sName & " already exists"
    Exit Function
End If
'Print "Ready to insert " & sName
oBitmaps.insertByName( sName, oGraph.GraphicURL )
sNewURL$ = oBitmaps.getByName( sName )
'Print "inserted URL " & sNewURL
oGraph.GraphicURL = sNewURL
EmbedLinkedGraphic = True
End If
End Function

```

5.9.6. Access OOO's internal storage system

- Use **File > Open**.
- Enter “vnd.sun.star.tdoc:”

You can now browse the internal storage systems for OOO. The neat thing is that you can load internally held items, such as images, for editing.

5.10. Setting Margins

The following macro assumes a text document. Draw and Impress documents use the `BorderLeft` method of the draw page.

***Listing 5.33:** Set the margins by modifying the text style.*

```

Sub Margins( )
    Dim oStyleFamilies, oFamilies, oPageStyles, oStyle
    Dim oVCurs, oPageStyleName
    Dim fromleft%, fromtop%, fromright%, frombottom%
    Dim oDoc
    oDoc = ThisComponent

    REM You don't need the view cursor, you can use any TextCursor
    oVCurs = oDoc.CurrentController.getViewCursor()
    oPageStyleName = oVCurs.PageStyleName
    oPageStyles = oDoc.StyleFamilies.getByName("PageStyles")
    oStyle = oPageStyles.getByName(oPageStyleName)
    REM fromleft, fromtop, fromright, frombottom = whatever you want
    oStyle.LeftMargin = fromleft
    oStyle.TopMargin = fromtop
    oStyle.RightMargin = fromright
    oStyle.BottomMargin = frombottom
End Sub

```

To remove manually applied margins, set the properties `ParaFirstLineIndent` and `ParaLeftMargin` to zero.

5.10.1. Setting the paper size

Setting the page size automatically sets the paper type.

Listing 5.34: *Set the page size using the Width and Height properties.*

```
Sub SetThePageStyle()  
    Dim oStyle  
    Dim sPageStyleName$  
    Dim oDoc  
    Dim s$  
    Dim oVC  
  
    oDoc = ThisComponent  
  
    REM You don't need the view cursor, you can use any TextCursor  
    oVC = oDoc.GetCurrentController().getViewCursor()  
    sPageStyleName = oVC.PageStyleName  
    DIM oPageStyles  
    oPageStyles = oDoc.StyleFamilies.GetByName("PageStyles")  
    oStyle = oPageStyles.GetByName(sPageStyleName)  
    REM Is this is Letter, then set to A4  
    If oStyle.Width = 27940 Then  
        Print "Setting to size A4"  
        oStyle.Width = 21000  
        oStyle.Height = 29700  
    Else  
        Print "Setting to size Letter"  
        oStyle.Width = 21590  
        oStyle.Height = 27940  
    End If  
  
    REM Note that the Width and Height properties are the same as the  
    REM values stored in the Size property. Seems silly, I know...  
    REM Setting the width or height sets both...  
    s="Width = " & CStr(oStyle.Width / 2540) & " inches" & CHR$(10)  
    s=s&"Height = "&CStr(oStyle.Height / 2540)&" inches" & CHR$(10)  
    s=s&"Width = "&CStr(oStyle.Size.Width / 2540) & " inches"&CHR$(10)  
    s=s&"Height = "&CStr(oStyle.Size.Height / 2540)&" inches" & CHR$(10)  
    MsgBox s, 0, "Page Style " & sPageStyleName  
End Sub
```

5.11. Calling an external program (Internet Explorer) using OLE

Use the Shell command or the OleObjectFactory (for Windows only).

Listing 5.35: Use the *OleObjectFactory* to start an application.

```
Sub using_IE( )
    Dim oleService
    Dim IE
    Dim s$

    s = "com.sun.star.bridge.OleObjectFactory"
    oleService = createUnoService(s)
    IE = oleService.CreateInstance("InternetExplorer.Application.1")
    IE.Visible = 1
    IE.Navigate("http://www.openoffice.org")
End Sub
```

5.12. Use the Shell command for files containing spaces

See the section on URL Notation! To summarize, use a %20 where the space should be.

Listing 5.36: You must use URL notation for spaces with the shell command.

```
Sub ExampleShell
    Shell("file:///C:/Andy/My%20Documents/oo/tmp/h.bat",2)
    Shell("C:\Andy\My%20Documents\oo\tmp\h.bat",2)
End Sub
```

The `ConvertToUrl` and `ConvertFromUrl` conveniently convert between URL notation and the notation used by your operating system – use these methods, they will save you time.

The third argument to the `Shell` function is the argument that is passed to the called program. The fourth argument, called `bSync`, determines if the shell command will wait until the shell process completes (`bSync = True`), or if the shell command returns immediately (`bSync = False`). The default value is **False**. If two consecutive `Shell` statements do not set the `bSync` argument to `True`, the second `Shell` statement is likely to be run before the first command has finished.

5.13. Read And Write Number In File

This shows how to read and write a string from a text file. The string is converted to a number and incremented. The number is then written back out to the file as a string.

Listing 5.37: Read and write a number in a file.

```
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub Read_Write_Number_In_File
    Dim CountFileName As String, NumberString As String
    Dim LongNumber As Long, iNum As Integer

    CountFileName = "C:\Andy\My Documents\oo\NUMBER.TXT"
    NumberString = "00000000"
```

```

LongNumber = 0

If FileExists(CountFileName) Then
    ON ERROR GOTO NoFile
    iNum = FreeFile

    OPEN CountFileName for input as #iNum
    LINE INPUT #iNum ,NumberString
    CLOSE #iNum
    MsgBox("Read " & NumberString, 64, "Read")

NoFile: 'in case an error occurred go here..
If Err <> 0 Then
    MsgBox("Can not read " & CountFileName, 64, "Error")
    NumberString = "00000001"
End If
On Local Error Goto 0
Else
    MsgBox(CountFileName & " does NOT exists", 64, "Warning")
    NumberString = "00000001"
End If

ON ERROR GOTO BadNumber
LongNumber = Int(NumberString) 'a single digit number is returned
LongNumber = LongNumber + 1
BadNumber:
If Err <> 0 Then
    MsgBox(NumberString & " is not a number", 64, "Error")
    LongNumber = 1
End If
On Local Error Goto 0
NumberString=Trim(Str(LongNumber))
While LEN(NumberString) < 8
    NumberString="0"&NumberString
Wend
MsgBox("Number is (" & NumberString & ")", 64, "Information")
iNum = FreeFile
OPEN CountFileName for output as #iNum
PRINT #iNum,NumberString
CLOSE #iNum
End Sub

```

5.14. Create Number Format Style

If you want a particular number format, then you can see if you have it and create it if you do not. For more information on valid formats, see the help contents on topic “number formats; formats”. They can be very complex.

Listing 5.38: Create a number format style.

```
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Function FindCreateNumberFormatStyle (_
    sFormat As String, Optional doc, Optional locale)
    Dim oDoc As Object
    Dim aLocale As New com.sun.star.lang.Locale
    Dim oFormats As Object
    Dim formatNum As Integer
    oDoc = IIf(IsMissing(doc), ThisComponent, doc)
    oFormats = oDoc.getNumberFormats()
    'If you choose to query on types, you need to use the type
    'com.sun.star.util.NumberFormat.DATE
    'I could set the locale from values stored at
    'http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt
    'http://www.chemie.fu-berlin.de/diverse/doc/ISO\_3166.html
    'I use a NULL locale and let it use what ever it likes.
    'First, see if the number format exists
    If (Not IsMissing(locale)) Then
        aLocale = locale
    End If
    formatNum = oFormats.queryKey (sFormat, aLocale, TRUE)
    MsgBox "Current Format number is" & formatNum
    'If the number format does not exist then add it
    If (formatNum = -1) Then
        formatNum = oFormats.addNew(sFormat, aLocale)
        If (formatNum = -1) Then formatNum = 0
        MsgBox "new Format number is " & formatNum
    End If
    FindCreateNumberFormatStyle = formatNum
End Function
```

5.14.1. View Supported Number Format Styles

The following macro enumerates the current number format styles. The style numbers (keys) and their text representation are inserted into the current document. The disadvantage to this version is that it enumerates the styles based on the locale. The original version enumerated the key from 0 to 1000, ignoring errors. This will find all formats regardless of locale, but I consider this macro a slightly cleaner solution.

Listing 5.39: View the supported number format styles.

```
Sub enumFormats()
    'Author : Laurent Godard
    'e-mail : listes.godard@laposte.net
    'Modified : Andrew Pitonyak
    Dim oText
    Dim vFormats, vFormat
```

```

Dim vTextCursor, vViewCursor
Dim iMax As Integer, i As Integer
Dim s$
Dim PrevChaine$, Chaine$
Dim aLocale As New com.sun.star.lang.Locale

vFormats = ThisComponent.getNumberFormats()
'RunSimpleObjectBrowser(vFormats)
oText = ThisComponent.Text
vViewCursor = ThisComponent.CurrentController.getViewCursor()
vTextCursor = oText.createTextCursorByRange(vViewCursor.getStart())
Dim v
v = vFormats.queryKeys(com.sun.star.util.NumberFormat.ALL, _
                        aLocale, False)
For i = LBound(v) To UBound(v)
    vFormat=vFormats.getbykey(v(i))
    chaine=vFormat.FormatString
    If Chaine<>Prevchaine Then
        PrevChaine=Chaine
        chaine=CStr(v(i)) & CHR$(9) & CHR$(9) & chaine & CHR$(10)
        oText.insertString(vTextCursor, Chaine, FALSE)
    End If
Next
MsgBox "Finished"
End Sub

```

5.15. Return the Fibonacci array

Listing 5.40: Return an array of Fibonacci numbers.

```

' *****
' http://disemia.com/software/openoffice/macro\_arrays.html
' Return the sequence of Fibonacci numbers
' assume that count >=2 is to make this code simpler
Function Fibonacci( nCount As Integer )
    If nCount < 2 Then nCount = 2

    Dim result( 1 to nCount) As Double
    Dim i As Integer

    result( 1) = 0
    result( 2) = 1

    For i = 3 to nCount
        result(i) = result( i - 2) + result( i - 1)
    Next i

    Fibonacci = result()

```

End Function

No matter how I choose to spell Fibonacci, I am told that I have made an incorrect choice. In response, I opted to research the subject, and this is what I found. Fibonacci, as he is usually called, is really Leonardo of Pisa, a great mathematician from the middle ages. Leonardo referred to himself as Fibonacci – short for the Latin phrase “filius Bonacci”, which means “the son of Bonaccio”. Fibonacci used the variations “Bonacci”, “Bonaccii” and “Bonacij”; the last usage is from the Latin. The different uses by Leonardo account for the different spellings in common usage today. In English, most modern authors use Fibonacci, but all bets are off if you switch to another language or use an older text.

5.16. Insert Text At Bookmark

Listing 5.41: Insert text at a bookmark.

```
oBookMark = oDoc.getBookmarks().getByName("<yourBookmarkName>")
oBookMark.getAnchor.setString("What you want to insert")
```

5.17. Saving And Exporting A Document

Saving a document is simple. The following macro will save a document if it has been modified, is not read-only, and has a location already set to save the document.

Listing 5.42: Save a document if it has not changed and it can be stored.

```
If (oDoc.isModified) Then
  If (oDoc.hasLocation AND (Not oDoc.isReadOnly)) Then
    oDoc.store()
  End If
End If
```

If the document is to be saved to a different location, then you must set some properties to direct where and how the document is to be stored.

Listing 5.43: Save a document to a different location.

```
Dim oProp(0) As New com.sun.star.beans.PropertyValue
Dim sUrl As String
sUrl = "file:///<complete/path/to/new/document>"

REM Set this to True if you want to overwrite the document.
oProp(0).Name = "Overwrite"
oProp(0).Value = False
oDoc.storeAsURL(sUrl, oProp())
```

To export a document to a different type, an export filter must be defined and any required properties must be set. You must know the name of the export filter and the file extension. Use Listing 5.46 to generate a list of filter names.

A separate method is required for the graphics filters and the rest. To export using a non-graphics format, use a form similar to the following code snippet.

Listing 5.44: Export a document.

```
Dim args2(1) As New com.sun.star.beans.PropertyValue
args2(0).Name = "InteractionHandler"
args2(0).Value = ""
args2(1).Name = "FilterName"
args2(1).Value = "MS Excel 97" REM Change the export filter
REM Use the correct file extension
oDoc.storeToURL("file:///c:/new_file.xls", args2())
```

Notice that I used the correct file extension and I specified the specific import filter. Graphics documents are a little different. First, you instantiate a `GraphicExportFilter` and then you tell it to export one page at a time.

Listing 5.45: Export a document using a `GraphicExportFilter`.

```
oFilter=CreateUnoService("com.sun.star.drawing.GraphicExportFilter")
Dim args3(1) As New com.sun.star.beans.PropertyValue
For i=0 to oDoc.drawpages.getcount()-1
    oPage=oDoc.drawPages(i)
    oFilter.setSourceDocument(opage)
    args3(0).Name = "URL"
    nom=opage.name
    args3(0).Value = "file:///c:/"&oPage.name&".JPG"
    args3(1).Name = "MediaType"
    args3(1).Value = "image/jpeg"
    oFilter.filter(args3())
Next
```

Tip

The online link to the import/export filters frequently changes, but you can probably find it at the <http://framework.openoffice.org> site. This is also available in the XML file located in the file:

<oooinstallationdir>\share\registry\data\org\openoffice\Office\TypeDetection.xcu

5.17.1. Filter names

You can list the filter names supported by OOo using a simple macro.

Listing 5.46: Enumerate all supported filter names.

```
Sub WriteFilterNamesToDoc
    Dim oFF ' FilterFactory object.
    Dim oFNames ' Filter names.
    Dim oDoc ' Newly created Write document.
    Dim oText ' Primary text object of new Write document.
    Dim oCursor ' Text cursor into Text object.
    Dim sURL$ ' URL to load a new empty Write document
    Dim i As Integer
    Dim iBRKConst As Long
```

```

oFF = createUnoService( "com.sun.star.document.FilterFactory" )
oFNames = oFF.getElementNames()

' Create a Writer doc and save the filter names to it.
SURL = "private:factory/swriter"
oDoc = StarDesktop.loadComponentFromURL( sURL, "_blank", 0, Array() )
oText = oDoc.getText()
oCursor = oText.createTextCursor()
oCursor.gotoEnd( False )
iBRKConst = com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK

' Print the filter names into a Writer document.
For i = LBound( oFNames ) To UBound( oFNames )
    oText.insertString( oCursor, oFNames(i), False )
    oText.insertControlCharacter( oCursor, iBRKConst, False )
Next
End Sub

```

5.18. User Fields

Most people use Master Fields. Master fields allow you to set the field's name and value.

5.18.1. Document Information

In OOO version 3.0, the document information object has been deprecated; use the document properties object instead. Rather than provide a long discussion, consider the following macro:

Listing 5.47: Display document information.

```

Sub GetDocumentProperties
    Dim oProps
    Dim oDocProps
    Dim s$
    Dim i As Integer
    Dim prop
    oDocProps = ThisComponent.getDocumentProperties()
    s = s & "Author = " & oDocProps.Author & CHR$(10) & _
        "AutoLoadSecs = " & oDocProps.AutoLoadSecs & CHR$(10) & _
        "AutoLoadURL = " & oDocProps.AutoLoadURL & CHR$(10) & _
        "CreationDate = " & DateStructToString(oDocProps.CreationDate) & _
        CHR$(10) & _
        "Default Target = " & oDocProps.DefaultTarget & CHR$(10) & _
        "Description = " & oDocProps.Description & CHR$(10) & _
        "EditingCycles = " & oDocProps.EditingCycles & CHR$(10) & _
        "EditingDuration = " & oDocProps.EditingDuration & CHR$(10) & _
        "Generator = " & oDocProps.Generator & CHR$(10) & _
        "Language = (" & oDocProps.Language.Country & ", " & _
        oDocProps.Language.Language & ", " & _

```

```

oDocProps.Language.Variant & ")" & CHR$(10) & _
"ModificationDate = " & _
DateStructToString(oDocProps.ModificationDate) & CHR$(10) & _
"ModifiedBy = " & oDocProps.ModifiedBy & CHR$(10) & _
"PrintDate = " & DateStructToString(oDocProps.PrintDate) & _
CHR$(10) & _
"PrintedBy = " & oDocProps.PrintedBy & CHR$(10) & _
"Subject = " & oDocProps.Subject & CHR$(10) & _
"TemplateDate = " & _
DateStructToString(oDocProps.TemplateDate) & CHR$(10) & _
"TemplateName" & oDocProps.TemplateName & CHR$(10) & _
"TemplateURL" & oDocProps.TemplateURL & CHR$(10) & _
"Title = " & oDocProps.Title & CHR$(10) & _
""

oProps = oDocProps.UserDefinedProperties
s = s & "Info 1 = " & oProps.[Info 1] & CHR$(10) & _
    "Info 2 = " & oProps.[Info 2] & CHR$(10) & _
    "Info 3 = " & oProps.[Info 3] & CHR$(10) & _
    "Info 4 = " & oProps.[Info 4]

MsgBox s
If LBound(oDocProps.Keywords) <= UBound(oDocProps.Keywords) Then
    MsgBox Join(oDocProps.Keywords, CHR$(10)), "Keywords"
End If
s = ""
Dim oStats : oStats = oDocProps.DocumentStatistics
For i = LBound(oStats) To UBound(oStats)
    s = s & oStats(i).Name & " = " & oStats(i).Value & CHR$(10)
Next
MsgBox s
End Sub

Function DateStructToString(ostruct) As String
    DateStructToString() = ostruct.Month & "/" & _
        ostruct.Day & "/" & ostruct.Year & " at " & _
        ostruct.Hours & ":" & ostruct.Minutes & ":" & _
        ostruct.Seconds & "." & ostruct.HundredthSeconds
End Function

```

Use loadFromMedium/storeToMedium to load and save document properties while a document is NOT open. This looks at the file and has no affect on any open documents. Any written changes will be over-written when the open document is saved.

5.18.2. Text Fields

The following macro, provide by Heike Talhammer, demonstrates how to enumerate the text fields contained in a document. The macro sets the field values and then uses refresh to cause the fields to update. In case you missed it, I will say it again: This macro changes the value of all of the fields contained in your document.

Listing 5.48: Enumerate text fields.

```
REM Author: Heike Talhammer <info@bios-pro.de>
REM Modified: Andrew Pitonyak
Sub EnumerateFields
    Dim vEnum
    Dim vVal
    Dim s1$, s2$
    Dim sFieldName$, sFieldValue$, sInstanceName$, sHint$, sContent$

    vEnum = thisComponent.getTextFields().createEnumeration()
    If Not IsNull(vEnum) Then
        Do While vEnum.hasMoreElements()
            vVal = vEnum.nextElement()
            If vVal.supportsService("com.sun.star.text.TextField.Input") Then
                sHint=vVal.getPropertyValue("Hint")
                sContent=vVal.getPropertyValue("Content")
                s1=s1 & "Hint:" & sHint & " - Content: " & sContent & chr(13)
                'change the content
                vVal.setPropertyValue("Content", "My new content")
                ThisComponent.TextFields.refresh()
            End If

            If vVal.supportsService("com.sun.star.text.TextField.User") Then
                sFieldName =vVal.textFieldMaster.Name
                sFieldValue = vVal.TextFieldMaster.Value
                sInstanceName= vVal.TextFieldMaster.InstanceName
                s2 = s2 & sFieldName & " = " & sFieldValue & chr(13) & _
                    "InstanceName: " & sInstanceName & chr(13)
                'new value for textfield
                vVal.TextFieldMaster.Value=25
            End If
        Loop
        MsgBox s1, 0, "=== Input Fields ==="
        MsgBox s2, 0, "=== User Fields ==="
    End If
    ThisComponent.TextFields.refresh()
End Sub
```

5.18.3. Master Fields

Master fields are nice, you can set your own values, formulas, or numeric values. This is only a brief investigation but it should be enough to get you started. I have found five types of master fields: Illustration, Table, Text, Drawing, and User. The names of these fields all begin with “com.sun.star.text.FieldMaster.SetExpression.” followed by the type and then finally followed by another period and the name. Here is a simple way to create or modify a text field.

Listing 5.49: Get a master field.

```
vDoc = ThisComponent
sName = "Author Name"
sServ = "com.sun.star.text.FieldMaster.User"
If vDoc.getTextFieldMasters().hasByName(sServ & sName) Then
    vField = vDoc.getTextFieldMasters().getByName(sServ & sName)
    vField.Content = "Andrew Pitonyak"
    'vField.Value = 2.3 REM If you would rather this were a number!
Else
    vField = vDoc.CreateInstance(sServ)
    vField.Name = sName
    vField.Content = "Andrew Pitonyak"
    'vField.Value = 2.3 REM If you would rather this were a number!
End If
```

This macro will display all of the master fields in the document.

Listing 5.50: Display all master fields.

```
Sub DisplayMasterFields(Optional oUseDoc)
    Dim oMasters      'Text field masters from the document.
    Dim sNames()      'Text field master names.
    Dim i%            'Generic loop variable.
    Dim s$            'Generic string variable.
    Dim oMasterField  'A particular master field.
    Dim sUserType$    'Holds the name for a User defined text field master

    sUserType = "com.sun.star.text.FieldMaster.User"

    If IsMissing(oUseDoc) Then
        oMasters = ThisComponent.getTextFieldMasters()
    Else
        oMasters = oUseDoc.getTextFieldMasters()
    End If
    sNames() = oMasters.getElementNames()
    s = "===Text Field Masters==="
    For i = LBound(sNames()) to UBound(sNames())
        s = s & Chr$(13) & "(" & sNames(i)
        oMasterField = oMasters.getByName(sNames(i))
        If Not IsNull(oMasterField) Then
            s = s & "," & oMasterField.Name
            If (Left$(sNames(i), Len(sUserType)) = sUserType) Then
                REM If the user type is an expression, then you can access
                REM the Value property, which is a double.
                s = s & "," & oMasterField.Content
            End If
        End If
    End If
    s = s & ")"
```

```

        s = s & " count = " & _
        (UBound(oMasterField.DependentTextFields) - _
        LBound(oMasterField.DependentTextFields) + 1)
    Next i
    MsgBox s, 0, "Text Field Masters"
End Sub

```

The following routines, posted by Rodrigo V Nunes [rodrigo.nunes@net-linx.com], show that setting Document variables (like for MS Word documents) are possible in OOo.

Listing 5.51: Setting document variables.

```

'=====
' CountDocVars - routine used to count the number of document variables
'                 available for the current document
' In - DocVars: array of current document variables (name) present in ad
'       DocVarValue: array of current document variables (value) present in ad
' Out - integer with the total number of doc variables found
'=====
Function CountDocVars(DocVars , DocVarValue) As Integer
    Dim VarCount As Integer
    Dim Names as Variant

    VarCount = 0
    Names = thisComponent.getTextFieldMasters().getElementNames()
    For i%=LBound(Names) To UBound(Names)
        if (Left$(Names(i%),34) = "com.sun.star.text.FieldMaster.User") Then
            xMaster = ThisComponent.getTextFieldMasters.getByName(Names(i%))
            DocVars(VarCount) = xMaster.Name
            DocVarValue(VarCount) = xMaster.Value
            VarCount = VarCount + 1 ' document variable created by the user
        End if
    Next i%

    CountDocVars = VarCount
End Function

'
'=====
' SetDocumentVariable - routine used to create/set value of a document
' variable into the document user's textfield list, without physically
' inserting its contents in the text of the ad.
' In - strVarName: string with the name of the variable to be set/created
'       aValue:      string with the value of the doc variable
' Out - boolean flag with the operation status: TRUE=OK, FALSE=variable
'       could not be set/created
'=====
Function SetDocumentVariable(ByVal strVarName As String, ByVal aValue As String
) As Boolean
    Dim bFound As Boolean

```

```

On Error GoTo ErrorHandler
oActiveDocument = thisComponent
oTextmaster = oActiveDocument.getTextFieldMasters()
sName = "com.sun.star.text.FieldMaster.User." + strVarName
bFound = oActiveDocument.getTextFieldMasters.hasbyname(sName) ' check if
variable exists
if bFound Then
    xMaster = oActiveDocument.getTextFieldMasters.getByName( sName )
    REM value MEMBER used for decimal values, CONTENT member for Strings
    'xMaster.value = aValue
    xMaster.Content = aValue
Else ' Document variable doesn't exist yet
    sService = "com.sun.star.text.FieldMaster.User"
    xMaster = oActiveDocument.createInstance( sService )
    xMaster.Name = strVarName
    xMaster.Content = aValue
End If
SetDocumentVariable = True 'Success
Exit Function

ErrorHandler:
    SetDocumentVariable = False
End Function

'=====
' InsertDocumentVariable - routine used to insert a document variable into
' the document user's textfield list and into the ad text, at the current
' cursor position
' In - strVarName: string with the name of the variable to be inserted
' oTxtCursor: current cursor object with the position to place the var
' Out - none
'=====
Sub InsertDocumentVariable(strVarName As String, oTxtCursor As Object)

    oActiveDocument = thisComponent
    objField = thisComponent.createInstance("com.sun.star.text.TextField.User")
    sName = "com.sun.star.text.FieldMaster.User." + strVarName
    bFound = oActiveDocument.getTextFieldMasters.hasbyname(sName)
    if bFound Then
        objFieldMaster = oActiveDocument.getTextFieldMasters.getByName(sName)
        objField.attachTextFieldMaster(objFieldMaster)
        ' Insert the Text Field
        oText = thisComponent.Text
        'oCursor = oText.createTextCursor()
        'oCursor.gotoEnd(false)
        oText.insertTextContent(oTxtCursor, objField, false)
    End If

```

```

End Sub

'=====
' DeleteDocumentVariable - routine used to eliminate a document variable
'       from the document user's textfield list
' In   - strVarName: string with the name of the variable to be deleted
' Out  - none
'=====

Sub DeleteDocumentVariable(strVarName As String)

    oActiveDocument = thisComponent
    objField = oActiveDocument.CreateInstance("com.sun.star.text.TextField.User")
    sName = "com.sun.star.text.FieldMaster.User." + strVarName
    bFound = oActiveDocument.getTextFieldMasters.hasbyname(sName) ' exists?
    if bFound Then
        objFieldMaster = oActiveDocument.getTextFieldMasters.getByName(sName)
        objFieldMaster.Content = ""
        objFieldMaster.dispose()
    End If

End Sub

'
'=====
' SetUserVariable - function used to set/create user variables inside of the
'       document.  These variables are for internal system
'       use/control only, and will NOT be available or used in
'       the java application (see 'SetDocumentVariables' for
'       document shared variable creation/set)
'
' In - strVarName: string with the name of the variable to be set.  If the
'       variable does not exist, it'll be created
'       avalue:    Variant value with the new content of the variable
'               defined in strVarName
' Out - boolean flag with the operation status: TRUE=OK, FALSE=variable
'       could not be set/created
'=====

Function SetUserVariable(ByVal strVarName$, ByVal aValue) As Boolean

    Dim aVar As Variant
    Dim index As Integer           'Index of the existing variable name
    Dim vCount As Integer

    On Error GoTo ErrorHandler
    'Look to see if the document variable already exists.
    '?? March 8, 2009

```

```

'Document Info object is deprecated, convert to use document properties.
oDocumentInfo = thisComponent.Document.Info
vCount = oDocumentInfo.getUserFieldCount()
bFound = false
For i% = 0 to (vCount - 1)
    If strVarName = oDocumentInfo.getUserFieldName(i%) Then
        bFound = true
        oDocumentInfo.setUserFieldValue(i%, avalue)
    End If
Next i%
If not bFound Then 'Document variable doesn't exist yet
    oDocumentInfo.setUserFieldName(i, strVarName)
    oDocumentInfo.setUserValue(i, avalue)
End If
' test if value is bigger than the number of user variables !

SetUserVariable = True 'Success
Exit Function

ErrorHandler:
    SetUserVariable = False
End Function

```

Christian Junker wrote the following macro to call and test the routines above:

Listing 5.52: Use the code in Listing 5.51

```

REM Here is code to call the routines above:
Sub Using_docVariables()

    odoc = thisComponent
    otext = odoc.getText()
    ocurser = otext.createTextCursor()
    ocurser.goToStart(false)

    SetDocumentVariable("docVar1", "Value 1") 'create my DocVariable
    InsertDocumentVariable("docVar1", ocurser) 'insert it into current document
End Sub

```

5.18.4. Removing Text Fields

Retrieve the text field and then dispose it using field.dispose()!

5.18.5. Insert a URL into a Calc cell

For reasons that defy me, the following functionality has been requested numerous times. I opted to add this example because I am tired of figuring out how to do it every time. The InsertURLIntoCell macro converts the text of a cell into a URL and then inserts a URL text field into the cell. Read the comments to see how this is done.

Listing 5.53: Insert a URL into a Calc cell.

```
Sub InsertURLIntoCell
    Dim oText    'Text object for the current object
    Dim oField   'Field to insert
    Dim oCell    'Get a specific cell

    Rem Get a cell, any cell. This obtains cell C3
    oCell = ThisComponent.Sheets(0).GetCellByPosition(2,2)

    REM Create a URL Text field
    oField = ThisComponent.CreateInstance("com.sun.star.text.TextField.URL")

    REM This is the actual text that is displayed for the URL
    REM This could just as easily be
    REM oField.Representation = "My Secret Text"
    oField.Representation = oCell.getString()

    REM The URL property is just a text string of the URL itself.
    oField.URL = ConvertToURL(oCell.getString())

    REM The text field is added as text content into the cell.
    REM If you do not now set the string to zero, then the existing
    REM text will remain and the new URL text field will be appended
    REM to the end.
    oCell.setString("")
    oText = oCell.getText()
    oText.insertTextContent(oText.createTextCursor(), oField, False)
End Sub
```

Warning This does not work in a Write document, so do not try it!

5.18.6. Find a URL in a Calc cell.

I know that I have a URL (hyperlink) in Calc cell A1.

Listing 5.54: Get URL from a Calc cell.

```
Sub FindHyperLinkInCell
    Dim oCell, oText, oParEnum, oParElement
    Dim oEnum, oElement
    oCell = ThisComponent.Sheets(0).getCellByPosition(0, 0)
    oParEnum = oCell.getText().createEnumeration()
    Do While oParEnum.hasMoreElements ()
        oParElement = oParEnum.nextElement()
        oEnum = oParElement.createEnumeration()
        Do While oEnum.hasMoreElements ()
            oElement = oEnum.nextElement()
            If oElement.TextPortionType = "TextField" Then
```

```

        If oElement.TextField.supportsService("com.sun.star.text.TextField.URL")
Then
    'STRING Representation = http://www.pitonyak.org
    'STRING TargetFrame =
    'STRING URL =
    Print oElement.TextField.URL
End If
End If
Loop
Loop
End Sub

```

5.18.7. Adding a SetExpression TextField

I create an use my own SetExpression fields to number my text tables, code listings, figured, and anything else that must be sequentially numbered. The following code assumes that you know how to add these manually and that there already exists a number field named Table. Use Insert | Fields | Other to open the Fields dialog. Select the Variables tab. In the type box, choose “Number Range”. Normally, I would enter the expression “Table+1”, but I want to add one using a macro.

Listing 5.55: Append a SetExpression text field to the end of the document.

```

Sub AddExpressionField
    Dim oField          ' This is the SetExpression field that is inserted.
    Dim oMasterField    ' The master field for the SetExpression field.
    Dim sMasterFieldName$ ' This is the name of the master field.
    Dim oDoc             ' The document that will contain the field.
    Dim oText           ' The documents text object.

    oDoc = ThisComponent

    REM The text field must be created by the document that will contain it.
    oField = oDoc.CreateInstance("com.sun.star.text.TextField.SetExpression")

    REM Set the expression
    oField.Content = "Table+1"

    REM Normally, you might want to create or check the number format.
    REM I am cheating because I happen to know what it is and I want a shorter
    REM example. I have examples elsewhere that show you how to get the index
    REM of a number format.
    oField.NumberFormat = 4
    oField.NumberingType = 4

    REM Wow, now that is a long name. All master fields are named this way.
    REM Use the name to get the master field that is assumed to exist.
    sMasterFieldName = "com.sun.star.text.FieldMaster.SetExpression.Table"

```



```

oMasterField = oDoc.getTextFieldMasters().getByName(sMasterFieldName)

REM Attach the text field to its master.
oField.attachTextFieldMaster(oMasterField)

REM Finally, insert the field at the END of the document.
oText = oDoc.getText()
oText.insertTextContent(oText.getEnd(), oField, False)
End Sub

```

5.19. User Defined Data Types

As of OOo 1.1.1, you can define your own data types.

Listing 5.56: *You can define your own data types.*

```

Type PersonType
    FirstName As String
    LastName As String
End Type

Sub ExampleCreateNewType
    Dim Person As PersonType
    Person.FirstName = "Andrew"
    Person.LastName = "Pitonyak"
    PrintPerson(Person)
End Sub

Sub PrintPerson(x)
    Print "Person = " & x.FirstName & " " & x.LastName
End Sub

```

I gave a presentation at the 2004 OOo Conference in Berlin concerning creating advanced data types using structures. The examples are in the presentation available on my web site.

5.20. Spell Check, Hyphenation, and Thesaurus

Performing a spell check, hyphenation, and a thesaurus lookup is very easy. These parts will return null values if their corresponding parts are not configured. In my initial testing, the Hyphenation routine always returned null until I configured the Hyphenation from the Options dialog.

Listing 5.57: *Spell, hyphenate, and use a thesaurus.*

```

Sub SpellCheckExample
    Dim s() As Variant
    Dim vReturn As Variant, i As Integer
    Dim emptyArgs(0) As New com.sun.star.beans.PropertyValue
    Dim aLocale As New com.sun.star.lang.Locale
    aLocale.Language = "en"

```

```

aLocale.Country = "US"

s = Array("hello", "anesthesiologist", _
          "PNEUMONOLTRAMICROSCOPICSILICOVOLCANOCONIOSIS", _
          "Pitonyak", "misspell")

'*****Spell Check Example!
Dim vSpeller As Variant
vSpeller = createUnoService("com.sun.star.linguistic2.SpellChecker")
'Use vReturn = vSpeller.spell(s, aLocale, emptyArgs()) if you want options!
For i = LBound(s()) To UBound(s())
    vReturn = vSpeller.isValid(s(i), aLocale, emptyArgs())
    MsgBox "Spell check on " & s(i) & " returns " & vReturn
Next

'*****Hyphenation Example!
Dim vHyphen As Variant
vHyphen = createUnoService("com.sun.star.linguistic2.Hyphenator")
For i = LBound(s()) To UBound(s())
    'vReturn = vHyphen.hyphenate(s(i), aLocale, 0, emptyArgs())
    vReturn = vHyphen.createPossibleHyphens(s(i), aLocale, emptyArgs())
    If IsNull(vReturn) Then
        'hyphenation is probably off in the configuration
        MsgBox "Hyphenating " & s(i) & " returns null"
    Else
        MsgBox "Hyphenating " & s(i) & _
               " returns " & vReturn.getPossibleHyphens()
    End If
Next

'*****Thesaurus Example!
Dim vThesaurus As Variant, j As Integer, k As Integer
vThesaurus = createUnoService("com.sun.star.linguistic2.Thesaurus")
s = Array("hello", "stamp", "cool")
For i = LBound(s()) To UBound(s())
    vReturn = vThesaurus.queryMeanings(s(i), aLocale, emptyArgs())
    If UBound(vReturn) < 0 Then
        Print "Thesaurus found nothing for " & s(i)
    Else
        Dim sTemp As String
        sTemp = "Hyphenated " & s(i)
        For j = LBound(vReturn) To UBound(vReturn)
            sTemp = sTemp & Chr(13) & "Meaning = " & _
                   vReturn(j).getMeaning() & Chr(13)
        Next j
        Dim vSyms As Variant
        vSyms = vReturn(j).querySynonyms()
        For k = LBound(vSyms) To UBound(vSyms)

```

```

        sTemp = sTemp & vSyms(k) & " "
    Next
    sTemp = sTemp & Chr(13)
Next
MsgBox sTemp
End If
Next
End Sub

```

5.21. Changing The Mouse Cursor

The quick answer is: This is not supported.

A desire to change the mouse cursor sparked an interesting discussion that I took the time to follow but I did not test. I have edited the messages for brevity.

anindya@agere.com asked: I want the mouse pointer to be an hour glass while a macro is running. What is wrong with this code?

Listing 5.58: *You can not change the mouse cursor as of OOO version 1.1.3.*

```

oDoc = oDesktop.loadComponentFromURL(fileName, "_blank", 0, mArg())
oCurrentController = oDoc.getCurrentController()
oFrame = oCurrentController.getFrame()
oWindow = oFrame.getContainerWindow()
oPointer = createUnoService("com.sun.star.awt.Pointer")
oPointer.SetType(com.sun.star.awt.SystemPointer.WAIT)
oWindow.setPointer(oPointer)

```

Mathias Bauer, whom we all love, responded. You can not set the mouse pointer of a document window via UNO-API. VCL manages the mouse pointer based on the window, not the top window. Any VCL window can have its own mouse pointer set. If you want to change the mouse pointer of the document window, you must access its XWindowPeer (not the peer of the frame window), and this is not available in the API. Another problem might be that OOO changes the mouse pointer internally and overrides your setting.

Berend Cornelius provided the final response. Your Sub works fine with any sub-window in your document. The following code refers to a control in a document:

Listing 5.59: *Switch the mouse pointer for a control.*

```

Sub Main
    Dim oController
    Dim oControl
    Dim oDrawControl
    GlobalScope.BasicLibraries.LoadLibrary("Tools")
    oController = ThisComponent.getCurrentController()
    oDrawControl = ThisComponent.Drawpage().getbyIndex(0).getControl()
    oControl = oController.getControl(oDrawControl)
    SwitchMousePointer(oControl.getPeer(), False)

```

```
End Sub
```

This routine changes the mouse pointer when it is over the control, but when the pointer is not over the control window it changes back. You want a "Wait" function that places the pointer in a wait state but this is currently not supported by the API.

It is my opinion that you can change it but not for all things.

```
oDoc.getCurrentController().getFrame().getContainerWindow().setPointer(...)
```

5.22. Setting The Page Background

Listing 5.60: Set a page background.

```
Sub Main
    ' First get the Style Families
    oStyleFamilies= ThisComponent.getStyleFamilies()
    ' then get the PageStyles
    oPageStyles= oStyleFamilies.getByName("PageStyles")
    ' then get YOUR page's style
    oMyPageStyle= oPageStyles.getByName("Standard")
    ' then set your background
    with oMyPageStyle
        .BackGraphicUrl= _
            convertToUrl( <pathToYourGraphic> )
        .BackGraphicLocation= _
            com.sun.star.style.GraphicLocation.AREA
    end with
End Sub
```

5.23. Manipulating the clipboard

Accessing the clipboard directly is not easy. Most access is accomplished using dispatch statements. To copy data to the clipboard, you must first select data. The optional Controller interface `com.sun.star.view.XSelectionSupplier` provides the ability to select objects and to access the currently selected objects. Write introduced the methods `getTransferable` and `insertTransferable`, which allow selected areas to be copied without using the clipboard. This method will be available for Calc version 2.3.

5.23.1. Copy Spreadsheet Cells With The Clipboard

The first example sent to me selects cells in a spreadsheet and then pastes them into a different spreadsheet.

Listing 5.61: Copy and paste a range using the clipboard.

```
'Author: Ryan Nelson
'email: ryan@aurelius-mfg.com
'Modified By: Christian Junker and Andrew Pitonyak
'This macro copies a range and pastes it into a new or existing spreadsheet.
```

```

Sub CopyPasteRange()
    Dim oSourceDoc, oSourceSheet, oSourceRange
    Dim oTargetDoc, oTargetSheet, oTargetCell
    Dim oDisp, octl
    Dim sUrl As String
    Dim NoArg()

    REM Set source doc/currentController/frame/sheet/range.
    oSourceDoc=ThisComponent
    octl = oSourcedoc.getCurrentController()
    oSourceframe = octl.getFrame()
    oSourceSheet= oSourceDoc.Sheets(0)
    oSourceRange = oSourceSheet.getCellRangeByPosition(0,0,100,10000)

    REM create the DispatcherService
    oDisp = createUnoService("com.sun.star.frame.DispatchHelper")

    REM select source range
    octl.Select(oSourceRange)

    REM copy the current selection to the clipboard.
    oDisp.executeDispatch(octl, ".uno:Copy", "", 0, NoArg())

    REM open new spreadsheet.
    sURL = "private:factory/scalc"
    oTargetDoc = Stardesktop.loadComponentFromURL(sURL, "_blank", 0, NoArg())
    oTargetSheet = oTargetDoc.getSheets.getByIndex(0)

    REM You may want to clear the target range prior to pasting to it if it
    REM contains data and formatting.
    REM Move focus to cell 0,0.
    REM This ensures the focus is on the "0,0" cell prior to pasting.
    REM You could set this to any cell.
    REM If you don't set the position, it will paste to the
    REM position that was last in focus when the sheet was last open.
    oTargetCell = oTargetSheet.getCellByPosition(0,0)
    oTargetDoc.getCurrentController().Select(oTargetCell)

    REM paste from the clipboard to your current location.
    oTargetframe = oTargetDoc.getCurrentController().getFrame()
    oDisp.executeDispatch(oTargetFrame, ".uno:Paste", "", 0, NoArg())
End Sub

```

5.23.2. Copy Spreadsheet Cells Without The Clipboard

You can copy, insert, move, and remove cells within the same Calc document without using the clipboard – even between different sheets. See:

<http://api.openoffice.org/docs/common/ref/com/sun/star/sheet/XCellRangeMovement.html> for more details. The following code was posted on the devapi mailing list.

Listing 5.62: *Copy and paste a range without the clipboard.*

```
' Author:  Oliver Brinzing
' email:   OliverBrinzing@t-online.de
Sub CopySpreadsheetRange
    REM Get sheet 1, the original, and 2, which will contain the copy.
    oSheet1 = ThisComponent.Sheets.getByIndex(0)
    oSheet2 = ThisComponent.Sheets.getByIndex(1)

    REM Get the range to copy and the range to copy to.
    oRangeOrg = oSheet1.getCellRangeByName("A1:C10").RangeAddress
    oRangeCpy = oSheet2.getCellRangeByName("A1:C10").RangeAddress

    REM The insert position
    oCellCpy = oSheet2.getCellByPosition(oRangeCpy.StartColumn, _
        oRangeCpy.StartRow).CellAddress

    REM Do the copy
    oSheet1.CopyRange(oCellCpy, oRangeOrg)
End Sub
```

Unfortunately, this copy does not copy formatting and such.

5.23.3. Getting the content-type of the Clipboard

I created the following clip to demonstrate how to access the clipboard directly. In general, this is not practical for anything other than direct text manipulations.

Listing 5.63: *Manipulate the clipboard.*

```
Sub ConvertClipToText
    REM Author: Andrew Pitonyak
    Dim oClip, oClipContents, oTypes
    Dim oConverter, convertedString$
    Dim i%, iPlainLoc%
    Dim sClipService As String

    iPlainLoc = -1
    sClipService = "com.sun.star.datatransfer.clipboard.SystemClipboard"
    oClip = createUnoService(sClipService)
    oConverter = createUnoService("com.sun.star.script.Converter")

    'Print "Clipboard name = " & oClip.getName()
    'Print "Implementation name = " & oClip.getImplementationName()
    oClipContents = oClip.getContents()
    oTypes = oClipContents.getTransferDataFlavors()
```

```

Dim msg$, iLoc%, outs
msg = ""
iLoc = -1
For i=LBound(oTypes) To UBound(oTypes)
    If oTypes(i).MimeType = "text/plain;charset=utf-16" Then
        iPlainLoc = i
        Exit For
    End If
    'msg = msg & "Mime type = " & x(ii).MimeType
    'msg = msg & " normal = " & x(ii).HumanPresentableName & Chr$(10)
Next
If (iPlainLoc >= 0) Then
    Dim oData
    oData = oClipContents.getTransferData(oTypes(iPlainLoc))
    convertedString = oConverter.convertToSimpleType(oData, _
                                                    com.sun.star.uno.TypeClass.STRING)
    MsgBox convertedString
End If
End Sub

```

5.23.4. Storing a string to the clipboard

Here is an interesting example from ms777 from the oooforum that demonstrates how to write a string to the clipboard.

Listing 5.64: *Write a string to the clipboard.*

```

Private oTRX

Sub Main
    Dim null As Object
    Dim sClipName As String

    sClipName = "com.sun.star.datatransfer.clipboard.SystemClipboard"
    oClip = createUnoService(sClipName)
    oTRX = createUnoListener("TR_", "com.sun.star.datatransfer.XTransferable")
    oClipContents = oClip.setContents(oTRX, null)
End Sub

Function TR_getTransferData( aFlavor As com.sun.star.datatransfer.DataFlavor )
As Any
    If (aFlavor.MimeType = "text/plain;charset=utf-16") Then
        TR_getTransferData = "From OO with love ..."
    EndIf
End Function

Function TR_getTransferDataFlavors() As Any
    Dim aF As New com.sun.star.datatransfer.DataFlavor
    aF.MimeType = "text/plain;charset=utf-16"

```

```

    aF.HumanPresentableName = "Unicode-Text"
    TR_getTransferDataFlavors = Array(aF)
End Function

Function TR_isDataFlavorSupported( aFlavor As
com.sun.star.datatransfer.DataFlavor ) As Boolean

    'My XP system beep - shows that this routine is called every 2 seconds
    'call MyPlaySoundSystem("SystemAsterisk", true)
    TR_isDataFlavorSupported = (aFlavor.MimeType = "text/plain;charset=utf-16")
End Function

```

5.23.5. View the clipboard as text

Most people access the clipboard using UNO dispatch commands. Sometimes, however, you need to access the clipboard directly. Listing 5.65 demonstrates how to access the clipboard as text. Too busy to explain how this code works.

Listing 5.65: View the clipboard as text.

```

Sub ViewClipboard
    Dim oClip, oClipContents, oTypes
    Dim oConverter, convertedString$
    Dim i%, iPlainLoc%

    iPlainLoc = -1

    Dim s$ : s$ = "com.sun.star.datatransfer.clipboard.SystemClipboard"
    oClip = createUnoService(s$)
    oConverter = createUnoService("com.sun.star.script.Converter")

    'Print "Clipboard name = " & oClip.getName()
    'Print "Implementation name = " & oClip.getImplementationName()
    oClipContents = oClip.getContents()
    oTypes = oClipContents.getTransferDataFlavors()

    Dim msg$, iLoc%, outS
    msg = ""
    iLoc = -1
    For i=LBound(oTypes) To UBound(oTypes)
        If oTypes(i).MimeType = "text/plain;charset=utf-16" Then
            iPlainLoc = i
            Exit For
        End If
    Next
    'msg = msg & "Mime type = " & x(ii).MimeType & " normal = " & _
    '      x(ii).HumanPresentableName & Chr$(10)
    Next
    If (iPlainLoc >= 0) Then
        convertedString = oConverter.convertToSimpleType( _

```



```

        oClipContents.getTransferData(oTypes(iPlainLoc)), _
        com.sun.star.uno.TypeClass.STRING)
    MsgBox convertedString
End If
End Sub

```

5.23.6. An alternative to the clipboard – transferable content

Sometime after version 2.0, the controller for Write introduced `getTransferable()` and `insertTransferable()`, which acts like an internal clipboard. The following macro uses a dispatch to select the entire document, creates a new Write document, and then copies all of the text content into the new document.

Listing 5.66: Copy a text document using transferable content.

```

oFrame = ThisComponent.CurrentController.Frame
dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
dim noargs()
dispatcher.executeDispatch(frame, ".uno:SelectAll", "", 0, noargs())

obj = frame.controller.getTransferable()
sURL = "private:factory/swriter"
doc = stardesktop.loadcomponentfromurl(sURL, "_blank", 0, noargs())
doc.currentController.insertTransferable(obj)

```

Support for transferable content will be supported in Calc as of version 2.3.

5.24. Setting The Locale (Language)

In OOo, characters contain a locale, which identifies the language and country. I use styles to format my macro code samples. I set the locale in the macro code styles to unknown so that their spelling is not checked – if the locale is not known, then OOo does not know which dictionary to use. To tell OOo that a word is French, you set the locale of the characters to French. I was asked how to set the locale for all of the text in a document to a single value. This seemed obvious at first. A cursor supports character properties which allows you to set the locale. I created a cursor, selected the entire document, and then set the locale. I received a runtime error. I found out that the locale property is optional – it may be empty, as in `IsEmpty(oCurs.CharLocale)` is true. Although my next try worked for my document, you should perform more testing with tables and other things. It is safer to use an enumeration, because an enumeration can enumerate sections that all use the same property values so you can then always set the locale.

Listing 5.67: Set the document locale.

```

Sub SetDocumentLocale
    Dim oCursor
    Dim aLocale As New com.sun.star.lang.Locale
    aLocale.Language = "fr"
    aLocale.Country = "FR"

```

```

REM This assumes a text document
REM Get the Text component from the document
REM Create a Text cursor
oCursor = ThisComponent.Text.createTextCursor()
REM Goto the start of the document
REM Then, goto the end of the document selecting ALL the text
oCursor.GoToStart(False)
Do While oCursor.gotoNextParagraph(True)
    oCursor.CharLocale = aLocale
    oCursor.goRight(0, False)
Loop
Msgbox "successfully francophonized"
End Sub

```

It may be prudent to add the line “On Local Error Resume Next”, but I did not try it and it would hide any errors during your initial testing.

You should be able to set the locale for selected text or text that was found using the built in search routines as well.

5.25. Setting the locale for selected text

To demonstrate a slightly different method, I the following macro sets the locale for selected text, or for the entire document. I removed most of the comments. See the section dealing with selected text in a text document. The macro in Listing 5.67 iterates through the document using a paragraph cursor. I opted to not use a paragraph cursor in Listing 5.68, because the selected text may not include an entire paragraph. The primary concern with this method, is that a very large document may take a lot of time to iterate through one character at a time.

Listing 5.68: *Set the locale for selected text (or the document).*

```

Sub MainSetLocale
    Dim oLoc As New com.sun.star.lang.Locale
    'oLoc.Language = "fr" : oLoc.Country = "FR"
    oLoc.Language = "en" : oLoc.Country = "US"
    SetLocaleForDoc(ThisComponent, oLoc)
End sub

Sub SetLocaleForDoc(oDoc, oLoc)
    Dim oCurs()
    Dim sPrompt$
    Dim i%

    sPrompt = "Set locale to (" & oLoc.Language & ", " & oLoc.Country & ")? "
    If NOT CreateSelTextIterator(oDoc, sPrompt, oCurs()) Then Exit Sub
    For i = LBound(oCurs()) To UBound(oCurs())
        SetLocaleForCurs(oCurs(i, 0), oCurs(i, 1), oLoc)
    Next i
End Sub

```

```

    Next
End Sub

Sub SetLocaleForCurs(oLCurs, oRCurs, oLoc)
    Dim oText

    If IsNull(oLCurs) OR IsNull(oRCurs) Then Exit Sub
    If IsEmpty(oLCurs) OR IsEmpty(oRCurs) Then Exit Sub

    oText = oLCurs.getText()

    If oText.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Sub
    oLCurs.goRight(0, False)
    Do While oLCurs.goRight(1, True) AND _
        oText.compareRegionEnds(oLCurs, oRCurs) >= 0
        oLCurs.CharLocale = oLoc
        oLCurs.goRight(0, False)
    Loop
End Sub

Function CreateSelTextIterator(oDoc, sPrompt As String, oCurs()) As Boolean
    Dim lSelCount As Long 'Number of selected sections.
    Dim lWhichSel As Long 'Current selection item.
    Dim oSels 'All of the selections
    Dim oLCurs As Object 'Cursor to the left of the current selection.
    Dim oRCurs As Object 'Cursor to the right of the current selection.

    CreateSelTextIterator = True
    If Not IsAnythingSelected(oDoc) Then
        Dim i%
        i% = MsgBox("No text selected!" + Chr(13) + sPrompt, _
            1 OR 32 OR 256, "Warning")
        If i% = 1 Then
            oLCurs = oDoc.getText().createTextCursor()
            oLCurs.gotoStart(False)
            oRCurs = oDoc.getText().createTextCursor()
            oRCurs.gotoEnd(False)
            oCurs = DimArray(0, 1)
            oCurs(0, 0) = oLCurs
            oCurs(0, 1) = oRCurs
        Else
            oCurs = DimArray()
            CreateSelTextIterator = False
        End If
    Else
        oSels = oDoc.getCurrentSelection()
        lSelCount = oSels.getCount()
    End If
End Function

```

```

        oCurs = DimArray(lSelCount - 1, 1)
        For lWhichSel = 0 To lSelCount - 1
            GetLeftRightCursors(oSels.getByIndex(lWhichSel), oLCurs, oRCurs)
            oCurs(lWhichSel, 0) = oLCurs
            oCurs(lWhichSel, 1) = oRCurs
        Next
    End If
End Function

Function IsAnythingSelected(oDoc) As Boolean
    Dim oSels 'All of the selections
    Dim oSel 'A single selection
    Dim oCursor 'A temporary cursor

    IsAnythingSelected = False
    If IsNull(oDoc) Then Exit Function
    oSels = oDoc.getCurrentSelection()
    If IsNull(oSels) Then Exit Function
    If oSels.getCount() = 0 Then Exit Function

    REM If there are multiple selections, then certainly something is selected
    If oSels.getCount() > 1 Then
        IsAnythingSelected = True
    Else
        oSel = oSels.getByIndex(0)
        oCursor = oSel.getText().CreateTextCursorByRange(oSel)
        If Not oCursor.IsCollapsed() Then IsAnythingSelected = True
    End If
End Function

Sub GetLeftRightCursors(oSel, oLeft, oRight)
    Dim oCursor
    If oSel.getText().compareRegionStarts(oSel.getEnd(), oSel) >= 0 Then
        oLeft = oSel.getText().CreateTextCursorByRange(oSel.getEnd())
        oRight = oSel.getText().CreateTextCursorByRange(oSel.getStart())
    Else
        oLeft = oSel.getText().CreateTextCursorByRange(oSel.getStart())
        oRight = oSel.getText().CreateTextCursorByRange(oSel.getEnd())
    End If
    oLeft.goRight(0, False)
    oRight.goLeft(0, False)
End Sub

```

5.26. Auto Text

I have not tested this code, but I have been assured that it works. You will not be able to use the code as written because it requires a dialog that you do not have, but the techniques used will be useful just the same. Some links that I found include:

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XAutoTextGroup.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/text/AutoTextGroup.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/text/AutoTextContainer.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XAutoTextContainer.html>

Listing 5.69: Using auto text.

```
'Author: Marc Messeant
'email: marc.liste@free.fr
'To copy one AutoText From a group to an other one
'ListBox1 : The initial group
'ListBox2 : the Destination Group
'ListBox3 : The Element of the initial group to copy
'ListBox4 : The Element of the Destination group (for information only)

Dim ODialog as object
Dim oAutoText as object

' This subroutine opens the Dialog and initialize the lists of Group

Sub OuvrirAutoText
    Dim aTableau() as variant
    Dim i as integer
    Dim oListGroupDepart as object, oListGroupArrivee as object

    oDialog = LoadDialog("CG95", "DialogAutoText")
    oListGroupDepart = oDialog.getControl("ListBox1")
    oListGroupArrivee = oDialog.getControl("ListBox2")
    oAutoText = createUnoService("com.sun.star.text.AutoTextContainer")
    aTableau = oAutoText.getElementNames()
    oListGroupDepart.removeItem(0, oListGroupDepart.getItemCount())
    oListGroupArrivee.removeItem(0, oListGroupArrivee.getItemCount())
    For i = LBound(aTableau()) To UBound(aTableau())
        oListGroupDepart.addItem(aTableau(i), i)
        oListGroupArrivee.addItem(aTableau(i), i)
    Next
    oDialog.Execute()
End Sub

'The 3 routines are called when the user selects one group to
'initialize the lists of AutoText elements for each group
Sub ChargerList1()
    ChargerListeGroupe("ListBox1", "ListBox3")
End Sub
Sub ChargerList2()
    ChargerListeGroupe("ListBox2", "ListBox4")
End Sub
```

```

Sub ChargerListeGroupe(ListGroupe as string,ListElement as string)
    Dim oGroupe as object
    Dim oListGroupe as object
    Dim oListElement as object
    Dim aTableau() as variant
    Dim i as integer

    oListGroupe = oDialog.getControl(ListGroupe)
    oListElement = oDialog.getControl(ListElement)
    oGroupe = oAutoText.getByIndex(oListGroupe.getSelectedItemPos())
    aTableau = oGroupe.getTitles()
    oListElement.removeItem(0,oListElement.getItemCount())
    For i = LBound(aTableau()) To UBound(aTableau())
        oListElement.addItem(aTableau(i),i)
    Next
End Sub

'This routine transfer one element of one group to an other one
Sub TransfererAutoText()
    Dim oGroupDepart as object,oGroupArrivee as object
    Dim oListGroupDepart as object, oListGroupArrivee as object
    Dim oListElement as object
    Dim oElement as object
    Dim aTableau() as string
    Dim i as integer

    oListGroupDepart = oDialog.getControl("ListBox1")
    oListGroupArrivee = oDialog.getControl("ListBox2")
    oListElement = oDialog.getControl("ListBox3")
    i =oListGroupArrivee.getSelectedItemPos()
    If oListGroupDepart.getSelectedItemPos() = -1 Then
        MsgBox ("Vous devez sélectionner un groupe de départ")
        Exit Sub
    End If
    If oListGroupArrivee.getSelectedItemPos() = -1 Then
        MsgBox ("Vous devez sélectionner un groupe d'arrivée")
        Exit Sub
    End If
    If oListElement.getSelectedItemPos() = -1 Then
        MsgBox ("Vous devez sélectionner un élément à copier")
        Exit Sub
    End If
    oGroupDepart = oAutoText.getByIndex(oListGroupDepart.getSelectedItemPos())
    oGroupArrivee = oAutoText.getByIndex(oListGroupArrivee.getSelectedItemPos())
    aTableau = oGroupDepart.getElementNames()
    oElement = oGroupDepart.getByIndex(oListElement.getSelectedItemPos())
    If oGroupArrivee.HasByName(aTableau(oListElement.getSelectedItemPos())) Then

```

```

    MsgBox ("Cet élément existe déjà")
Exit Sub
End If
oGroupArrivee.insertNewByName(aTableau(oListElement.getSelectedItemPos()), _
    oListElement.getSelectedItem(), oElement.Text)
ChargerListeGroupe("ListBox2", "ListBox4")
End Sub

```

5.27. Decimal Feet To Fraction

I modified this code March 31, 2011 to better meet my needs as a woodworker.

I was asked to convert some Microsoft Office Macros to OOo Macros. I decided to improve them. The first set took a decimal number of feet and converted this to feet and inches in fractions. I decided to produce some general routines and ignore the existing code. This also avoided a few bugs that I found in the existing code. The quickest method that I know to reduce a fraction is to find the GCD (Greatest Common Divisor). The fraction macro calls GCD to simplify the fraction.

Listing 5.70: Calculate the GCD

```

'Author: Olivier Bietzer
'e-mail: olivier.bietzer@free.fr
'This uses Euclide's algorithm and it is very fast!
Function GCD(ByVal x As Long, ByVal y As Long) As Long
    Dim pgcd As Long, test As Long

    ' We must have x >= y and positive values
    x=abs(x)
    y=abs(y)
    If (x < y) Then
        test = x : x = y : y = test
    End If
    If y = 0 Then Exit Function

    ' Euclide says ....
    pgcd = y          ' by definition, PGCD is the smallest
    test = x MOD y    ' rest of division
    Do While (test)   ' While not 0
        pgcd = test    ' pgcd is the rest
        x = y          ' x,y and current pgcd permutation
        y = pgcd
        test = x MOD y ' test again
    Loop
    GCD = pgcd        ' pgcd is the last non 0 rest ! Magic ...
End Function

```

The following macro determines the fraction. If x is negative, then both the numerator and the returned value of x are negative on output. Note that the parameter x is modified.

Listing 5.71: Convert a double to a fraction.

```
'n: on output, contains the numerator
'd: on output, contains the denominator
'x: Input x to turn into a fraction, output the integer portion
'max_d: Maximum denominator
Sub ToFraction(n&, d&, x#, ByVal max_d As Long)
    Dim neg_multiply&, y#
    n = 0 : d = 1 : neg_multiply = 1 : y = Fix(x)
    If (x < 0) Then
        x = -x : neg_multiply = -1
    End If

    n = (x - ABS(y)) * max_d
    d = max_d

    REM Just in case x does not contain a fraction
    If (n <> 0) Then
        d = GCD(n, max_d)
        n = neg_multiply * n / d
        d = max_d / d
        x = y
    Else
        n = neg_multiply
        d = 1
    End If
    x = y
End Sub
```

I like the following utility function to simply convert a number into a fraction string.

```
Function ToFractionString(x#, ByVal max_d As Long) As String
    Dim numerator&, denominator&
    ToFraction(numerator, denominator, x, max_d)
    If numerator = denominator Then
        ToFractionString = "1"
    ElseIf numerator = -denominator Then
        ToFractionString = "-1"
    Else
        ToFractionString = CStr(numerator) & "/" & CStr(denominator)
    End If
End Function
```

To test this routine, I created the following test code.

```
Sub FractionTest
    Dim x#, inc#, first#, last#, y#, z#, epsilon#
    Dim d&, n&, max_d&
    first = -10 : last = 10 : inc = 0.001
    max_d = 128
```



```

epsilon = 1.0 / CDBl(max_d)
For x = first To last Step inc
    y = x
    ToFraction(n, d, y, max_d)
    z = y + CDBl(n) / CDBl(d)
    If abs(x-z) > epsilon Then Print "Incorrectly Converted " & x & " to " & z
Next
End Sub

```

Although I pretty much ignored the starting code, I wanted to preserve the input and output formats from the initial macro even if they are nothing alike.

Listing 5.72: Convert decimal feet to a string.

```

Function DecimalFeetToString(ByVal x#, ByVal max_denominator&) As String
    Dim numerator&, denominator&
    Dim feet#, decInch#, s As String

    s = ""
    If (x < 0) Then
        s = "-"
        x = -x
    End If

    feet = Fix(x)      'Whole Feet
    x = (x - feet) * 12 'Inches
    ToFraction(numerator, denominator, x, maxDenominator)
    REM Handle some rounding issues
    If (numerator = denominator AND numerator <> 0) Then
        numerator = 0
        x = x + 1
    End If

    If feet = 0 AND x = 0 AND numerator = 0 Then
        s = s & "0"
    Else
        If feet <> 0 Then
            s = s & feet & " "
            If x <> 0 OR numerator <> 0 Then s = s & "-"
        End If
        If x <> 0 Then
            s = s & x
            If numerator <> 0 Then s = s & " "
        End If
        If numerator <> 0 Then s = s & numerator & "/" & denominator
        If x <> 0 OR numerator <> 0 Then s = s & ""
    End If
    DecimalToString = s
End Function

```

```

Function DecimalFeetToString(ByVal x#, ByVal maxDenominator&, ByVal includeFeet
As Boolean, ByVal includeInches As Boolean) As String
    DecimalFeetToString = DecimalInchesToString(x * 12.0, maxDenominator,
includeFeet, includeInches)
End Function

```

```

Function DecimalInchesToString(ByVal x#, ByVal maxDenominator&, ByVal
includeFeet As Boolean, ByVal includeInches As Boolean) As String

```

```

    Dim numerator&      : numerator = 0
    Dim denominator&    : denominator = 1
    Dim s As String      : s = ""
    Dim feet#
    Dim inch#

```

```

    If (x < 0) Then
        s = "-"
        x = -x
    End If

```

```

    If includeFeet AND NOT includeInches Then
        x = x / 12.0
        feet = Fix(x)
        x = x - feet
        ToFraction(numerator, denominator, x, maxDenominator)
        REM Handle some rounding issues
        If (numerator = denominator AND numerator <> 0) Then
            numerator = 0
            feet = feet + 1
        End If
        If feet <> 0 OR numerator = 0 Then
            s = s & feet
        End If
        If numerator <> 0 Then
            s = s & " " & numerator & "/" & denominator
        End If
        s = s & ""
    End If

```

```

    ElseIf NOT includeFeet AND includeInches Then
        inch = Fix(x)
        x = x - inch
        ToFraction(numerator, denominator, x, maxDenominator)
        REM Handle some rounding issues
        If (numerator = denominator AND numerator <> 0) Then
            numerator = 0
            inch = inch + 1
        End If
        If inch <> 0 OR numerator = 0 Then

```

```

        s = s & inch
    End If
    If numerator <> 0 Then
        If inch <> 0 Then
            s = s & " "
        End If
        s = s & numerator & "/" & denominator
    End If
    s = s & """"
ElseIf includeFeet AND includeInches Then
    feet = Fix(x / 12.0)
    x = x - feet * 12.0
    inch = Fix(x)
    x = x - inch
    ToFraction(numerator, denominator, x, maxDenominator)
    REM Handle some rounding issues
    If feet = 0 AND inch = 0 AND numerator = 0 Then
        s = "0""""
    Else
        Dim ss$ : ss = ""
        If feet <> 0 Then
            ss = ss & Feet & ""
        End If
        If inch <> 0 OR numerator <> 0 Then
            If inch <> 0 Then
                If ss <> "" Then
                    ss = ss & " "
                End If
                ss = ss & inch
            End If
            If numerator <> 0 Then
                If ss <> "" Then
                    ss = ss & " "
                End If
                ss = ss & numerator & "/" & denominator
            End If
            ss = ss & """"
        End If
        s = s & ss
    End If
Else
    s = ""
End If
DecimalInchesToString = s
End Function

Function StringToDecimalInches(s$) As Double

```

```
StringToDecimalInches = StringToDecimalFeet(s) * 12.0
End Function
```

```
Function StringToDecimalFeet(s$) As Double
    REM Maximum number of tokens would include
    REM <feet><'><space><inches><space><numerator></><denominator><">
    REM The first token MUST be a number!
    Dim tokens(8) As String '0 to 8
    Dim i%, j%, num_tokens%, c%
    Dim feet#, inches#, n#, d#, leadingNeg#

    Dim numbers(0 To 4) As Long
    Dim iNum%
    Dim bFracFound As Boolean

    bFracFound = False
    iNum = -1
    feet = 0 : inches = 0 : n = 0 : d = 1 : i = 1 : leadingNeg = 1.0
    s = Trim(s) ' Lose leading and trailing spaces
    If (Len(s) > 0) Then
        If Left(s, 1) = "-" Then
            leadingNeg = -1.0
            s = Mid(s, 2)
        End If
    End If

    i = 1
    Do While i <= Len(s)
        Select Case Mid(s, i, 1)
            Case " "
                i = i + 1
            Case "0" To "9"
                j = i
                c = Asc(Mid(s, j, 1))
                Do While (48 <= c AND c <= 57)
                    j = j + 1
                    If j > Len(s) Then Exit Do
                    c = Asc(Mid(s, j, 1))
                Loop
                If bFracFound Then
                    'Must be the denominator
                    d = CLng(Mid(s, i, j-i))
                    bFracFound = False
                Else
                    iNum = iNum + 1
                    numbers(iNum) = CLng(Mid(s, i, j-i))
                End If
            Case "<"/>"/>

```

```

        End If
        i = j
    Case "'"
        i = i + 1
        If iNum >= 0 Then
            feet = numbers(iNum)
        End If
        iNum = 0
        if n <> 0 Then
            feet = feet + n / d
        End If
    Case "/"
        ' Assume that the previous number was the numerator.
        i = i + 1
        bFracFound = True
        If iNum < 0 Then
            'Error in input, there should be at least one number
            Exit Function
        End If
        n = numbers(iNum)
        iNum = iNum - 1

    Case """"
        i = i + 1
        If iNum >= 0 Then
            inches = numbers(iNum)
        End If
        iNum = 0
        if n <> 0 Then
            inches = inches + n / d
        End If
    Case Else
        'Hmm, this is an error
        i = i + 1
        Print "In the else"
    End Select
Loop
StringToDecimalFeet = leadingNeg * (feet + inches/12.0)
End Function

```

5.27.1. Convert number to words

Converting an integer with values from 0 to 999 into words is simple. Some different processing is required for languages other than English, however.

Listing 5.73: Convert 0 to 999 into words

```

Function SmallIntToText(ByVal n As Integer) As String
    REM by Andrew D. Pitonyak

```

```

Dim sOneWords()
Dim sTenWords()
Dim s As String

If n > 999 Then
    Print "Warning, n = " & n & " which is too large!"
    Exit Function
End If

sOneWords() = Array("zero", _
    "one",      "two",      "three",    "four",    "five", _
    "six",      "seven",    "eight",   "nine",    "ten", _
    "eleven",   "twelve",   "thirteen", "fourteen", "fifteen", _
    "sixteen",  "seventeen", "eighteen", "nineteen", "twenty")

sTenWords() = Array( "zero", "Ten", "twenty", "thirty", "fourty", _
    "fifty", "sixty", "seventy", "eighty", "ninety")

s = ""
If n > 99 Then
    s = sOneWords(Fix(n / 100)) & " hundred"
    n = n MOD 100
    If n = 0 Then
        SmallIntToText = s
        Exit Function
    End If
    s = s & " "
End If

If (n > 20) Then
    s = s & sTenWords(Fix(n / 10))
    n = n MOD 10
    If n = 0 Then
        SmallIntToText = s
        Exit Function
    End If
    s = s & " "
End If

SmallIntToText = s & sOneWords(n)
End Function

```

Converting larger numbers is a bit more work. I included the names as used in the USA, UK, and Germany. I do not handle decimal values or negative numbers. This is only a macro to get you started!

Listing 5.74: Convert big numbers into words

```
Function NumberToText(ByVal n) As String
```

```

REM by Andrew D. Pitonyak
Dim sBigWordsUSA()
Dim sBigWordsUK()
Dim sBigWordsDE()

REM 10^100 is a googol (10 duotrigintillion)
REM This goes to 10^303, how big do you really want me to go?
sBigWordsUSA = Array( "", _
    "thousand", "million", "billion", "trillion", "quadrillion", _
    "quintillion", "sexillion", "septillion", _
    "octillion", "nonillion", "decillion", "undecillion", "duodecillion", _
    "tredecillion", "quattuordecillion", "quindecillion", "sexdecillion", _
    "septdecillion", "octodecillion", "novemdecillion", "vigintillion", _
    "unvigintillion", "duovigintillion", "trevigintillion", _
    "quattuorvigintillion", "quinvigintillion", "sexvigintillion", _
    "septvigintillion", "octovigintillion", "novemvigintillion", _
    "trigintillion", "untrigintillion", "duotrigintillion", _
    "tretrigintillion", "quattuortrigintillion", "quintrigintillion", _
    "sextrigintillion", "septtrigintillion", "octotrigintillion", _
    "novemtrigintillion", "quadragintillion", "unquadragintillion", _
    "duoquadragintillion", "trequadragintillion", _
    "quattuorquadragintillion", _
    "quinquadragintillion", "sexquadragintillion", "septquadragintillion", _
    "octoquadragintillion", "novemquadragintillion", "quinguagintillion", _
    "unquinguagintillion", "duoquinguagintillion", "trequinguagintillion", _
    "quattuorquinguagintillion", "quinquagintillion", _
    "sexquinguagintillion", "septquinguagintillion", _
    "octoquinguagintillion", _
    "novemquinguagintillion", "sexagintillion", "unsexagintillion", _
    "duosexagintillion", "tresexagintillion", "quattuorsexagintillion", _
    "quinsexagintillion", "sexsexagintillion", "septsexagintillion", _
    "octosexagintillion", "novemsexagintillion", "septuagintillion", _
    "unseptuagintillion", "duoseptuagintillion", "treseptuagintillion", _
    "quattuorseptuagintillion", "quinseptuagintillion", _
    "sexseptuagintillion", _
    "septseptuagintillion", "octoseptuagintillion", "novemseptuagintillion", _
    "octogintillion", "unoctogintillion", "duooctogintillion", _
    "treoctogintillion", "quattuoroctogintillion", "quinoctogintillion", _
    "sexoctogintillion", "septoctogintillion", "octooctogintillion", _
    "novemoctogintillion", "nonagintillion", "unnonagintillion", _
    "duononagintillion", "trenonagintillion", "quattuornonagintillion", _
    "quinnonagintillion", "sexnonagintillion", "septnonagintillion", _
    "octononagintillion", "novemnonagintillion", "centillion" _
)

sBigWordsUK() = Array( "", "thousand", _
    "milliard", "billion", "billiard", "trillion", "trilliard", _

```

```

"quadrillion", "quadrilliard", "quintillion", "quintilliard", _
"sextillion", "sextilliard", "septillion", "septilliard", _
"octillion", "octilliard", "nonillion", "nonilliard", "decillion", _
"decilliard", "undecillion", "undecilliard", "dodecillion", _
"dodecilliard", "tredecillion", "tredecilliard", "quattuordecillion", _
"quattuordecilliard", "quindecillion", "quindecilliard", "sexdecillion", _
"sexdecilliard", "septendecillion", "septendecilliard", "octodecillion", _
"octodecilliard", "novemdecillion", "novemdecilliard", "vigintillion", _
"vigintilliard", "unvigintillion", "unvigintilliard", "duovigintillion", _
"duovigintilliard", "trevigintillion", "trevigintilliard", _
"quattuorvigintillion", "quattuorvigintilliard", "quinvigintillion", _
"quinvigintilliard", "sexvigintillion", "sexvigintilliard", _
"septenvigintillion", "septenvigintilliard", "octovigintillion", _
"octovigintilliard", "novemvigintillion", "novemvigintilliard", _
"trigintillion", "trigintilliard", "untrigintillion", _
"untrigintilliard", "duotrigintillion", "duotrigintilliard", _
"tretrigintillion", "tretrigintilliard", "quattuortrigintillion", _
"quattuortrigintilliard", "quintrigintillion", "quintrigintilliard", _
"sextrigintillion", "sextrigintilliard", "septentrigintillion", _
"septentrigintilliard", "octotrigintillion", "octotrigintilliard", _
"novemtrigintillion", "novemtrigintilliard", "quadragintillion", _
"quadragintilliard", "unquadragintillion", "unquadragintilliard", _
"duoquadragintillion", "duoquadragintilliard", "trequadragintillion", _
"trequadragintilliard", "quattuorquadragintillion", _
"quattuorquadragintilliard", "quinquadragintillion", _
"quinquadragintilliard", "sexquadragintillion", "sexquadragintilliard", _
"septenquadragintillion", "septenquadragintilliard", _
"octoquadragintillion", "octoquadragintilliard", _
"novemquadragintillion", "novemquadragintilliard", "quinguagintillion", _
"quinguagintilliard" _
)

sBigWordsDE() = Array( "", "Tausand", _
"Milliarde", "Billion", "Billiarde", "Trillion", "Trilliarde", _
"Quadrillion", "Quadrilliarde", "Quintillion", "Quintilliarde", _
"Sextillion", "Sextilliarde", "Septillion", "Septilliarde", _
"Oktilion", "Oktilliarde", "Nonillion", "Nonilliarde", _
"Dezillion", "Dezilliarde", "Undezillion", "Undezilliarde", _
"Duodezillion", "Doudezilliarde", "Tredezillion", _
"Tredizilliarde", "Quattuordezillion", "Quattuordezilliarde", _
"Quindezillion", "Quindezilliarde", "Sexdezillion", _
"Sexdezilliarde", "Septendezillion", "Septendezilliarde", _
"Oktodezillion", "Oktodezilliarde", "Novemdezillion", _
"Novemdezilliarde", "Vigintillion", "Vigintilliarde", _
"Unvigintillion", "Unvigintilliarde", "Duovigintillion", _
"Duovigintilliarde", "Trevigintillion", "Trevigintilliarde", _
"Quattuorvigintillion", "Quattuorvigintilliarde", "Quinvigintillion", _

```



```

    "Quinvigintilliarde", "Sexvigintillion", "Sexvigintilliarde", _
    "Septenvigintillion", "Septenvigintilliarde", "Oktovigintillion", _
    "Oktovigintilliarde", "Novemvigintillion", "Novemvigintilliarde", _
    "Trigintillion", "Trigintilliarde", "Untrigintillion", _
    "Untrigintilliarde", "Duotrigintillion", "Duotrigintilliarde", _
    "Tretigintillion", "Tretigintilliarde", "Quattuortrigintillion", _
    "Quattuortrigintilliarde", "Quintrigintillion", "Quintrigintilliarde", _
    "Sextrigintillion", "Sextrigintilliarde", "Septentrigintillion", _
    "Septentrigintilliarde", "Oktotrigintillion", "Oktotrigintilliarde", _
    "Novemtrigintillion", "Novemtrigintilliarde", "Quadragintillion", _
    "Quadragintilliarde", "Unquadragintillion", "Unquadragintilliarde", _
    "Duoquadragintillion", "Duoquadragintilliarde", "Trequadragintillion", _
    "Trequadragintilliarde", "Quattuorquadragintillion", _
    "Quattuorquadragintilliarde", "Quinquadragintillion", _
    "Quinquadragintilliarde", "Sexquadragintillion", _
    "Sexquadragintilliarde", "Septenquadragintillion", _
    "Septenquadragintilliarde", "Oktoquadragintillion", _
    "Oktoquadragintilliarde", "Novemquadragintillion", _
    "Novemquadragintilliarde", "Quinquagintillion", "Quinquagintilliarde" _
)

Dim i As Integer
Dim iInt As Integer
Dim s As String
Dim dInt As Double

REM Chop off the decimal portion.
dInt = Fix(n)
If (dInt < 1000) Then
    NumberToText = SmallIntToText(CInt(dInt))
    Exit Function
End If

REM i is the index into the sBigWords array
i = 0
s = ""
Do While dInt > 0
    iInt = CInt(dInt - Fix(dInt / 1000) * 1000)
    If iInt <> 0 Then
        If Len(s) > 0 Then s = " " & s
        s = SmallIntToText(iInt) & " " & sBigWordsUSA(i) & s
    End If
    i = i + 1
    dInt = Fix(dInt / 1000)
    'Print "s = " & s & " dInt = " & dInt
Loop

```

```

    NumberToText = s
End Function

```

The following macro is an example that converts numbers to text. The accepted form is “\$123,453,223.34”. Leading dollar sign is removed. All commas are removed. The decimal splits the dollars from the cents.

Listing 5.75: Convert US currency to words.

```

Function USCurrencyToWords(s As String) As String
    Dim sDollars As String
    Dim sCents As String
    Dim i%

    If (s = "") Then s = "0"
    If (Left(s, 1) = "$") Then s = Right(s, Len(s) - 1)

    If (InStr(s, ".") = 0) Then
        sDollars = s
        sCents = "0"
    Else
        sDollars = Left(s, InStr(s, ".")-1)
        sCents = Right(s, Len(s) - InStr(s, "."))
    End If

    Do While (InStr(sDollars, ",") > 0)
        i = InStr(sDollars, ",")
        sDollars = Left(sDollars, i-1) & Right(sDollars, Len(sDollars) - i)
    Loop

    If (sDollars = "") Then sDollars = "0"
    If (sCents = "") Then sCents = "0"
    If (Len(sCents) = 1) Then sCents = sCents & "0"

    USCurrencyToWords = NumberToText(sDollars) & " Dollars and " & _
        NumberToText(sCents) & " Cents"
End Function

```

5.28. Sending Email

OOo provides a means of sending email but it must be properly configured, especially form Linux. OOo uses an existing client email program rather than directly supporting the email protocols. On first installation, it should know how to use some common email clients such as Mozilla/Netscape, Evolution, and K-Mail. On Windows, OOo uses MAPI so all MAPI

compatible clients should work. You need to use “com.sun.star.system.SimpleSystemMail”. The SimpleCommandMail uses system command line tools to send mail, but I have only gotten this service to work on Linux. The following example was provided by Laurent Godard.

Listing 5.76: Send email.

```
Sub SendSimpleMail()  
    Dim vMailSystem, vMail, vMessage  
    'vMailSystem = createUnoService( "com.sun.star.system.SimpleCommandMail" )  
    vMailSystem=createUnoService("com.sun.star.system.SimpleSystemMail")  
    vMail=vMailSystem.querySimpleMailClient()  
    vMessage = vMail.createsimplEmailMessage()  
    vMessage.setrecipient("andrew@pitonyak.org")  
    vMessage.setsubject("This is my test subject")  
  
    'Attachments are set by a sequence which in basic means an array  
    'I could use ConvertToURL() to build the URL!  
    Dim vAttach(0)  
    vAttach(0) = "file:///c:/macro.txt"  
    vMessage.setAttachement(vAttach())  
  
    'DEFAULTS Launch the currently configured system mail client.  
    'NO_USER_INTERFACE Do not show the interface, just do it!  
    'NO_LOGON_DIALOG No logon dialog but will throw an exception if one  
    ' is required.  
    vMail.sendSimpleMailMessage(vMessage, _  
                                com.sun.star.system.SimpleMailClientFlags.NO_USER_INTERFACE)  
End Sub
```

Neither the SimpleSystemMail, nor the SimpleCommandMail service are able to send an email text body. According to Mathias Bauer, the intent of these services was to deliver a document as an attachment. It is possible to use a “mailto” URL, to send an email message with a text body, but this does not contain an attachment. The idea is let the operating system pass the mailto URL to the default object that can hopefully parse the entire text. Support for this method are dependent upon the operating system and the installed software.

Listing 5.77: Send email using a URL.

```
Dim noargs()  
email_dispatch_url = "mailto:demo@someplace.com?subject=Test&Body=Text"  
dispatcher = createUnoService( "com.sun.star.frame.DispatchHelper")  
dispatcher.executeDispatch( StarDesktop,email_dispatch_url, "", 0, noargs())
```

According to Daniel Juliano (daniel.juliano@rainhail.com), the message size is limited by the operating system. With Windows 2000, the limit seems to be close to 500 characters. If the size is exceeded, the email is not sent and an error does NOT occur. (Andrew Pitonyak suspects that the message size is limited because it is sent as a command line. Different

command interpreters support different command line lengths. For example, 4NT probably supports a longer command line than the command line provided by Microsoft.)

If running on Windows using Outlook, you can easily send body text and attachments as you desire.

Listing 5.78: Send email using Microsoft Outlook

```
Sub UseOutlook( )
    Dim oOLEService
    Dim oOutlookApp
    Dim oOutlookMail

    oOLEService = createUnoService("com.sun.star.bridge.OleObjectFactory")
    oOutlookApp = oOLEService.CreateInstance("Outlook.Application")
    oOutlookMail = oOutlookApp.CreateItem(0)

    REM I can directly set the recipients by setting the To property
    oOutlookMail.To = "andrew@pitonyak.org"

    REM I can also add to the list, but in my experiments, this access the
    REM mail box so Outlook asks me if I can do this. In other words, it then
    REM requires user interaction. I can probably set the security in outlook
    REM to simply allow this, but then I have opened things for virus activity.
    'oOutlookMail.Recipients.Add("andrew@pitonyak.org")

    oOutlookMail.Subject = "Test Subject"
    oOutlookMail.Body = "This is my body text for the email message"

    REM You can also add attachments to the message
    'oOutlookMail.Attachments.Add("C:\foo.txt")

    REM I can display and edit the message
    'oOutlookMail.Display()

    REM Or I can send the message
    'oOutlookMail.send()
End Sub
```

5.29. Macro libraries

This section discusses how to use and distribute (install) macro libraries.

5.29.1. The vocabulary

To understand libraries, you must understand the difference between a library container, a library, and a module.

5.29.1.1. Library container

A library container contains macro libraries. The OOo application contains two library containers, “OpenOffice.org Macros” and “My Macros”. Use **Tools > Macros > Organize Macros > OpenOffice.org Basic** to view the available library containers.

Macros distributed with OOo are stored in the OpenOffice.org Macros container and you should not modify them. You should store all of your macros in the My Macros container. Each document is also a library container and is visible as an available library container.

5.29.1.2. Libraries

A library contains modules. A library is used for high level grouping of functionality. For example, if I wanted to write a group of related macros and release them, I would probably store them all in the same library.

You can not run a macro contained in a library unless the library has already been loaded. You can load a library using the GUI, or from within a macro.

Every library container automatically has a library named Standard. The Standard library is always loaded. To guarantee that a specific macro is always available, store the macro in the Standard library. For example, I frequently store macros called by form controls in the Standard library. These “event handler” macros may then load other libraries and call macros in other libraries as required.

5.29.1.3. Modules

Modules contains macro subroutines and functions (and dialogs).

5.29.2. Where are libraries stored?

Assume that OOo is installed in “C:\Program Files\OpenOffice”. The “OpenOffice.org Macros” are stored in “C:\Program Files\OpenOffice\user\basic\”. Your macros are stored in a directory similar to “C:

\Documents and Settings\<user name>\Application Data\OpenOffice\user\basic\”. With Linux, your macros are stored off of your home directory under “.OpenOffice.org/user/basic”.

The directory contains the files Script.xlc and Dialog.xlc, which reference the libraries visible in OOo. If a library exists, but you can not see it, it is probably because of a problem in one of these two files.

Each library is represented as a directory with the same name as the library. Each library is referenced in Script.xlc and Dialog.xlc. Each library folder contains the modules with the .xba or .xdl filename extensions as well as script.xlb and dialog.xlb, which list the modules contained in the library. Each libraries is linked to a specific document or to the OOo application.

5.29.3. The library container

Prior to version 1.0, LibraryContainer was available in Basic and not in any other language. The “com.sun.star.script.ApplicationScriptLibraryContainer” service opens the libraries to languages other than Basic, but the service is not officially published – it is considered bad practice to use unpublished interfaces and services. See the “com.sun.star.script.XLibraryContainer” interface to learn how to use this service.

The BasicLibraries variable, available only from Basic, references the Basic libraries stored in ThisComponent. Likewise, the DialogLibraries variable references the dialog libraries stored in ThisComponent. The application level libraries are available using GlobalScope.BasicLibraries and GlobalScope.DialogLibraries.

Warning	The document's libraries are also available using the deprecated method getLibraryContainer() and its corresponding property LibraryContainer. This is also the only way to access the libraries in Basic, for a document that is NOT ThisComponent.

The following example demonstrates how to manipulate libraries using the ApplicationScriptLibraryContainer.

Listing 5.79: Using the ApplicationScriptLibraryContainer.

```
Sub LibContainer()  
    REM Christian Junker  
    Dim allLibs()  
    Const newlib As String = "dummy" 'Name of your new library  
    Dim sService As String  
    sService = "com.sun.star.script.ApplicationScriptLibraryContainer"  
    oLibCont = createUnoService(sService)  
  
    'create a new library  
    If (Not oLibCont.hasByName(newlib)) Then  
        oLibCont.CreateLibrary(newlib)  
    End If  
    'check if it is loaded, if not load it!  
    If (Not oLibCont.isLibraryLoaded(newlib)) Then  
        oLibCont.loadLibrary(newlib)  
    End If  
    'set a password for it (must not be read-only)  
    oLibCont.setLibraryReadOnly(newlib, False)  
    oLibCont.changeLibraryPassword(newlib, "", "password")  
    MsgBox "The password: ""password"" was set for library " & newlib  
    'show me all libraries including my new one:  
    allLibs = oLibCont.getElementNames()  
    ShowArray(allLibs()) 'This function is in the Tools Library  
    'Remove the library (must not be read-only)  
    oLibCont.removeLibrary(newlib)  
    'Show all libraries again, "dummy" was deleted
```

```

allLibs = oLibCont.getElementNames()
ShowArray(allLibs())
End Sub

```

Unfortunately, renaming a library during runtime did not work in this example.??

5.29.4. Warning about unpublished services

The ApplicationScriptLibraryContainer service is neither published nor documented. According to Jürgen Schmidt from Sun, There are probably good reasons that a service is not published. Although you found and can use the service, it may change because it is not officially published. In the future we will document unpublished APIs, but they will be marked and should be carefully used. We learned that it is sometimes better to have some experience with an API and obtain feedback before the API is published, because published means “not changeable”. Even the “best” design may require changes.

5.29.5. What does it means to load a Library?

When a library is loaded, the contained macros are made visible to the Basic engine. It is at this time, that the XML files are loaded and the macros are compiled. In other words, if a library is not loaded, you can not call the subroutines, functions, or dialogs that it contains. You do not want to load all of the libraries, because you usually do not use all of the macros and so it would waste space. The Standard library, however, is always loaded and available.

5.29.6. Distribute/deploy a library

Adding a macro to a document is the easiest way to share a library. If, however, you have numerous macros that you want to deploy for the entire application, the pkgchk tool might be preferred. The pkgchk is also used to register components that you have written in languages other than Basic. The simple explanation is that pkgchk packages (the abbreviation pkgchk means packagecheck) libraries into one collection which is stored as a .zip file in the “C:\Program Files\OpenOffice\user\uno-packages\” directory. If the “--shared” parameter is used, then the collection is stored in the “C:\Program Files\OpenOffice\share\uno-packages\” directory instead. Use the following steps to create the zip file:

Copy your library folder (or library folders) into a temporary directory.

Zip the libraries into using your favorite zip program. Be certain to preserve the directory structure.

Find the pkgchk program – it is located in the program directory off of the OpenOffice.org installation directory.

To install the “mymacros.zip” package run “pkgchk -shared mymacros.zip” – you probably need to provide the complete path to the file “mymacros.zip”. The macros should be installed in the shared UNO packages directory.

Some code written by Sunil Menon provides an example of this process using a macro.
[Andrew Pitonyak notes: I do this differently in my book using BasicLibraries and such]

Listing 5.80: Deploy a macro using the ApplicationScriptLibraryContainer.

```
'author:   Sunil Menon
'email:    sunil.menon@itb-india.com
service_name = "com.sun.star.script.ApplicationScriptLibraryContainer"
Set oLibLoad = objServiceManager.createInstance(service_name)
If Not oLibLoad Is Nothing Then
    On Error Resume Next
    If oLibLoad.isLibraryLoaded("mymacros") Then
        oLibLoad.removeLibrary ("mymacros")
    End If
    spath = "file:///D:/StarOfficeManual/mymacros"
    slib = "mymacros"
    Call oLibLoad.CreateLibraryLink(slib, spath, False)
    oLibLoad.loadLibrary ("mymacros")
    oLibLoad = Nothing
End If
```

If the macro already exists, then it must be registered again before the new library will be seen. This is accomplished by unloading and then reloading the library. The CreateLibraryLink method creates a link to an external library accessible using the library manager. The format of the StorageURL is implementation dependent. The boolean parameter is a read only flag.

5.30. Setting Bitmap Size

If you load an image, the size may not be as you desire. Vance Lankhaar first brought this problem to my attention. His first solution produced a very small image.

Listing 5.81: Insert a GraphicObjectShape.

```
'Author:   Vance Lankhaar
'email:    vlankhaar@linux.ca
Dim oDesktop As Object, oDoc As Object
Dim mNoArgs()
Dim sGraphicURL As String
Dim sGraphicService As String, sUrl As String
Dim oDrawPages As Object, oDrawPage As Object
Dim oGraphic As Object
sGraphicURL = "http://api.openoffice.org/branding/images/logonew.gif"
sGraphicService = "com.sun.star.drawing.GraphicObjectShape"
sUrl = "private:factory/simpres"
oDesktop = createUnoService("com.sun.star.frame.Desktop")
oDoc = oDesktop.loadComponentFromURL(sUrl, "_default", 0, mNoArgs())
oDrawPages = oDoc.DrawPages
oDrawPage = oDrawPages.insertNewByIndex(1)
```



```

oGraphic = oDoc.CreateInstance(sGraphicService)
oGraphic.GraphicURL = sGraphicURL
oDrawPage.add(oGraphic)

```

The first solution by Laurent Godard sets the size to the maximum allowable size.

Listing 5.82: *Set a graphic to the maximum supported size.*

```

'Maximum size, lose the aspect ration.
Dim TheSize As New com.sun.star.awt.Size
Dim TheBitmapSize As New com.sun.star.awt.Size
Dim TheBitmap as object
Dim xmult as double, ymult as double

TheBitmap=oGraphic.GraphicObjectFillBitmap
TheBitmapSize=TheBitmap.GetSize

xmult=TwipsPerPixelX/567*10*100
ymult=TwipsPerPixelY/567*10*100

TheSize.width=TheBitmapSize.width*xmult
TheSize.height=TheBitmapSize.height*ymult

oGraphic.setsize(TheSize)

```

Vance Lankhaar's final solution maximizes the size but preserves the aspect ratio.

Listing 5.83: *Set a graphic to the maximum supported size preserving the aspect ratio.*

```

oBitmap = oGraphic.GraphicObjectFillBitmap
aBitmapSize = oBitmap.GetSize
iWidth = aBitmapSize.Width
iHeight = aBitmapSize.Height

iPageWidth = oDrawPage.Width
iPageHeight = oDrawPage.Height
dRatio = CDbl(iHeight) / CDbl(iWidth)
dPageRatio = CDbl(iPageHeight) / CDbl(iPageWidth)

REM This is is fit-maximum-dimension
REM s/</>/ for fit-minimum-dimension
If (dRatio < dPageRatio) Then
    aSize.Width = iPageWidth
    aSize.Height = CInt(CDbl(iPageWidth) * dRatio)
Else
    aSize.Width = CInt(CDbl(iPageHeight) / dRatio)
    aSize.Height = iPageHeight
End If

aPosition.X = (iPageWidth - aSize.Width)/2

```

```

aPosition.Y = (iPageHeight - aSize.Height)/2

oGraphic.SetSize(aSize)
oGraphic.SetPosition(aPosition)

```

5.30.1. Insert, size, and position a graphic in a Calc document.

David Woody [dwoody1@airmail.net] needed to insert a graphics object at a specific position at a specific size. With a little help and a lot of work, he developed the following solution:

This reply took some time because I had another problem to solve with setting the correct value for the X and Y coordinates. The following code inserts a graphic, sizes it, and moves it to the desired location. I had to add the following line to the code in Andrew's macro book in section the section on setting bitmap size.

```
Dim aPosition As New com.sun.star.awt.Point
```

The other problem I had was that I had to determine the ratio that was needed for aPosition.X and aPosition.Y to properly position the graphic. On my computer the value of 2540 for either X or Y coordinate = 1 inch on the screen. The values below will put the graphic 1 inch down from the top of the sheet and 1 inch over from the left of the sheet.

Listing 5.84: Insert and position a graphic in a Calc document.

```

Sub InsertAndPositionGraphic
    REM Get the sheet
    Dim vSheet
    vSheet = ThisComponent.Sheets(0)

    REM Add the graphics object
    Dim oDesktop As Object, oDoc As Object
    Dim mNoArgs()
    Dim sGraphicURL As String
    Dim sGraphicService As String, sUrl As String
    Dim oDrawPages As Object, oDrawPage As Object
    Dim oGraphic As Object
    sGraphicURL = "file:///OOo/share/gallery/bullets/blkpearl.gif"
    sGraphicService = "com.sun.star.drawing.GraphicObjectShape"
    oDrawPage = vSheet.getDrawPage()
    oGraphic = ThisComponent.createInstance(sGraphicService)
    oGraphic.GraphicURL = sGraphicURL
    oDrawPage.add(oGraphic)

    REM Size the object
    Dim TheSize As New com.sun.star.awt.Size
    TheSize.width=400
    TheSize.height=400
    oGraphic.setsize(TheSize)

```

```

REM Position the object
Dim aPosition As New com.sun.star.awt.Point
aPosition.X = 2540
aPosition.Y = 2540
oGraphic.setposition(aPosition)
End Sub

```

5.30.2. Insert image into a text table cell

I have not attempted this, but according to Fernand Vanrie, it is difficult to find the dimensions of a cell when cells are merged. He provides this code that will insert a graphic in Tablecells using the cell dimensions to dimensioning the graphics.

I expect that this has errors, but I will not try it now.

```

oText = oDocument.text
RasterofCursor = True
oViewCursor = oDocument.getCurrentController().getViewCursor()
If Not isEmpty(oViewCursor.TextTable) Then ' Cursor is in een Tabel
    ' sFrameofTabel = "TABEL"
    ' inFrameofTabel = true
    ' RasterofCursor = False
    Twidth = oViewCursor.TextTable.width
    TTtotalpercent = oViewCursor.TextTable.TableColumnRelativeSum
    startcell = oViewCursor.cell.cellname
    Kindex = asc(left(oViewCursor.cell.cellname,1))-65
    Rindex = int(mid(oViewCursor.cell.cellname,2,2))-1
    CurRow = oViewCursor.texttable.rows.getbyindex(Rindex)
    If CurRow.IsAutoHeight = True Then
        MsgBox "Row height is not fixed..."
        Exit Sub
    End If
    If oViewCursor.cell.compareRegionStarts( oViewCursor.cell.getstart, _
        oViewCursor.cell.getend) <> 0 or oViewCursor.cell.string <>"" Then
        MsgBox "er mag in de cell enkel één" & chr(13) & _
            "RETURN staan" & chr(13) & "Eerst aanpassen, dan pas kan je verder"
        Exit Sub
    EndIf
    If oViewCursor.paraStyleName <> "Standard" Then
        MsgBox "de ParagraphStyle in de Tabelcell staat niet op default" & _
            chr(13) & "wordt nu automatisch aangepast !!"
        oViewCursor.ParaStyleName = "Standard"
    End If
    If oViewCursor.CharStyleName <> "" Then
        MsgBox "de CharacterStyle in de Tabelcell staat niet op default" & _
            chr(13) & "wordt nu automatisch aangepast !!"
        oViewCursor.setPropertyToDefault("CharStyleName")
    End If
End If

```

```

End If
If oViewCursor.texttable.Horiorient = 6 Then
    MsgBox " De Table mag niet AUTOMATISCH gealigneerd zijn" & _
        chr(13) & "wordt nu automatisch aangepast naar ""center"" !!"
    oViewCursor.texttable.Horiorient = 2
End If

' welke cell ??
startcellname = oViewCursor.cell.cellname
startrow = int(mid(startcellname,2,2))
If oViewCursor.cell.VertOrient <> 2 Then
    MsgBox "Cursor staat niet in de HOOGTE gecentreerd" & chr(13) & _
        "wordt nu automatisch aangepast !!"
    oViewCursor.cell.VertOrient = 2
End If
If oViewCursor.cell.createEnumeration.nextelement.paraAdjust <> 3 Then
    MsgBox "Tabelcell-Cursor is Niet in de BREEDTE gecentreerd" & _
        chr(13) & "wordt nu automatisch aangepast !!"
    oViewCursor.cell.createEnumeration.nextelement.paraAdjust = 3
End If
oViewcursor.goDOWN(1,false)
If isEmpty(oViewCursor.TextTable) Then
    startcellpos = "ONDER"
    oViewcursor.goUP(1,false)
End If

If startcellpos = "ONDER" Then
    'eerst terug naar startcell
    otext = oviewcursor.texttable.getcellbyname(startcellname,true).text
    otabelCursor = oText.createTextCursor()
    oViewcursor.gotorange(otabelCursor,false)
    If otext.supportsService("com.sun.star.text.Paragraph") then
        print "is paragraaf"
        exit sub
    End If

    laatsterow = oViewCursor.TextTable.rows.count
    If laatsterow = int(mid(oViewCursor.cell.cellname,2,2)) then
        iFotH = oViewCursor.texttable.rows.getbyindex(Rindex).height
    Else
        For i= 0 to laatsterow - int(mid(oViewCursor.cell.cellname,2,2))
            iFotH = iFotH + oViewCursor.texttable.rows.getbyindex(Rindex+ i ).height
        Next i
    End If
Else 'cell boven of midden
    ' eerst terug naar de startcell
    otext = oviewcursor.texttable.getcellbyname(startcellname,true).text

```

```

otabelCursor = oText.createTextCursor()
oViewcursor.gotorange(otabelCursor,false)
startrow = int(mid(oViewCursor.cell.cellname,2,2))
oViewcursor.goDOWN(1,false)
volgendorow = int(mid(oViewCursor.cell.cellname,2,2))
If isEmpty(oViewCursor.TextTable) Then 'terug naar binnen de tabel
    oViewcursor.goUP(1,false)
End If
' terug naar de startcell
otext = oviewcursor.texttable.getcellbyname(startcellname,true).text
otabelCursor = oText.createTextCursor()
oViewcursor.gotorange(otabelCursor,false)
For i= 1 To volgendorow - startrow
    iFotH = iFotH + _
    oViewCursor.texttable.rows.getbyindex(Rindex + i-1 ).height
Next i
EndIf

' iFotH = CurRow.height

tabstops = CurRow.TableColumnSeparators()
If uBound(tabstops)< 0 Then
    iFotW = tWidth
Else
    If kindex = 0 Then
        iFotW = (twidth * tabstops(Kindex).position/TTotalpercent)
    elseif kindex -1 = ubound(tabstops) then
        iFotW = twidth - (twidth * tabstops(Kindex-1).position/TTotalpercent)
    Else
        iFotW = (twidth * tabstops(Kindex).position/TTotalpercent) - _
            (twidth * tabstops(Kindex-1).position/TTotalpercent)
    EndIf
End If

otext = oViewCursor.text

lAnchor = com.sun.star.text.TextContentAnchorType.AS_CHARACTER 'AT_PARAGRAPH
oShape = oDocument.createInstance("com.sun.star.drawing.GraphicObjectShape")
oGraphic = oDocument.createInstance("com.sun.star.text.GraphicObject")
oDocument.getDrawPage().add(oShape)
oOriginalGraph = getGraphFromUrl(sGraphicURL)
oShape.Graphic = oOriginalGraph
oGraphic.GraphicUrl = oShape.GraphicUrl
oGraphic.AnchorType = lAnchor

oCell = oViewCursor.cell
oText = oCell.getText()

```

```

iFOTh = iFOTH - oCell.BottomBorderDistance - oCell.TopBorderDistance
iFOTw = iFOTw - oCell.LeftBorderDistance - oCell.RightBorderDistance
oCursor = oText.createTextCursor()
oText.insertTextContent( oCursor, oGraphic, False )

' We no longer require the shape object.
oDocument.getDrawPage().remove(oShape)
oShape = nothing

oGraphic.HoriOrientPosition = 0
oGraphic.VertOrientPosition = 0
oGraphic.SetPropertyValue("GraphicCrop" , aCrop)
oGraphic.SetPropertyValue("HyperLinkName" , sGraphicURL)
oGraphic.PositionProtected = False
oGraphic.SizeProtected = False
oGraphic.ContentProtected = True
oGraphic.width = iFotW
oGraphic.height = iFotH
oGraphic.HoriOrient = NONE
oGraphic.VertOrient = NONE

```

5.30.3. Export an image at a specified size

This from Sven Jacobi [Sven.Jacobi@sun.com]

Although it is not documented in the Developer's Guide, as of OOo 1.1, it is possible to export an image at a specified resolution. The MediaDescriptor in each graphic filter supports the “FilterData” property sequence, which sets the image size in pixels using the properties PixelWidth and PixelHeight. The logical size can be set in units of 1/100 mm using the properties LogicalWidth and LogicalHeight.

[Andy adds] This uses the GraphicExportFilter, which is only able to export a shape, shapes, or a draw page. The macro shown below, obtains the object to export as the selected object. In a Writer document, for example, a selected inserted graphic is not a shape; it is a TextGraphicObject.

Listing 5.85: *Export current page as a graphic at a specified size.*

```

Sub ExportCurrentPageOrSelection
    REM Filter dependent filter properties
    Dim aFilterData (4) As New com.sun.star.beans.PropertyValue
    Dim sFileUrl As String

    aFilterData(0).Name = "PixelWidth"
    aFilterData(0).Value = 1000
    aFilterData(1).Name = "PixelHeight"
    aFilterData(1).Value = 1000

```

```

aFilterData(2).Name = "LogicalWidth"
aFilterData(2).Value = 1000
aFilterData(3).Name = "LogicalHeight"
aFilterData(3).Value = 1000
aFilterData(4).Name = "Quality"
aFilterData(4).Value = 60

sFileUrl = "file:///d:/test2.jpg"

REM A smart person would force this to be a Draw or Impress document
xDoc = ThisComponent
xView = xDoc.currentController
xSelection = xView.selection
If isEmpty( xSelection ) Then
    xObj = xView.currentPage
Else
    xObj = xSelection
End If
Export( xObj, sFileUrl, aFilterData() )
End Sub

Sub Export( xObject, sFileUrl As String, aFilterData )
    Dim xExporter

    xExporter = createUnoService( "com.sun.star.drawing.GraphicExportFilter" )
    xExporter.SetSourceDocument( xObject )

    Dim aArgs (2) As New com.sun.star.beans.PropertyValue
    Dim aURL As New com.sun.star.util.URL

    aURL.complete = sFileUrl
    aArgs(0).Name = "MediaType"
    aArgs(0).Value = "image/jpeg"
    aArgs(1).Name = "URL"
    aArgs(1).Value = aURL
    aArgs(2).Name = "FilterData"
    aArgs(2).Value = aFilterData
    xExporter.filter( aArgs() )
End Sub

```

5.30.4. Draw a Line in a Calc Document

David Woody [dwoody1@airmail.net] provides the following:

Be aware that TheSize variables are relative to the aPosition variable so that if you want x1 = 500 and x2 = 2000 then TheSize.width = x2 - x1. Similarly for the Y coordinate.

Listing 5.86: Draw a line in a Calc document.

```
Sub DrawLineInCalcDocument
    Dim xPage as object, xDoc as object, xShape as object
    Dim aPosition As New com.sun.star.awt.Point
    Dim TheSize As New com.sun.star.awt.Size

    xDoc = thiscomponent
    xPage = xDoc.DrawPages(0)
    xShape = xDoc.CreateInstance( "com.sun.star.drawing.LineShape" )
    xShape.LineColor = rgb( 255, 0, 0 )
    xShape.LineWidth = 100
    aPosition.X = 2500
    aPosition.Y = 2500
    xShape.setPosition(aPosition)
    TheSize.width = 2500
    TheSize.height=5000
    xShape.setSize(TheSize)
    xPage.add( xShape )
End Sub
```

5.31. Extracting a Zip File

Laurent Godard [listes.godard@laposte.net] strikes again with this solution. I modified his post.

Hi all,

Thank you very much for your Help! Combining the different advices you all gave, I finally managed to make it work ! The point is to handle the content of the input stream as OOO's API does : don't care what it is !

To solve my problem I set an OutputStream and write my InputStream in it, That's all. And it seems to work (tested on a text file, but should work otherwise ...). So as promised, here is a first shot of my macro to UNZIP a known file in a ZIP package. There remains a lot to do but it can perhaps help Andrew, you can use this in your macro Doc.

Thanks again for all you help

Laurent Godard.

Listing 5.87: Unzip a file.

```
Sub UnzipAFile(ZipURL as string, SrcFileName as string, DestFile as string)
    Dim bExists as boolean

    ozip=createUnoService("com.sun.star.packages.Package")

    Dim oProp(0)
    oProp(0)=ConvertToURL(ZipURL)
```



```

oZip.Initialize(oProp())

'does srcFile exists ?
bExists=oZip.HasByHierarchicalName(SrcFileName)
if not bExists then exit sub

'retreive a PackageStream
ThePackageStream=oZip.GetByHierarchicalName(SrcFileName)

'Retreive the InputStream on SrcFileName
MyInputStream=ThePackageStream.GetInputStream()

'Define the output
oFile = createUnoService("com.sun.star.ucb.SimpleFileAccess")
oFile.WriteFile(ConvertToURL(DestFile), MyInputStream)
End Sub

```

5.31.1. Another Zip File Example

Dan Juliano <daniel.juliano@rainhail.com> <djuliano@dmacc.edu> expands on the example by Laurent Godard. The following example extracts all of the files from a zip file.

Listing 5.88: *Extract all files in a zip file.*

```

' Test usage for the following subs
call unzipFileFromArchive("c:\test.zip", "test.txt", "c:\test.txt")
call unzipArchive("c:\test.zip", "c:\")

Sub unzipFileFromArchive( _
    strZipArchivePath As String, _
    strSourceFileName As String, _
    strDestinationFilePath As String)

    Dim blnExists           As Boolean
    Dim args(0)             As Variant
    Dim objZipService        As Variant
    Dim objPackageStream     As Variant
    Dim objOutputStream      As Variant
    Dim objInputStream       As Variant
    Dim i                   As Integer

    '=====
    ' Unzip a single file from an archive. You must know the exact name
    ' of the file inside the archive before this sub can dig it out.
    '
    ' strZipArchivePath = full path (directory and filename)
    ' to the .zip archive file.
    ' strSourceFileName = the name of the file being dug from the .zip archive.
    ' strDestinationFilePath = full path (directory and filename) where

```

```

' the source file will be dumped.
'=====

' Create a handle to the zip service,
objZipService = createUnoService("com.sun.star.packages.Package")
args(0) = ConvertToURL(strZipArchivePath)
objZipService.initialize(args())

' Does the source file exist?
If Not objZipService.HasByHierarchicalName(strSourceFileName) Then Exit Sub

' Get the file input stream from the archive package stream.
objPackageStream = objZipService.GetByHierarchicalName(strSourceFileName)
objInputStream = objPackageStream.GetInputStream()

' Define the output.
objOutputStream = createUnoService("com.sun.star.ucb.SimpleFileAccess")
objOutputStream.WriteFile(ConvertToURL(strDestinationFilePath), _
                           objInputStream)
End Sub

Sub unzipArchive( _
    strZipArchivePath As String, _
    strDestinationFolder As String)

    Dim args(0)           As Variant
    Dim objZipService      As Variant
    Dim objPackageStream   As Variant
    Dim objOutputStream    As Variant
    Dim objInputStream     As Variant
    Dim arrayNames()      As Variant
    Dim strNames           As String
    Dim i                  As Integer

    '=====
    ' Unzip the an entire .zip archive to a destination directory.
    '
    ' strZipArchivePath = full path to the .zip archive file.
    ' strDestinationFolder = folder where the source files will be dumped.
    '=====

    ' Create a handle to the zip service,
    objZipService = createUnoService("com.sun.star.packages.Package")
    args(0) = ConvertToURL(strZipArchivePath)
    objZipService.initialize(args())

    ' Grab a package stream containing the entire archive.

```

```

objPackageStream = objZipService.GetByHierarchicalName("")

' Grab a listing of all files in the archive.
arrayNames = objPackageStream.getElementNames()

' Run through each file in the name array and pipe from archive
' to destination folder.
For i = LBound(arrayNames) To UBound(arrayNames)
    strNames = strNames & arrayNames(i) & Chr(13)

    ' Read in and pump out one file at a time to the filesystem.
    ObjInputStream = _
        objZipService.GetByHierarchicalName(arrayNames(i)).GetInputStream()
    objOutputStream = createUnoService("com.sun.star.ucb.SimpleFileAccess")
    objOutputStream.WriteFile(ConvertToURL(strDestinationFolder & _
        arrayNames(i)), objInputStream)
Next
MsgBox strNames
End Sub

```

5.31.2. Zip Entire Directories

Laurent Godard provides this example as well. This macro zips the content of a directory respecting subdirectories

Listing 5.89: Create a zip file.

```

'-----
sub ExempleAppel
    call ZipDirectory("C:\MesFichiers\Ooo\Rep", "C:\resultat.zip")
end sub

REM The paths should NOT be URLs.
REM Warning, the created ZIP file contains two extra artifacts.
REM (1) A Meta-Inf direction, which contains a manifest file.
REM (2) A mime-type file of zero length.
sub ZipDirectory(sSrcDir As String, sZipName As String)
    'Author: Laurent Godard - listes.godard@laposte.net
    'Modified: Andrew Pitonyak

    Dim sDirs() As String
    Dim oUcb          ' com.sun.star.ucb.SimpleFileAccess
    Dim oZip          ' com.sun.star.packages.Package
    Dim azipper
    Dim args(0)       ' Initialize zip package to zip file name.
    Dim argsDir(0)    ' Set to true to include directories in the zip
    Dim sBaseDir$
    Dim i%
    Dim chaine$

```

```

Dim decoupe      ' Each directory component in an array.
Dim repZip
Dim RepPere
Dim RepPereZip
Dim sFileName$
Dim oFile        ' File stream

'Create the package!
oZip=createUnoService("com.sun.star.packages.Package")
args(0)=ConvertToURL(sZipName)
oZip.initialize(args())

'création de la structure des repertoires dans le zip
call Recursedirectory(sSrcDir, sDirs())

argsDir(0)=true

'on saute le premier --> repertoire contenant
'Pourra etre une option a terme
sBaseDir=sDirs(1)

For i=2 To UBound(sDirs)
    chaine=mid(sDirs(i),len(sBaseDir)+2)
    decoupe=split(mid(sDirs(i),len(sBaseDir)+1),getPathSeparator())
    repZip=decoupe(UBound(decoupe))
    azipper=oZip.CreateInstanceWithArguments(argsDir())

    If len(chaine)<>len(repZip) then
        RepPere=left(chaine,len(chaine)-len(repZip)-1)
        RepPere=RemplaceChaine(reppere, getpathseparator, "/", false)
    Else
        RepPere=""
    Endif

    RepPereZip=oZip.getByHierarchicalName(RepPere)
    RepPereZip.insertbyname(repzip, azipper)
Next i

'insertion des fichiers dans les bons repertoires
dim args2(0)
args2(0)=false
oUcb = createUnoService("com.sun.star.ucb.SimpleFileAccess")

for i=1 to UBound(sDirs)
    chaine=mid(sDirs(i),len(sBaseDir)+2)
    repzip=remplacechaine(chaine, getpathseparator, "/", false)
    sFileName=dir(sDirs(i)+getPathSeparator(), 0)

```

```

While sFileName<>""
    azipper=oZip.createInstanceWithArguments(args2())
    oFile = oUcb.OpenFileRead(ConvertToURL(sDirs(i)+"/"+sFileName))
    azipper.SetInputStream(oFile)
    RepPere=oZip.getByHierarchicalName(repZip)
    RepPere.insertbyname(sFileName, azipper)
    sFileName=dir()
Wend
next i

'Valide les changements
oZip.commitChanges()
MsgBox "Finished"
End Sub

REM Read the directory names
Sub RecurseDirectory(sRootDir$, sDirs As Variant)
    'Author: laurent Godard - listes.godard@laposte.net
    'Modified: Andrew Pitonyak
    Redim Preserve sDirs(1 to 1)

    Dim nNumDirs%      ' Track the number of directories or files
    Dim nCurIndex%     ' Current index into the directories or files
    Dim sCurDir$       ' Current directory.

    nNumDirs=1
    sDirs(1)=sRootDir
    nCurIndex=1
    sCurDir = dir(ConvertToUrl(sRootDir & "/"), 16)

    Do While sCurDir <> ""
        If sCurDir <> "." AND sCurDir<> ".." Then
            nNumDirs=nNumDirs+1
            ReDim Preserve sDirs(1 to nNumDirs)
            sDirs(nNumDirs)=convertfromurl(sDirs(nCurIndex)+"/"+sCurDir)
        endif
        sCurDir=dir()

        Do While sCurDir = "" AND nCurIndex < nNumDirs
            nCurIndex = nCurIndex+1
            sCurDir=dir(convertToURL(sDirs(nCurIndex)+"/"),16)
        Loop
    Loop
End Sub

Function RemplaceChaine(ByVal sSearchThis$, sFindThis$, dest$, bCase As Boolean)
    'Auteurs: Laurent Godard & Bernard Marcelly

```

```

' fournit une sSearchThis dont toutes les occurences de sFindThis ont
' été remplacées par dest
' bCase = true pour distinguer majuscules/minuscules, = false sinon
Dim nSrcLen As Integer
Dim i%      ' Current index.
Dim nUseCase% ' InStr Argument, determines if case sensitive.
Dim sNewString As String

sNewString=""
nUseCase = IIF(bCase, 0, 1)
nSrcLen = len(sFindThis)
i = instr(1, sSearchThis, sFindThis, nUseCase)

REM While nSearchThis contains sFindThis
Do While i<>0
    REM If the location is past 32K, remoe the first 32000 characters.
    REM This is done to prevent negative values.
    Do While i<0
        sNewString = sNewString & Left(sSearchThis,32000)
        sSearchThis = Mid(sSearchThis,32001)
        i=InStr(1, sSearchThis, sFindThis, nUseCase)
    Loop

    If i>1 Then
        sNewString = sNewString & Left(sSearchThis, i-1) & dest
    else
        sNewString = sNewString & dest
    endif
    ' raccourcir en deux temps car risque : i+src > 32767
    sSearchThis = Mid(sSearchThis, i)
    sSearchThis = Mid(sSearchThis, 1+nSrcLen)
    i = instr(1, sSearchThis, sFindThis, nUseCase)
Loop
ReplaceChaine = sNewString & sSearchThis
End Function

```

5.32. Run a macro by string name

A given macro subroutine or function name can be called using the dispatch API. This is useful when the precise routine to call is not definable when the macro is initially written. Consider, for example, a list of routines to call that is held in an external file. Thanks to Paolo Mantovani for the following solution:

Listing 5.90: *Run a macro based on the value in a string.*

```

Sub RunGlobalNamedMacro
    oDisp = createUnoService("com.sun.star.frame.DispatchHelper")
    sMacroURL = "macro:///Gimmicks.AutoText.Main"

```

```

oDisp.executeDispatch(StarDesktop, sMacroURL, "", 0, Array())
End Sub

```

Notice that the desktop is used as the object that handles the dispatch.

5.32.1. Run a macro from the command line

Run a macro from the command line by specifying the name:

```
soffice.exe macro:///standard.module1.macro1
```

In this example, “standard” is the library name, “module1” is the module name, and “macro1” is the name of the macro.

Tip

If the macro makes or opens nothing within a document, the macro is implemented and closed StarOffice again.

5.32.2. Run a named macro in a document

All of the examples to this point run macros contained in the global object container. It is possible to run a macro that is contained in a document.

Listing 5.91: *Run a macro in a document based on the value in a string.*

```

Sub RunDocumentNamedMacro
    Dim oDisp
    Dim sMacroURL As String
    Dim sMacroName As String
    Dim sMacroLocation As String
    Dim oFrame

    oDisp = createUnoService("com.sun.star.frame.DispatchHelper")

    REM To figure out the URL, add a button and then set the button
    REM to call a macro.
    sMacroName = "vnd.sun.star.script:Standard.Module1.MainExternal"
    sMacroLocation = "?language=Basic&location=document"
    sMacroURL = sMacroName & sMacroLocation

    REM I want to call a macro contained in ThisComponent, so I
    REM must use the frame from the document containing the macro
    REM as the dispatch recipient.
    oFrame = ThisComponent.CurrentController.Frame
    oDisp.executeDispatch(oFrame, sMacroURL, "", 0, Array())
    'oDisp.executeDispatch(StarDesktop, sMacroURL, "", 0, Array())
End Sub

```

But wait, it can be even easier... I stored this in the document “delme.odt” so I can simply use the following URL, even if I use StarDesktop as the dispatch receiver:

```
sMacroURL = "macro:///delme/Standard.Module1.MainExternal"
```

5.33. Using a “default application” to open a file

When I use a Windows computer, the first thing that I do is to install 4NT from JP Software (<http://www.jpsoft.com>) because I use the command line. When I want to open a PDF file, I simply type the name of the file and press enter. Windows looks at the file extension and then automatically opens a PDF reader. The GUI equivalent is to double click on a file and the is opened in the correct software.

You can accomplish the same thing using OOo by using the SystemShellExecute service. (Thanks to Russ Phillips [avantman42@users.sourceforge.net] for pointing me to Erik Anderson's findings <http://www.ooforum.org/forum/viewtopic.php?t=6657>)

The magic is performed by the SystemShellExecute service, which contains one method; execute!

Listing 5.92: *Open a file based on the default application.*

```
Sub LaunchOutsideFile()  
    Dim oSvc as object  
    oSvc = createUnoService("com.sun.star.system.SystemShellExecute")  
  
    Rem File:  
    ' oSvc.execute(ConvertToUrl("C:\sample.txt"), "", 0)  
    Rem Folder:  
    ' oSvc.execute(ConvertToUrl("C:\Program Files\OpenOffice.org1.1.0"), "", 0)  
    Rem Web address:  
    ' oSvc.execute("http://www.openoffice.org/", "", 0)  
    Rem Email:  
    ' oSvc.execute("mailto:anonymous@ftp.com", "", 0)  
End Sub
```

5.34. Listing Fonts

The available fonts are known by the container window. The getFontDescriptors() method returns an array of AWT FontDescriptor structures that contain a lot of information about the font. The font descriptor can be passed to the getFont() method, which returns an object that supports the AWT XFont interface. The XFont interface provides methods to determine font metrics, and the width of an individual character or an entire string of characters.

Listing 5.93: *List fonts.*

```
Sub ListFonts  
    Dim oWindow      'The container window supports the awt XDevice interface.  
    Dim oDescriptor  'Array of awt FontDescriptor structures  
    Dim s$           'Temporary string variable to hold all of the string names.  
    Dim i%           'General index variable  
  
    oWindow = ThisComponent.getCurrentController().getFrame().getContainerWindow()  
    oDescriptor = oWindow.getFontDescriptors()
```



```

s = ""

For i = LBound(oDescript) to UBound(oDescript)
    s = s & oDescript(i).Name & ", "
Next
MsgBox s
End Sub

```

5.35. Get the document URL, filename, and directory

Do not try to obtain the document URL unless it has a URL. If the document has not yet been stored, for example. Rather than write my own routines, I use some functions in the Strings Module stored in the Tools library.

Listing 5.94: Extracting file and path information from a URL.

```

REM Author: Andrew Pitonyak
Sub DocumentFileNames
    Dim oDoc
    Dim sDocURL
    oDoc = ThisComponent
    If (Not GlobalScope.BasicLibraries.isLibraryLoaded("Tools")) Then
        GlobalScope.BasicLibraries.LoadLibrary("Tools")
    End If
    If (oDoc.hasLocation()) Then
        sDocURL = oDoc.getURL()
        Print "Document Directory = " & DirectoryNameoutofPath(sDocURL, "/")
        Print "Document File Name = " & FileNameoutofPath(sDocURL, "/")
    End If
End Sub

```

5.36. Get and set the current directory

In OOo, ChDir and ChDrive currently do nothing – this is intentional. The following is based on a discussion between Andreas Bregas, Christian Junker, Paolo Mantovani and Andrew Pitonyak.

Initially, the ChDir and ChDrive statements made file system calls, but they were rewritten using the UCB layer – as was all file system related functionality. This is why all of the commands now also accept URL notation. Support for a current working directory is not supported by the UCB and the underlying sal/osl API because of the inherent problems (bugs) in a multi threaded environment. In Windows, for example, the File Open dialog changes the process' current working directory, as do other API calls. You expect the current working directory to be one thing, but then another thread changes it. Paolo recommends the use of the PathSettings service, which contains numerous path values (see Table 5.2).

Table 5.2. Properties supported by the *com.sun.star.util.PathSettings* service.

Property	Description
Backup	Automatic backup copies of documents are stored here.
Basic	The Basic files, used by the AutoPilots, can be found here. The value can be more than one path separated by a semicolon.
Favorite	Path to save folder bookmarks
Gallery	Location of the Gallery database and multimedia files. The value can be more than one path separated by a semicolon.
Graphic	This directory is displayed when the dialog for opening a graphic or for saving a new graphic is called.
Help	The path to the Office help files.
Module	This is the path for the modules.
Storage	Mail, News files and other information (for example, about FTP Server) are stored here.
Temp	The base URL to the office temp-files
Template	The templates originate from these folders and sub-folders. The value can be more than one path separated by a semicolon.
UserConfig	Folder that contains the user settings.
Work	User's work folder, which can be modified – used by the Open and Save dialogs.

The following code clarifies how this works:

Listing 5.95: Use the *PathSettings* service.

```

Author: Paolo Mantovani
Function pmxCurDir() As String
    Dim oPathSettings
    oPathSettings = CreateUnoService("com.sun.star.util.PathSettings")

    'The path of the work folder can be modified according to the user's needs.
    'The path specified here can be seen in the Open or Save dialog.
    pmxCurDir = oPathSettings.Work
End Function

Function pmxChDir(sNewDir As String) As String
    Dim oPathSettings
    oPathSettings = CreateUnoService("com.sun.star.util.PathSettings")
    oPathSettings.Work = ConvertToUrl(sNewDir)
    pmxChDir = oPathSettings.Work
End Function

```

There is also a *com.sun.star.util.PathSubstitution* service, which provides access to many interesting path related values. The path variables are not case sensitive and are always returned as a UCB-compliant URL, for example, “file:///c:/temp” or “file:///usr/install”. The

supported list of values are stored in the Office configuration file (org/openoffice/Office/Substitution.xml). The variables with predefined values are as follows:

Table 5.3. Variables recognized by the the com.sun.star.util.PathSubstitution service.

Variable	Description
\$(inst)	Installation path of the Office.
\$(prog)	Program path of the Office.
\$(user)	The user installation directory.
\$(work)	The user's work directory of the user. Under Windows this is the "MyDocuments" subdirectory. Under Unix this is the home-directory
\$(home)	The user's home directory of the user. Under Unix this is the home-directory. Under Windows this is the "Documents and Settings " subdirectory.
\$(temp)	The current temporary directory.
\$(path)	The value of PATH environment variable.
\$(lang)	The country code used by the Office, like 01=english, 49=german.
\$(langid)	The language code used by the Office, like 0x0009=english, 0x0409=english us.
\$(vlang)	The language used by the Office as a string. Like "german" for a German Office.

Listing 5.96: Use the PathSubstitution service.

```
oPathSubst = createUnoService("com.sun.star.util.PathSubstitution")
Print oPathSubst.getSubstituteVariableValue("${inst}")
```

5.37. Writing to a file

The methods provided directly by BASIC contains certain flaws and interesting behavior. I take an entire chapter to discuss this in my book. I recently found the following little snippet by Christian Junker that I will need to explore sometime. You can set the text encoding use the setEncoding() method.

Listing 5.97: SimpleFileAccess allows you to set the output format.

```
fileAccessService = createUnoService("com.sun.star.ucb.SimpleFileAccess")
textOutputStream = createUnoService("com.sun.star.io.TextOutputStream")
'now open the file..
outputStream = fileAccessService.openFileWrite(<yourFileName>)
outputStream.truncate()
textOutputStream.setOutputStream(outputStream)
'now write something into the file
textOutputStream.writeString("This is utf-8 format.")
'and don't forget to close it..
textOutputStream.closeOutput()
```

5.38. Parsing XML

A wonderful example of parsing XML is provided by DannyB on the oooforum (see <http://www.oooforum.org/forum/viewtopic.php?t=4907>). You should read this before using the following macro:

Listing 5.98: Parsing XML.

```
Sub Main
    cXmlFile = "C:\TestData.xml"
    cXmlUrl = ConvertToURL( cXmlFile )
    ReadXmlFromUrl( cXmlUrl )
End Sub

' This routine demonstrates how to use the Universal Content Broker's
' SimpleFileAccess to read from a local file.
Sub ReadXmlFromUrl( cUrl )
    ' The SimpleFileAccess service provides mechanisms to
    ' open, read, write files,
    ' as well as scan the directories of folders to see what they contain.
    ' The advantage of this over Basic's ugly file manipulation is that this
    ' technique works the same way in any programming language.
    ' Furthermore, the program could be running on one machine,
    ' while the SimpleFileAccess
    ' accesses files from the point of view of the machine running OOo,
    ' not the machine
    ' where, say a remote Java or Python program is running.
    oSFA = createUnoService( "com.sun.star.ucb.SimpleFileAccess" )

    ' Open input file.
    oInputStream = oSFA.openFileRead( cUrl )
    ReadXmlFromInputStream( oInputStream )
    oInputStream.closeInput()
End Sub

Sub ReadXmlFromInputStream( oInputStream )
    ' Create a Sax Xml parser.
    oSaxParser = createUnoService( "com.sun.star.xml.sax.Parser" )

    ' Create a document event handler object.
    ' As methods of this object are called, Basic arranges
    ' for global routines (see below) to be called.
    oDocEventsHandler = CreateDocumentHandler()

    ' Plug our event handler into the parser.
    ' As the parser reads an Xml document, it calls methods
    ' of the object, and hence global subroutines below
    ' to notify them of what it is seeing within the Xml document.
```

```

oSaxParser.setDocumentHandler( oDocEventsHandler )

' Create an InputSource structure.
oInputSource = createUnoStruct( "com.sun.star.xml.sax.InputSource" )
With oInputSource
    .aInputStream = oInputStream ' plug in the input stream
End With

' Now parse the document.
' This reads in the entire document.
' Methods of the oDocEventsHandler object are called as
' the document is scanned.
oSaxParser.parseStream( oInputSource )
End Sub

'=====
'   Xml Sax document handler.
'=====

' Global variables used by our document handler.
'
' Once the Sax parser has given us a document locator,
' the glLocatorSet variable is set to True,
' and the goLocator contains the locator object.
'
' The methods of the locator object has cool methods
' which can tell you where within the current Xml document
' being parsed that the current Sax event occurred.
' The locator object implements com.sun.star.xml.sax.XLocator.
'
Private goLocator As Object
Private glLocatorSet As Boolean

' This creates an object which implements the interface
' com.sun.star.xml.sax.XDocumentHandler.
' The document handler is returned as the function result.
Function CreateDocumentHandler()
    ' Use the CreateUnoListener function of Basic.
    ' Basic creates and returns an object that implements a
    ' particular interface.
    ' When methods of that object are called,
    ' Basic will call global Basic functions whose names are the same
    ' as the methods, but prefixed with a certain prefix.
    oDocHandler = CreateUnoListener( "DocHandler_", _
                                     "com.sun.star.xml.sax.XDocumentHandler" )

    glLocatorSet = False

```

```

    CreateDocumentHandler() = oDocHandler
End Function

'=====
'   Methods of our document handler call these
'   global functions.
'   These methods look strangely similar to
'   a SAX event handler. ;-)
'   These global routines are called by the Sax parser
'   as it reads in an XML document.
'   These subroutines must be named with a prefix that is
'   followed by the event name of the
'   com.sun.star.xml.sax.XDocumentHandler interface.
'=====

Sub DocHandler_startDocument()
    Print "Start document"
End Sub

Sub DocHandler_endDocument()
    Print "End document"
End Sub

Sub DocHandler_startElement( cName$, _
                             oAttr As com.sun.star.xml.sax.XAttributeList )
    Print "Start element", cName
End Sub

Sub DocHandler_endElement( cName As String )
    Print "End element", cName
End Sub

Sub DocHandler_characters( cChars As String )
End Sub

Sub DocHandler_ignorableWhitespace( cWhitespace As String )
End Sub

Sub DocHandler_processingInstruction( cTarget As String, cData As String )
End Sub

Sub DocHandler_setDocumentLocator( oLocator As com.sun.star.xml.sax.XLocator )
    ' Save the locator object in a global variable.
    ' The locator object has valuable methods that we can
    ' call to determine
    goLocator = oLocator
    glLocatorSet = True

```

End Sub

DannyB recommends starting with a small file for your initial tests:

```
<Employees>
  <Employee id="101">
    <Name>
      <First>John</First>
      <Last>Smith</Last>
    </Name>
    <Address>
      <Street>123 Main</Street>
      <City>Lawrence</City>
      <State>KS</State>
      <Zip>66049</Zip>
    </Address>
    <Phone type="Home">785-555-1234</Phone>
  </Employee>
  <Employee id="102">
    <Name>
      <First>Bob</First>
      <Last>Jones</Last>
    </Name>
    <Address>
      <Street>456 Puke Drive</Street>
      <City>Lawrence</City>
      <State>KS</State>
      <Zip>66049</Zip>
    </Address>
    <Phone type="Home">785-555-1235</Phone>
  </Employee>
</Employees>
```

5.39. Manipulating Dates

My book contains complete coverage of dates, along with all of their idiosyncrasies. Remember that the fractional portion represents the time and the decimal portion represents the days. You can, therefore, simply add in the number of days to a date object to increment the current day.

Listing 5.99: *Adding two days together is easy.*

```
Function addDays(StartDate as Date, nDays As Integer) As Date
  REM To add days, simply add them in
  addDay = StartDate + nDays
End Function
```

Although adding days to a date is easy, there are complications when adding years or months. February had 29 days in 2004 and 28 days in 2005. You can not, therefore, simply add one to the year and be safe; similar problems exist for the month. You must decide what it means to

add one to the year or month. The initial routine was provided by Eric VanBuggenhaut, did not properly handle these situations. Antoine Jarrige noticed incorrect behavior and provided a solution, but problems still remained while adding 12 months to a date in December.

I did an almost complete rewrite using tricks presented in my Macro book. The final code first adds years and months. When adding years and months, an initial date that starts as the last day of the month, stays on the last day of the month. When this is completed, the days are added. If this is not what you desire, then change the macro.

The SumDate function adds the specified number of years, months, and days to a date variable. The primary disadvantages to this routine is that it drops the time component and does not properly handle dates with a year value below 100.

Listing 5.100: Add years, months, and days to a date.

```
Function SumDate(StartDate As Date, nYears%, nMonths%, nDays%) As Date
    REM Author: Eric Van Buggenhaut [Eric.VanBuggenhaut@AdValvas.be]
    REM Modified By:
    REM      Antoine Jarrige [pierre-antoine.jarrige@laposte.net]
    REM Almost complete rewrite by Andrew Pitonyak

    Dim lDateValue As Long      ' The start date is as a long integer.
    Dim nDateDay   As Integer   ' The day for the start date.
    Dim nDateMonth As Integer   ' The month.
    Dim nDateYear  As Integer   ' The year.
    Dim nLastDay_1 As Integer   ' Last day of the month for initial date.
    Dim nLastDay_2 As Integer   ' Last day of the month for target date.

    REM Determine the year, month, and day.
    nDateDay   = Day(StartDate)
    nDateMonth = Month(StartDate)
    nDateYear  = Year(StartDate)

    REM Find the last day of the month.
    If nDateMonth = 12 Then
        REM December always has 31 days
        nLastDay_1 = 31
    Else
        nLastDay_1 = Day(DateSerial(nDateYear, nDateMonth+1, 1)-1)
    End If

    REM Adding a year is only a problem on February 29th of a leap year.
    nDateYear = nDateYear + nYears
    nDateMonth = nDateMonth + nMonths
    If nDateMonth > 12 Then
        nDateYear = nDateYear + (nDateMonth - 1) \ 12
        nDateMonth = (nDateMonth - 1) MOD 12 + 1
    End If
```



```

REM Find the last day of the month.
If nDateMonth = 12 Then
    REM December always has 31 days
    nLastDay_2 = 31
Else
    nLastDay_2 = Day(DateSerial(nDateYear, nDateMonth+1, 1)-1)
End If

REM Force the last day of the month to stay on the last day of
REM the month. Do not allow an overflow into the next month.
REM The concern is that adding one month to Jan 31 will end
REM up in March.
If nDateDay = nLastDay_1 OR nDateDay > nLastDay_2 Then
    nDateDay = nLastDay_2
End If

REM While adding days, however, all bets are off.
SumDate=CDate(DateSerial(nDateYear, nDateMonth, nDateDay)+nDays)
End Function

```

5.40. Is OpenOffice embedded into a web browser?

OpenOffice can open a document directly into your web browser. OpenOffice supports an undocumented (and internally used) property, `isPlugged`, which indicates if the desktop is plugged into a browser.

```
Stardesktop.isPlugged()
```

Although it works in OOO version 1.1.2, the rumor is that the `isPlugged` method will be removed by version 2.0.

5.41. Focus (bring to the front) a new document

To cause the document referenced by the variable `oDoc2` to become the focused document, use either of the two methods:

Listing 5.101: *Make the current window active.*

```

oDoc2.CurrentController.Frame.ContainerWindow.toFront()
oDoc2.CurrentController.Frame.Activate()

```

This will not change the value of `ThisComponent`.

5.42. What is the document type (based on URL)

Christian Junker noted that you can use “deep” type detection to determine a document's type. This means that the correct type is returned even if the file extension is not. In other words, it will detect a Calc document with a `.doc` extension. The returned string is the internal format name.

Listing 5.102: Determine a document's type.

```
Sub DetectDocType()  
    Dim oMediaDescr(30) As new com.sun.star.beans.PropertyValue  
    Dim s$ : s$ = "com.sun.star.document.TypeDetection"  
    Dim oTypeManager  
  
    oMediaDescr(0).Name = "URL"  
    oMediaDescr(0).Value = ThisComponent.getURL()  
  
    oTypeManager = createUnoService(s$)  
    REM Perform a deep type detection  
    REM not just based on filename extension.  
    MsgBox oTypeManager.queryTypeByDescriptor(oMediaDescr(), True)  
End Sub
```

5.43. Connect to a remote OOO server using Basic

You can connect to a remote OOO server using Basic.

Listing 5.103: Determine a document's type.

```
Sub connectToRemoteOffice()  
    REM Author: Christian Junker  
    REM Author: Modified by Andrew Pitonyak  
    Dim sURL$      ' Connection URL to the remote host.  
    Dim sHost$     ' IP address running the remote host.  
    Dim sPort$     ' Port used on the remote host.  
    Dim oRes       ' URL Resolver.  
    Dim oRemote    ' Remote manager for the remote server.  
    Dim oDesk      ' Desktop object from the remote server.  
    Dim oDoc       ' The opened document.  
  
    REM Set the host and port running the server. The host must  
    REM have started a server listening on the specified port:  
    REM If you do not specify "host=0", it will not accept  
    REM connections from the network. For example, I started  
    REM soffice.exe on a windows computer using the following arguments:  
    REM "-accept=socket,host=0,port=8100;urp;StarOffice.ServiceManager"  
  
    sHost = "192.168.0.5"  
    sPort = "8100"  
    sURL = "uno:socket,host=" & sHost & _  
           ",port=" & sPort & _  
           ";urp;StarOffice.ServiceManager"  
    oRes = createUNOService("com.sun.star.bridge.UnoUrlResolver")  
    oRemote = oRes.resolve(sURL)  
    oDesk = oRemote.createInstance("com.sun.star.frame.Desktop")
```

```

REM Specify the document to open!
sURL = "private:factory/swriter"
'sURL = "file:///home/andy/PostData.doc"
oDoc = oDesk.loadComponentFromURL(sURL, "_blank", 0, Array())
Print "The document is now open"
End Sub

```

Something else to consider: have you looked at oood.py? A simple daemon for OpenOffice.org. <http://udk.openoffice.org/python/oood/>

5.44. Toolbars

New section under construction...

Toolbars have names. Custom toolbars all start with “private:resource/toolbar/custom_”. The standard toolbar names are shown in Listing 5.104.

Listing 5.104: Standard toolbar names.

```

Sub PrintStandardToolBarNames()
    MsgBox Join(GetStandardToolBarNames(), CHR$(10))
End Sub

Function GetStandardToolBarNames()
    GetStandardToolBarnames = Array ( _
        "private:resource/toolbar/alignmentbar", _
        "private:resource/toolbar/arrowshapes", _
        "private:resource/toolbar/basicshapes", _
        "private:resource/toolbar/calloutshapes", _
        "private:resource/toolbar/colorbar", _
        "private:resource/toolbar/drawbar", _
        "private:resource/toolbar/drawobjectbar", _
        "private:resource/toolbar/extrusionobjectbar", _
        "private:resource/toolbar/fontworkobjectbar", _
        "private:resource/toolbar/fontworkshapetypes", _
        "private:resource/toolbar/formatobjectbar", _
        "private:resource/toolbar/formcontrols", _
        "private:resource/toolbar/formdesign", _
        "private:resource/toolbar/formsfilterbar", _
        "private:resource/toolbar/formsnavigationbar", _
        "private:resource/toolbar/formsobjectbar", _
        "private:resource/toolbar/formtextobjectbar", _
        "private:resource/toolbar/fullscreenbar", _
        "private:resource/toolbar/graphicobjectbar", _
        "private:resource/toolbar/insertbar", _
        "private:resource/toolbar/insertcellsbar", _
        "private:resource/toolbar/insertobjectbar", _
        "private:resource/toolbar/mediaobjectbar", _
        "private:resource/toolbar/moreformcontrols", _

```

```

        "private:resource/toolbar/previewbar", _
        "private:resource/toolbar/standardbar", _
        "private:resource/toolbar/starshapes", _
        "private:resource/toolbar/symbolshapes", _
        "private:resource/toolbar/textobjectbar", _
        "private:resource/toolbar/toolbar", _
        "private:resource/toolbar/viewerbar", _
        "private:resource/menubar/menubar" _
    )
End Function

```

Use a frame's LayoutManager to find the current toolbars. It may be a but the *Listing 5.105* displays toolbars, menus, and status bars.

Listing 5.105: *Display toolbars in the current document.*

```

Sub SeeComponentsElements()
    Dim oDoc, oFrame
    Dim oCfgManager
    Dim oToolInfo
    Dim x
    Dim s$
    Dim iToolType as Integer

    oDoc = ThisComponent
    REM This is the integer value three.
    iToolType = com.sun.star.ui.UIElementType.TOOLBAR

    oFrame = oDoc.getCurrentController().getFrame()
    oCfgManager = oDoc.getUIConfigurationManager()
    oToolInfo = oCfgManager.getUIElementsInfo( iToolType )
    For Each x in oFrame.LayoutManager.getElements()
        s = s & x.ResourceURL & CHR$(10)
    Next
    MsgBox s, 0, "Toolbars in Component"
End Sub

```

Use the layout manager to see if a specific toolbar is currently visible. The `isElementVisible` method checks all element types, not just toolbars.

Listing 5.106: *See if a specified toolbar is visible.*

```

Sub TestToolBarVisible
    Dim s$, sName$
    For Each sName In GetStandardToolBarnames()
        s = s & IsToolBarVisible(ThisComponent, sName) & _
            " : " & sName & CHR$(10)
    Next
    MsgBox s, 0, "Toolbar Is Visible"
End Sub

```

```

Function IsToolbarVisible(oDoc, sURL) As Boolean
    Dim oFrame
    Dim oLayout

    oFrame = oDoc.GetCurrentController().getFrame()
    oLayout = oFrame.LayoutManager
    IsToolbarVisible = oLayout.isElementVisible(sURL)
End Function

```

Use `hideElement` and `showElement` to hide or show a toolbar. Prior to version 2.0, you had to rely on dispatches to toggle the visibility. For example, the following dispatches were used: `".uno:MenuBarVisible"`, `".uno:ObjectBarVisible"`, `".uno:OptionBarVisible"`, `".uno:NavigationBarVisible"`, `".uno:StatusBarVisible"`, `".uno:ToolBarVisible"`, `".uno:MacroBarVisible"`, `".uno:FunctionBarVisible"`, and `".uno:InputLineVisible"`.

Listing 5.107: *Toggle a toolbar's visibility.*

```

Sub ToggleToolbarVisible(oDoc, sURL)
    Dim oLayout

    oLayout = oDoc.CurrentController.getFrame().LayoutManager
    If oLayout.isElementVisible(sURL) Then
        oLayout.hideElement(sURL)
    Else
        oLayout.showElement(sURL)
    End If
End Sub

```

5.44.1. Create a toolbar for a component type

It is possible to create a new toolbar without any coding using an add-on. Create the XML that defines the toolbar, and then install it. I have not pursued this, so I do not know how this is accomplished.

Use a document's configuration manager to create and store a toolbar for a specific document, rather than a specific component type.

To create a toolbar attached to a component type (Writer document, Calc document, Basic IDE, etc.), retrieve the module user interface configuration manager and change the module dependent toolbars. (Thanks Carsten Driesner for this information and basic examples).

5.44.1.1. My first toolbar

I will create a toolbar that calls a macro written by me that resides in the UI module contained in the PitonyakUtil library.

```

Sub TBTest
    Print "In TBTest"
End Sub

```

Each toolbar item is an array of property values.

Listing 5.108: Create a simple toolbar item.

```
Rem A com.sun.star.ui.ItemDescriptor is an array of property
Rem values. This example does not set all supported values,
Rem such as "Style", which uses values
Rem from com.sun.star.ui.ItemStyle. For menu items, the
Rem "ItemDescriptorContainer" is usually set as well.
Function CreateSimpleToolbarItem( sCommand$, sLabel ) as Variant
    Dim oItem(3) As New com.sun.star.beans.PropertyValue

    oItem(0).Name = "CommandURL"
    oItem(0).Value = sCommand
    oItem(1).Name = "Label"
    oItem(1).Value = sLabel

    REM Other supported types include SEPARATOR_LINE,
    rem SEPARATOR_SPACE, and SEPARATOR_LINEBREAK.
    oItem(2).Name = "Type"
    oItem(2).Value = com.sun.star.ui.ItemType.DEFAULT

    oItem(3).Name = "Visible"
    oItem(3).Value = True

    CreateSimpleToolbarItem = oItem()
End Function
```

Creating the toolbar is a simple matter.

Listing 5.109: Add a simple toolbar to the Basic IDE.

```
Sub CreateBasicIDEToolbar
    Dim sToolbarURL$ ' URL of the custom toolbar.
    Dim sCmdID$      ' Command for a single toolbar button.
    Dim sCmdLabel    ' Label for a single toolbar button.
    Dim sDocType$    ' Component type that will contain the toolbar.
    Dim sSupplier$   ' ModuleUIConfigurationManagerSupplier
    Dim oSupplier    ' ModuleUIConfigurationManagerSupplier
    Dim oModuleCfgMgr ' Module manager.
    Dim oTBSettings  ' Settings that comprise the toolbar.
    Dim oToolbarItem ' Single toolbar button.
    Dim nCount%

    REM Name of the custom toolbar; must start with "custom_".
    sToolbarURL = "private:resource/toolbar/custom_test"

    REM Retrieve the module configuration manager from the
    REM central module configuration manager supplier
```

```

sSupplier = "com.sun.star.ui.ModuleUIConfigurationManagerSupplier"
oSupplier = CreateUnoService(sSupplier)

REM Specify the document type associated with this toolbar.
REM sDocType = "com.sun.star.text.TextDocument"
sDocType = "com.sun.star.script.BasicIDE"

REM Retrieve the module configuration manager with module identifier
REM *** See com.sun.star.frame.ModuleManager for more information.
oModuleCfgMgr = oSupplier.getUIConfigurationManager( sDocType )

REM To remove a toolbar, you can use something like the following:
'If (oModuleCfgMgr.hasSettings(sToolbarURL)) Then
'  oModuleCfgMgr.removeSettings(sToolbarURL)
'  Exit Sub
'End If

REM Create a settings container to define the structure of the
REM custom toolbar.
oTBSettings = oModuleCfgMgr.createSettings()

REM *** Set a title for our new custom toolbar
oTBSettings.UIName = "My little custom toolbar"

REM *** Create a button for our new custom toolbar
sCmdID   = "macro:///PitonyakUtil.UI.TBTest()"
sCmdLable = "Test"
nCount = 0
oToolbarItem = CreateSimpleToolbarItem( sCmdID, sCmdLable )
oTBSettings.insertByIndex( nCount, oToolbarItem )

REM To add a second item, increment nCount, create a new
REM toolbar item, and insert it.

REM *** Set the settings for our new custom toolbar. (replace/insert)
If ( oModuleCfgMgr.hasSettings( sToolbarURL ) ) Then
  oModuleCfgMgr.replaceSettings( sToolbarURL, oTBSettings )
Else
  oModuleCfgMgr.insertSettings( sToolbarURL, oTBSettings )
End If
End Sub

```

Carsten Driesner provided a macro that adds a button to the standard toolbar in a Writer document.

Listing 5.110: Add a toolbar button to the standard Writer toolbar.

```
REM *** This example creates a new basic macro toolbar button on
REM *** the Writer standard bar. It doesn't add the button twice.
REM *** It uses the Writer image manager to set an external image
REM *** for the macro toolbar button.

Sub AddButtonToToolbar
    Dim sToolbar$ : sToolbar = "private:resource/toolbar/standardbar"
    Dim sCmdID$    : sCmdID  = "macro:///Standard.Module1.Test()"
    Dim sDocType$ : sDocType = "com.sun.star.text.TextDocument"
    Dim sSupplier$
    Dim oSupplier
    Dim oModuleCfgMgr
    Dim oImageMgr
    Dim oToolbarSettings
    Dim bHasButton As Boolean
    Dim nCount As Integer
    Dim oToolbarButton()
    Dim nToolbarButtonCount As Integer
    Dim i%, j%

    REM Retrieve the module configuration manager from the
    REM central module configuration manager supplier
    sSupplier = "com.sun.star.ui.ModuleUIConfigurationManagerSupplier"
    oSupplier = CreateUnoService(sSupplier)

    REM Retrieve the module configuration manager with module identifier
    REM *** See com.sun.star.frame.ModuleManager for more information
    oModuleCfgMgr = oSupplier.getUIConfigurationManager( sDocType )
    oImageMgr = oModuleCfgMgr.getImageManager()

    oToolbarSettings = oModuleCfgMgr.getSettings( sToolbar, True )

    REM Look for our button with the CommandURL property.
    bHasButton = False
    nCount = oToolbarSettings.getCount()
    For i = 0 To nCount-1
        oToolbarButton() = oToolbarSettings.getByIndex( i )
        nToolbarButtonCount = ubound(oToolbarButton())
        For j = 0 To nToolbarButtonCount
            If oToolbarButton(j).Name = "CommandURL" Then
                If oToolbarButton(j).Value = sCmdID Then
                    bHasButton = True
                End If
            End If
        Next
    Next
Next
```



```

Dim oImageCmds(0)
Dim oImages(0)
Dim oImage
REM *** Check if image has already been added
If Not oImageMgr.hasImage( 0, sCmdID ) Then
    REM Try to load the image from the file URL
    oImage = GetImageFromURL( "file:///tmp/test.bmp" )
    If Not IsNull( oImage ) Then
        REM *** Insert new image into the Writer image manager
        oImageCmds(0) = sCmdID
        oImages(0) = oImage
        oImageMgr.insertImages( 0, oImageCmds(), oImages() )
    End If
End If

If Not bHasButton Then
    sString = "My Macro's"
    oToolBarItem = CreateToolBarItem( sCmdID, "Standard.Module1.Test" )
    oToolBarSettings.insertByIndex( nCount, oToolBarItem )
    oModuleCfgMgr.replaceSettings( sToolBar, oToolBarSettings )
End If
End Sub

Function GetImageFromURL( URL as String ) as Variant
    Dim oMediaProperties(0) As New com.sun.star.beans.PropertyValue
    Dim sProvider$ : sProvider = "com.sun.star.graphic.GraphicProvider"
    Dim oGraphicProvider

    REM Create graphic provider instance to load images from files.
    oGraphicProvider = createUnoService( sProvider )

    REM Set URL property so graphic provider is able to load the image
    oMediaProperties(0).Name = "URL"
    oMediaProperties(0).Value = URL

    REM Retrieve the com.sun.star.graphic.XGraphic instance
    GetImageFromURL = oGraphicProvider.queryGraphic( oMediaProperties() )
End Function

Function CreateToolBarItem( Command$, Label$ ) as Variant
    Dim aToolBarItem(3) as new com.sun.star.beans.PropertyValue

    aToolBarItem(0).Name = "CommandURL"
    aToolBarItem(0).Value = Command
    aToolBarItem(1).Name = "Label"
    aToolBarItem(1).Value = Label

```

```

aToolBarItem(2).Name = "Type"
aToolBarItem(2).Value = 0
aToolBarItem(3).Name = "Visible"
aToolBarItem(3).Value = true

CreateToolBarItem = aToolBarItem()
End Function

```

Time permitting, I will add code that demonstrates how to copy a custom toolbar stored in a document to another document. Time, all I need is time.

5.45. Load hidden then setVisible

In earlier versions of OOo, it was not safe to load a document hidden and then set it to visible because not all of required items were initialized. As of 2.3, however, this is now safe.

Listing 5.111: Load hidden then setVisible

```

Sub Hide_Show_Doc
    Dim Doc As Object
    Dim aMediaDesc(0) as New com.sun.star.beans.PropertyValue
    Dim sURL$

    aMediaDescriptor(0).Name = "Hidden"
    aMediaDescriptor(0).Value = TRUE
    sURL = "private:factory/swriter"

    Doc = StarDesktop.loadComponentFromURL(sURL, "_default", 0, aMediaDesc)
    Doc.getText().setString("I was hidden!")
    wait 10000
    Doc.getCurrentController().getFrame().getContainerWindow().setVisible(True)
End Sub

```

5.46. Extension Manager

Prior to OOo 3.0, you could create a service with arguments, but you had to use the process service manager.

Listing 5.112: Create the extension manager using process service manager.

```

Sub CreateExtensionManager
    Dim oListener As Object
    Dim oServiceManager
    Dim oPackagemanager
    oServiceManager = GetProcessServiceManager()
    oPackagemanager = oServiceManager.CreateInstanceWithArguments(_
        "com.sun.star.deployment.ui.PackageManagerDialog", _
        Array(ThisComponent.CurrentController.Frame.ContainerWindow, _
        "shared"))
    oPackagemanager.setDialogTitle("This is a test")

```

```

oPackagemanager.startExecuteModal(oListener)
End Sub

```

OOo 3.0 introduces CreateUnoServiceWithArguments, which provides a short cut; you do not need to create the ProcessServiceManager.

Listing 5.113: *Create the extension manager directly.*

```

Sub MainNew
    Dim oListener As Object
    Dim oPackagemanager
    oPackagemanager = CreateUnoServiceWithArguments(_
        "com.sun.star.deployment.ui.PackageManagerDialog", _
        Array(ThisComponent.CurrentController.Frame.ContainerWindow, "shared"))
    oPackagemanager.setDialogTitle("This is a test")
    oPackagemanager.startExecuteModal(oListener)
End Sub

```

5.47. Embed data in a document

Did you know that there is an API to embed stuff in a document?

<http://api.openoffice.org/docs/common/ref/com/sun/star/embed/module-ix.html>

Data embedded using the API is not available using the UI, only by your macro.

5.48. Toggle design mode

Although you can use a dispatch to toggle design mode on and off, you cannot tell if you are in design mode. This macro uses a listener to determine if design mode is on or off. I last tested this macro in 2004, and changes have been made, so, test at your own risk.

Listing 5.114: *Set design mode on or off.*

```

Sub setDesignModeOn()
    If NOT isDesignModeOn() Then
        SwitchDesignMode()
    End If
    REM Set DesignMode ON for the next time the document is opened!
    'Stardesktop.getCurrentComponent().ApplyFormDesignMode = True
    ThisComponent.ApplyFormDesignMode = True
End Sub

Sub setDesignModeOff()
    If isDesignModeOn() Then
        SwitchDesignMode()
    End If
    REM Set DesignMode OFF for the next time the document is opened!
    'Stardesktop.getCurrentComponent().ApplyFormDesignMode = False

```

```

    ThisComponent.ApplyFormDesignMode = False
End Sub

REM This function is only called when the DesignMode is off
Sub SwitchDesignMode( )
    Dim sCommand
    Dim oFrame
    Dim oDisp
    'Print "Switching design mode"
    sCommand = ".uno:SwitchControlDesignMode"
    oFrame = ThisComponent.getCurrentController().getFrame()
    oDisp = createUnoService("com.sun.star.frame.DispatchHelper")
    oDisp.executeDispatch(oFrame, ".uno:SwitchControlDesignMode", "", 0, Array())
End Sub

Function isDesignModeOn() As Boolean
    Dim oFrame          ' Current frame
    Dim oDisp            ' The created dispatcher
    Dim oParser          ' URL Transformer to parse the URL.
    Dim oStatusListener ' The status listener that is created
    Dim sListenerName    ' The type of listener that is created
    Dim oUrl as New com.sun.star.util.URL

    REM Location 3 is used to prevent the state from being
    REM recorded more than once. Probably not really required.
    repository(3, True)

    REM Parse the URL as required
    REM and then save it in the registry at location 0.
    oUrl.Complete = ".uno:SwitchControlDesignMode"
    oParser = createUnoService("com.sun.star.util.URLTransformer")
    oParser.parseStrict(oUrl)
    repository(0, oUrl)

    REM See if the current Frame supports this UNO command
    REM then save the dispatch in the registry.
    oFrame = ThisComponent.getCurrentController().getFrame()
    oDisp = oFrame.queryDispatch(oUrl, "", 0)
    repository(1, oDisp)

    REM Create the status listener and save a reference to it in the repository
    If (Not IsNull(oDisp)) Then
        sListenerName = "com.sun.star.frame.XStatusListener"
        oStatusListener = CreateUnoListener("Status_", sListenerName)
        repository(2, oStatusListener)
        oDisp.addStatusListener(oStatusListener, oUrl)
        isDesignModeOn = repository(4)
    Else

```

```

        isDesignModeOn = False
    End If
End Function

REM The definition of the listener requires this, but we do not use this.
Function Status_disposing(oEvt)
End Function

REM This is called whtn the status changes. In other words, when
REM the design mode is toggled and when the listener is first created.
Function Status_statusChanged(oEvt)
    'Print "In status changed: " & oEvt.State
    If repository(3) Then
        repository(3, False)
        repository(4, oEvt.State)
    End If
    removeStatusListener()
End Function

Function removeStatusListener()
    Dim oStatusListener ' The saved status listener to remove.
    Dim oUrl             ' The saved URL object
    Dim oDisp            ' The saved dispatcher object
    Dim x                ' This is an empty variant variable!

    REM Verify that the dispatcher exists before trying to remove
    REM the status listener.
    oDisp = repository(1)
    If NOT IsEmpty(oDisp) AND NOT IsNull(oDisp) Then
        oUrl = repository(0)
        oStatusListener = repository(2)
        repository(0, x) ' Remove the URL
        repository(1, x) ' Remove the dispatch
        repository(2, x) ' Remove the status listener
        oDisp.removeStatusListener(oStatusListener, oURL)
    End If
End Function

REM This is used to store a series of variables rather than pollute
REM the global name space. If the optional argument is passed, then
REM it is added to the array. Either way, the value is returned.
Function repository(n As Integer, Optional x)
    Static oObj(5) As Variant
    If Not IsMissing(x) Then oObj(n) = x
    repository = oObj(n)
End Function

```


6. Calc macros

6.1. Is this a spreadsheet document?

A spreadsheet document is composed of a set of sheets. Before you can use the spreadsheet specific methods, you must have a spreadsheet document. You may verify this as follows:

Listing 6.1: *Is this a Calc document, using error handling.*

```
Function IsSpreadsheetDoc(oDoc) As Boolean
    Dim s$ : s$ = "com.sun.star.sheet.SpreadsheetDocument"
    On Local Error GoTo NODOCUMENTTYPE
    IsSpreadsheetDoc = oDoc.SupportsService(s$)
NODOCUMENTTYPE:
    If Err <> 0 Then
        IsSpreadsheetDoc = False
        Resume GOON
    GOON:
    End If
End Function
```

If error handling is not an issue because the function will never be called with a null or empty argument, and the object will always implement the supportsService() method then you can use this version:

Listing 6.2: *Is this a Calc document with no error handling.*

```
Function IsSpreadsheetDoc(oDoc) As Boolean
    Dim s$ : s$ = "com.sun.star.sheet.SpreadsheetDocument"
    IsSpreadsheetDoc = oDoc.SupportsService(s$)
End Function
```

You can call the test method as follows:

```
Sub checking( )
    MsgBox IsSpreadsheetDoc(thisComponent)
End Sub
```

6.2. Display cell value, string, or formula

Listing 6.3: *Accessing a cell in a Calc document.*

```
'*****
'Author: Sasa Kelecevic
'email: scat@teol.net
Sub ExampleGetValue
    Dim oDoc As Object, oSheet As Object, oCell As Object
    oDoc=ThisComponent
    oSheet=oDoc.Sheets.getByName("Sheet1")
    oCell=oSheet.getCellByPosition(0,0) 'A1
    Rem a cell's contents can have one of the three following types:
    Print oCell.getValue()
    'Print oCell.getString()
```

```
'Print oCell.getFormula()
End Sub
```

6.3. Set cell value, format, string, or formula

Listing 6.4: Accessing a cell in a Calc document.

```
'*****
'Author: Sasa Kelecevic
'email: scat@teol.net
Sub ExampleSetValue
    Dim oDoc As Object, oSheet As Object, oCell As Object
    oDoc=ThisComponent
    oSheet=oDoc.Sheets.getByName("Sheet1")
    oCell=oSheet.getCellByPosition(0,0) 'A1
    oCell.SetValue(23658)
    'oCell.NumberFormat=2 '23658.00
    'oCell.SetString("oops")
    'oCell.setFormula("=FUNCTION()")
    'oCell.IsCellBackgroundTransparent = TRUE
    'oCell.CellBackColor = RGB(255,141,56)
End Sub
```

6.3.1. Reference a cell in another document

In your spreadsheet, you can access a cell in another document using a form similar to

```
file:///PATH/filename'#$Data.P40
```

This can also be done when setting a formula in a macro.

```
oCell = thiscomponent.sheets(0).getcellbyposition(0,0) ' A1
oCell.setFormula("=" & "'file:///home/USER/CalcFile2.sxc'#$Sheet2.K89")
```

6.4. Clear a cell

A list of things that can be cleared can be found at

<http://api.openoffice.org/docs/common/ref/com/sun/star/sheet/CellFlags.html>

Listing 6.5: Clear a cell.

```
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub ClearDefinedRange
    Dim oDoc As Object, oSheet As Object, oSheets As Object
    Dim oCellRange As Object
    Dim nSheets As Long

    oDoc = ThisComponent
    oSheets = oDoc.Sheets
```



```

nSheets = oDoc.Sheets.Count
REM Get the third sheet, as in 0, 1, 2
oSheet = oSheets.getByIndex(2)
REM You can use a range such as "A1:B2"
oCellRange = oSheet.getCellRangeByName("<range_you_set>")
oCellRange.clearContents(_
    com.sun.star.sheet.CellFlags.VALUE OR _
    com.sun.star.sheet.CellFlags.DATETIME OR _
    com.sun.star.sheet.CellFlags.STRING OR _
    com.sun.star.sheet.CellFlags.ANNOTATION OR _
    com.sun.star.sheet.CellFlags.FORMULA OR _
    com.sun.star.sheet.CellFlags.HARDATTR OR _
    com.sun.star.sheet.CellFlags.STYLES OR _
    com.sun.star.sheet.CellFlags.OBJECTS OR _
    com.sun.star.sheet.CellFlags.EDITATTR)
End Sub

```

6.5. Selected text, what is it?

Selected text in a spreadsheet can be a few different things; some of them I understand and some I do not.

1. One cell Selected. Click in a cell once and then hold down the shift key and click in the cell again.
2. Partial text in a single cell selected. Double click in a single cell and then select some text.
3. Nothing selected. Single click in a cell or tab between cells.
4. Multiple cells selected. Single click in a cell and then drag the cursor.
5. Multiple disjoint selections. Select some cells. Hold down the control key and select some more.

So far, I have not been able to distinguish the first three cases. If I can figure out how to extract the selected text in case 2, then I can solve this problem.

Listing 6.6: *Is anything selected in a Calc document.*

```

Function CalcIsAnythingSelected(oDoc As Object) As Boolean
    Dim oSels
    Dim oSel
    Dim oText
    Dim oCursor

    IsAnythingSelected = False
    If IsNull(oDoc) Then Exit Function
    ' The current selection in the current controller.
    'If there is no current controller, it returns NULL.
    oSels = oDoc.getCurrentSelection()

```

```

If IsNull(oSels) Then Exit Function
If oSels.supportsService("com.sun.star.sheet.SheetCell") Then
    Print "One Cell selected = " & oSels.getImplementationName()
    MsgBox "getString() = " & oSels.getString()
ElseIf oSels.supportsService("com.sun.star.sheet.SheetCellRange") Then
    Print "One Cell Range selected = " & oSels.getImplementationName()
ElseIf oSels.supportsService("com.sun.star.sheet.SheetCellRanges") Then
    Print "Multiple Cell Ranges selected = " & oSels.getImplementationName()
    Print "Count = " & oSels.getCount()
Else
    Print "Somethine else selected = " & oSels.getImplementationName()
End If
End Function

```

6.5.1. Simple example processing selected cells

Consider a very simple example that divides all selected cells by a single value. This example provides no error checking. Without error checking, dividing a cell by a value is easy.

Listing 6.7: *Divide a single cell by a numeric value.*

```

Sub DivideCell(oCell, dDivisor As Double)
    oCell.setValue(oCell.getValue()/dDivisor)
End Sub

```

Dividing a cell range is more difficult. Arrays are copied by reference rather than by value. Because of this, the array oRow() does not need to be copied back into the array oData().

Listing 6.8: *Divide every cell in a cell range by a single value.*

```

Sub DivideRange(oRange, dDivisor As Double)
    Dim oData()
    Dim oRow()
    Dim i As Integer
    Dim j As Integer

    oData() = oRange.getDataArray()
    For i = LBound(oData()) To UBound(oData())
        oRow() = oData(i)
        For j = LBound(oRow()) To UBound(oRow())
            oRow(j) = oRow(j) / dDivisor
        Next
    Next
    oRange.setDataArray(oData())
End Sub

```

The following code assumes that the current document is a Calc document. The current selection is obtained from the current controller and passed to the DivideRegions routine.

Listing 6.9: Divide the current selection by a numeric value.

```
Sub DivideSelectedCells
    Dim dDivisor As Double
    Dim oSels

    dDivisor = 10
    oSels = ThisComponent.getCurrentController().getSelection()
    DivideRegions(oSels, dDivisor)
End Sub
```

Do not be tempted to move the code in *Listing 6.10*, where the real work is accomplished, into *Listing 6.9*. The advantage of a separate routine becomes apparent when disjoint cells are selected; in other words, the selection is not a simple cell range. A multi-region selection is composed of multiple `SheetCellRange` selections. By separating *Listing 6.10* into its own routine, it can call itself recursively when a multiple regions are selected.

Listing 6.10: Primary work code to divide cells by a numeric value.

```
Sub DivideRegions(oSels, dDivisor As Double)
    Dim oSel
    Dim i As Integer

    If oSels.supportsService("com.sun.star.sheet.SheetCell") Then
        DivideCell(oSels, dDivisor)
    ElseIf oSels.supportsService("com.sun.star.sheet.SheetCellRanges") Then
        For i = 0 To oSels.getCount() - 1
            DivideRegions(oSels.getByIndex(i), dDivisor)
        Next
    ElseIf oSels.supportsService("com.sun.star.sheet.SheetCellRange") Then
        DivideRange(oSels, dDivisor)
    End If
End Sub
```

A `SheetCell` is also a `SheetCellRange`, so *Listing 6.7* is not really required; you can use *Listing 6.8* instead (see *Listing 6.11*).

Listing 6.11: Primary work code to divide cells by a numeric value.

```
Sub DivideRegions(oSels, dDivisor As Double)
    Dim oSel
    Dim i As Integer

    If oSels.supportsService("com.sun.star.sheet.SheetCellRanges") Then
        For i = 0 To oSels.getCount() - 1
            DivideRegions(oSels.getByIndex(i), dDivisor)
        Next
    ElseIf oSels.supportsService("com.sun.star.sheet.SheetCellRange") Then
        DivideRange(oSels, dDivisor)
    End If
```

End Sub

An uncontrolled experiment, with many running processes, leads me to believe that it is more efficient to handle a single cell as a cell, than as a range (so *Listing 6.10* should run faster than *Listing 6.11*). On the other hand, if performance is important, reorder the comparisons so that more common situations are tested and handled first.

6.5.2. Get the active cell and ignore the rest

If you want only the cell that contains the cursor, and you want to ignore the rest, you can tell the controller to select an empty range that was created by the document. The following subroutine does the following:

1. Save the current selection. This is useful if more than a single cell is active.
2. Select an empty range so that only the cell with the cursor is selected. The cell is selected with an outline around the cell, but it is not completely blacked out. If you use the controller to select a range, this method can also be used to change the selection from a completely selected cell, to merely an active cell.
3. Use the CellAddressConversion service to obtain the address of the active cell. This is new to 1.1.1, part of the "linked controls" implementation.

Listing 6.12: *Find the active cell.*

```
REM Author: Paolo Mantovani
REM email: mantovani.paolo@tin.it
Sub RetrieveTheActiveCell()
    Dim oOldSelection 'The original selection of cell ranges
    Dim oRanges       'A blank range created by the document
    Dim oActiveCell   'The current active cell
    Dim oConv         'The cell address conversion service
    REM store the current selection
    oOldSelection = ThisComponent.CurrentSelection

    oRanges = ThisComponent.CreateInstance("com.sun.star.sheet.SheetCellRanges")
    ThisComponent.CurrentController.Select(oRanges)
    'get the active cell!
    oActiveCell = ThisComponent.CurrentSelection

    REM a nice service I've just found!! :-)
    oConv =
    ThisComponent.CreateInstance("com.sun.star.table.CellAddressConversion")
    oConv.Address = oActiveCell.getCellAddress
    Print oConv.UserInterfaceRepresentation
    Print oConv.PersistentRepresentation

    'restore the old selection (but loosing the previous active cell)
    ThisComponent.CurrentController.Select(oOldSelection)
```

```
End Sub
```

6.5.3. Select a Cell

Click on a cell to select the cell. Although the cursor is not displayed in the cell, the cell is selected. Use the arrow keys to “move the cursor” and select a different cell. The behavior is the same. In OOO 2.4, the following macro used to select the entire cell so that the cell was highlighted. The behavior appears to have changed in OOO 3.0.

Listing 6.13: Select a single cell.

```
Dim oCell
Dim oSheet

REM Get the first sheet.
oSheet = ThisComponent.getSheets().getByIndex(0)
REM Get cell A2
oCell = oSheet.GetCellbyPosition( 0, 1 )
REM Move the selection to cell A2
ThisComponent.CurrentController.Select(oCell)
```

In OOO 2.4, to select a cell with an outline around it, you had to select the cell, and then select an empty range to change the focus.

Listing 6.14: Select a single cell so it has an outline.

```
Sub MoveCursorToCell
Dim oCell
Dim oSheet
Dim oRanges

REM Get the first sheet.
oSheet = ThisComponent.getSheets().getByIndex(0)
REM Get cell A2
oCell = oSheet.GetCellbyPosition( 0, 1 )
REM Move the selection to cell A2
ThisComponent.CurrentController.Select(oCell)

REM Select an empty range..
oRanges = ThisComponent.CreateInstance("com.sun.star.sheet.SheetCellRanges")
ThisComponent.CurrentController.Select(oRanges)
End Sub
```

Use the `SetInputMode` dispatch to focus the cursor in the cell in the input mode (this is the same as double clicking in a cell, or pressing F2).

```
Dim oFrame
Dim oDisp
oFrame = ThisComponent.CurrentController.Frame
oDisp = createUnoService("com.sun.star.frame.DispatchHelper")
```

```
oDisp.executeDispatch(oFrame, ".uno:SetInputMode", "", 0, Array())
```

6.6. Human readable address of cell

The `com.sun.star.table.CellAddressConversion` service can be used to obtain a human readable text string that represents the address of a cell. I have not found any documentation on this service, but as of OOO 1.1.1, it seems to work well enough. The following code snippet assumes that the current document is a Calc document, and that only a single cell is selected.

Listing 6.15: *Cell address in a readable form using CellAddressConversion.*

```
oActiveCell = ThisComponent.getCurrentSelection()
oConv =
ThisComponent.createInstance("com.sun.star.table.CellAddressConversion")
oConv.Address = oActiveCell.getCellAddress
Print oConv.UserInterfaceRepresentation
Print oConv.PersistentRepresentation
```

If dealing with a cell range address, use `CellRangeAddressConversion` instead.

I created the following function before I knew about the `CellAddressConversion` service.

Listing 6.16: *Cell address in a readable form.*

```
'Given a cell, extract the normal looking address of a cell
'First, the name of the containing sheet is extracted.
'Second, the column number is obtained and turned into a letter
'Lastly, the row is obtained. Rows start at 0 but are displayed as 1
Function PrintableAddressOfCell(the_cell As Object) As String
    PrintableAddressOfCell = "Unknown"
    If Not IsNull(the_cell) Then
        PrintableAddressOfCell = the_cell.getSpreadSheet().getName + ":" + _
            ColumnNumberToString(the_cell.CellAddress.Column) +
            (the_cell.CellAddress.Row+1)
    End If
End Function

' Columns are numbered starting at 0 where 0 corresponds to A
' They run as A-Z,AA-AZ,BA-BZ,...,IV
' This is essentially a question of how do you convert a Base 10 number to
' a base 26 number.
' Note that the_column is passed by value!
Function ColumnNumberToString(ByVal the_column As Long) As String
    Dim s$
    'Save this so I do NOT modify the parameter.
    'This was an icky bug that took me a while to find
    Do while the_column >= 0
        s$ = Chr(65 + the_column MOD 26) + s$
```

```

        the_column = the_column \ 26 - 1
    Loop
    ColumnNumberToString = s$
End Function

```

6.7. Insert formatted date into cell

Insert the date into the current cell. An error message is displayed if the current document is not a spreadsheet. Code is provided to format the date in the style of your choice, you need to remove the comments. A final warning, this macro assumes that only a single cell is selected and it will fail if this is not the case. If you want to deal with the possibility of having more than one cell selected, then look at the section that deals with selected text in a Calc document.

Listing 6.17: Formatted date in a cell.

```

'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'uses: FindCreateNumberFormatStyle
Sub InsertDateIntoCell
    Dim oSelection 'The currently selected cell
    Dim oFormats   'Available formats

    REM Verify that this is a Calc document
    If ThisComponent.SupportsService("com.sun.star.sheet.SpreadsheetDocument")
    Then
        oSelection = ThisComponent.CurrentSelection

        Rem Set the time, date, or date and time
        'oSelection.setValue(DateValue(Now())) 'Set only the date
        'oSelection.setValue(TimeValue(Now())) 'Set only the time
        oSelection.setValue(Now())             'Set the date and time

        Rem I could use FunctionAccess to set the date and/or time.
        'Dim oFunction 'Use FunctionAccess service to call the Now function
        'oFunction = CreateUnoService("com.sun.star.sheet.FunctionAccess")
        'oFunction.NullDate = ThisComponent.NullDate
        'oSelection.setValue(oFunction.callFunction("NOW", Array()))

        Rem Set the date number format to default
        oFormats = ThisComponent.NumberFormats
        Dim aLocale As New com.sun.star.lang.Locale
        oSelection.NumberFormat = oFormats.getStandardFormat(_
            com.sun.star.util.NumberFormat.DATETIME, aLocale)

        Rem Set the format to something completely different
        'oSelection.NumberFormat = FindCreateNumberFormatStyle(_

```

```

        ' "YYYYMMDD.hhmmss", doc)
Else
    MsgBox "This macro must be run in a spreadsheet document"
End If
End Sub

```

6.7.1. A shorter way to do it

Consider the following two methods (provided by Shez):

Listing 6.18: *Formatted date in a cell with a shorter method.*

```

Sub DateNow
    Dim here As Object
    here=ThisComponent.CurrentSelection
    here.SetValue(DateValue(Now))
    here.NumberFormat=75
End sub

Sub TimeNow
    Dim here As Object
    here=ThisComponent.CurrentSelection
    here.SetValue(TimeValue(Now))
    here.NumberFormat=41
End sub

```

These two methods assume a Calc document and hard code the display format.

6.8. Display selected range in message box

Listing 6.19: *Display selected range.*

```

' *****
' Author: Sasa Kelecevic
' email: scat@teol.net
' This macro will take the current selection and print a message
' box indicating the selected range and the number of selected
' cells
Sub SelectedCells
    oSelect=ThisComponent.CurrentSelection.getRangeAddress
    oSelectColumn=ThisComponent.CurrentSelection.Columns
    oSelectRow=ThisComponent.CurrentSelection.Rows

    CountColumn=oSelectColumn.getCount
    CountRow=oSelectRow.getCount

    oSelectSC=oSelectColumn.getByIndex(0).getName
    oSelectEC=oSelectColumn.getByIndex(CountColumn-1).getName

    oSelectSR=oSelect.StartRow+1
    oSelectER=oSelect.EndRow+1

```



```

NoCell=(CountColumn*CountRow)

If CountColumn=1 AND CountRow=1 Then
    MsgBox("Cell " + oSelectSC + oSelectSR + chr(13) + "Cell No = " + NoCell,,
"SelectedCells")
Else
    MsgBox("Range(" + oSelectSC + oSelectSR + ":" + oSelectEC + oSelectER + ") "
+ chr(13) + "Cell No = " + NoCell,, "SelectedCells")
End If
End Sub

```

6.9. Fill selected range with text

This simple macro iterates through the selected rows and columns setting the text to “OOPS”.

Listing 6.20: Iterate through cells and set the text.

```

'*****
'Author: Sasa Kelecevic
'email: scat@teol.net
Sub FillCells
    oSelect=ThisComponent.CurrentSelection
    oColumn=oSelect.Columns
    oRow=oSelect.Rows
    For nc= 0 To oColumn.getCount-1
        For nr = 0 To oRow.getCount-1
            oCell=oSelect.getCellByPosition (nc,nr).setString ("OOOPS")
        Next nr
    Next nc
End Sub

```

Although this loop technique is frequently used with no complaints, it might be faster to set all of the values at one time using the method `setDataArray()`.

6.10. Some stats on a selected range

Listing 6.21: Information about the selected range.

```

'Author: Sasa Kelecevic
'email: scat@teol.net
'Print a message indicating the selected range and the number of
' selected cells
Sub Analyze
    sSum="=SUM (" + GetAddress + ") "
    sAverage="=AVERAGE (" + GetAddress + ") "
    sMin="=MIN (" + GetAddress + ") "
    sMax="=MAX (" + GetAddress + ") "
    CellPos(7,6).setString(GetAddress)
    CellPos(7,8).setFormula(sSum)

```

```

    CellPos(7,8).NumberFormat=2
    CellPos(7,10).setFormula(sAverage)
    CellPos(7,10).NumberFormat=2
    CellPos(7,12).setFormula(sMin)
    CellPos(7,12).NumberFormat=2
    CellPos(7,14).setFormula(sMax)
    CellPos(7,14).NumberFormat=2
End sub
Function GetAddress 'selected cell(s)
    oSelect=ThisComponent.CurrentSelection.getRangeAddress
    oSelectColumn=ThisComponent.CurrentSelection.Columns
    oSelectRow=ThisComponent.CurrentSelection.Rows

    CountColumn=oSelectColumn.getCount
    CountRow=oSelectRow.getCount

    oSelectSC=oSelectColumn.getByIndex(0).getName
    oSelectEC=oSelectColumn.getByIndex(CountColumn-1).getName

    oSelectSR=oSelect.StartRow+1
    oSelectER=oSelect.EndRow+1
    NoCell=(CountColumn*CountRow)

    If CountColumn=1 AND CountRow=1 then
        GetAddress=oSelectSC+oSelectSR
    Else
        GetAddress=oSelectSC+oSelectSR+": "+oSelectEC+oSelectER
    End If
End Function
Function CellPos(lColumn As Long,lRow As Long)
    CellPos= ActiveSheet.getCellByPosition (lColumn,lRow)
End Function
Function ActiveSheet
    ActiveSheet=StarDesktop.CurrentComponent.CurrentController.ActiveSheet
End Function
Sub DeleteDbRange(sRangeName As String)
    oRange=ThisComponent.DatabaseRanges
    oRange.removeByName (sRangeName)
End Sub

```

6.11. Database range

I modified the following macros by declaring all variables and adding improved error checking. I also created the following test routine.

Listing 6.22: *Test the define and remove database range routine.*

```

Sub TestDefineAndRemoveRange
    Dim s As String

```

```

Dim oDoc
Dim sNames()
oDoc = ThisComponent
s = "blah1"
If oDoc.DatabaseRanges.hasByName(s) Then Print "It already Exists"
sNames() = oDoc.DatabaseRanges.getElementNames()
MsgBox Join(sNames(), CHR$(10)), 0, "Before Adding " & s
DefineDbRange(s)
sNames() = oDoc.DatabaseRanges.getElementNames()
MsgBox Join(sNames(), CHR$(10)), 0, "After Adding " & s
DeleteDbRange(s)
sNames() = oDoc.DatabaseRanges.getElementNames()
MsgBox Join(sNames(), CHR$(10)), 0, "After Removing " & s
End Sub

```

6.11.1. Set selected cells to a database range

Listing 6.23: Set selected cells to a database range.

```

'Author: Sasa Kelecevic
'email: scat@teol.net
'Modified by : Andrew Pitonyak
Sub DefineDbRange(sRangeName As String) 'selected range
    Dim oSelect
    Dim oRange
    Dim oRanges
    On Error GoTo DUPLICATENAME
    oSelect=ThisComponent.CurrentSelection.RangeAddress
    oRanges = ThisComponent.DatabaseRanges
    If oRanges.hasByName(sRangeName) Then
        MsgBox("Duplicate name",,"INFORMATION")
    Else
        oRange= oRanges.addNewByName (sRangeName,oSelect)
    End If
DUPLICATENAME:
    If Err <> 0 Then
        MsgBox("Error adding database name " & sRangeName,, "INFORMATION")
    End If
End Sub

```

6.11.2. Delete database range

Listing 6.24: Delete a database range.

```

'Author: Sasa Kelecevic
'email: scat@teol.net
'Modified by : Andrew Pitonyak
Sub DeleteDbRange(sRangeName As String)
    Dim oRanges

```

```

oRanges=ThisComponent.DatabaseRanges
If oRanges.hasByName(sRangeName) Then
    oRanges.removeByName(sRangeName)
End If
End Sub

```

6.12. Table borders

When a structure is obtained from a service, a copy of the structure is returned rather than a reference to the structure. This prevents you from directly modifying the structure. Instead, you must make a copy of the structure, modify it, and then copy it back (see Listing 6.25).

Listing 6.25: *Set Calc border using a temporary.*

```

'Author: Niklas Nebel
'email: niklas.nebel@sun.com
' setting_borders_in_calc
oRange = ThisComponent.Sheets(0).getCellRangeByPosition(0,1,0,63)
aBorder = oRange.TableBorder

aBorder.BottomLine = lHor
oRange.TableBorder = aBorder

```

Niklas included an example that fails because it modifies a temporary structure.

Listing 6.26: *This fails because the TableBorder structure is a copy.*

```

lHor.Color = 0
lHor.InnerLineWidth = 0
lHor.OuterLineWidth = 150
Dim lHor As New com.sun.star.table.BorderLineHor.LineDistance
oRange = ThisComponent.Sheets(0).getCellRangeByPosition(0,1,0,63)
oRange.TableBorder.BottomLine = lHor

```

And here is a working solution from David Woody [dwoody1@airmail.net]

Listing 6.27: *Another example setting a Calc border.*

```

Sub Borders
    Dim aBorder, oRange, oDoc, oSheets
    Dim TableBorder As New com.sun.star.table.TableBorder
    Dim aTopLine As New com.sun.star.table.BorderLine

    oDoc = ThisComponent
    oSheets = oDoc.Sheets(0)
    oRange = oSheets.getCellRangeByPosition(8,2,8,5)
    aBorder = oRange.TableBorder
    aTopLine.OuterLineWidth = 250
    aTopLine.InnerLineWidth = 0
    aTopLine.Color = 170000

```

```

oRange.TableBorder.IsTopLineValid = 1
aBorder.TopLine = aTopLine
oRange.TableBorder = aBorder
End Sub

```

6.13. Sort range

The macro in Listing 6.28 performs a descending sort based on columns. In other words, the rows are moved around.

Listing 6.28: *Descending sort in a Calc document.*

```

'*****
'Author: Sasa Kelecevic
'email: scat@teol.net
Sub SortRange
    Dim oSheetDSC,oDSCRange As Object
    Dim aSortFields(0) As New com.sun.star.util.SortField
    Dim aSortDesc(0) As New com.sun.star.beans.PropertyValue

    'set your sheet name
    oSheetDSC = ThisComponent.Sheets.getByName("Sheet1")

    'set your range address
    oDSCRange = oSheetDSC.getCellRangeByName("A1:L16")
    ThisComponent.getCurrentController.select(oDSCRange)

    aSortFields(0).Field = 0
    aSortFields(0).SortAscending = FALSE

    aSortDesc(0).Name = "SortFields"
    aSortDesc(0).Value = aSortFields()
    oDSCRange.Sort(aSortDesc())
End Sub

```

Assume that I want to sort on the second and third columns where the first column is text and the second column is to be sorted numerically. I will need two sort fields rather than one.

```

Sub Main
    Dim oSheetDSC As Object, oDSCRange As Object
    Dim aSortFields(1) As New com.sun.star.util.SortField
    Dim aSortDesc(0) As New com.sun.star.beans.PropertyValue

    'set your sheet name
    oSheetDSC = THISCOMPONENT.Sheets.getByName("Sheet1")

    'set your range address

```

```

oDSCRange = oSheetDSC.getCellRangeByName("B3:E6")
THISCOMPONENT.getCurrentController.select(oDSCRange)

'Another valid sort type is
'com.sun.star.util.SortFieldType.AUTOMATIC
'Remember that the fields are zero based so this starts sorting
'in column B, not column A
aSortFields(0).Field = 1
aSortFields(0).SortAscending = TRUE
'It turns out that there is no reason to set the field type while
'sorint in a spreadsheet document becuase this is ignored.
'A spreadsheet alraedy knows the type.
'aSortFields(0).FieldType = com.sun.star.util.SortFieldType.ALPHANUMERIC

aSortFields(1).Field = 2
aSortFields(1).SortAscending = TRUE
aSortFields(1).FieldType = com.sun.star.util.SortFieldType.NUMERIC

aSortDesc(0).Name = "SortFields"
aSortDesc(0).Value = aSortFields()      ' aSortFields(0)
oDSCRange.Sort(aSortDesc())            ' aSortDesc(0)
End Sub

```

To specify the first row as a header row, use another property. Be certain that you dimension enough properties.

```

aSortDesc(1).Name = "ContainsHeader"
aSortDesc(1).Value = True

```

Bernard Marcelly verified that the “Orientation” property works properly. To set the orientation, use the property “Orientation” and set the value to one of the following:

```

com.sun.star.table.TableOrientation.ROWS
com.sun.star.table.TableOrientation.COLUMNS

```

When sorting using the GUI, you can not sort on more than three rows or columns at a time. Bernard Marcelly pointed out that you can not circumvent this limitation with a macro. There is a limitation of three rows or columns.

6.14. Display all data in a column

While traversing the cell and printing values, I want to print information about the cell. Here is how I did it.

```

Sub PrintDataInColumn (a_column As Integer)
    Dim oCells As Object, aCell As Object, oDoc As Object
    Dim oColumn As Object, oRanges As Object

    oDoc = ThisComponent
    oColumn = oDoc.Sheets(0).Columns(a_column)

```

```

Print "Using column " + oColumn.getName
oRanges = oDoc.CreateInstance("com.sun.star.sheet.SheetCellRanges")
oRanges.InsertByName("", oColumn)
oCells = oRanges.Cells.CreateEnumeration
If Not oCells.hasMoreElements Then Print "Sorry, no text to display"
While oCells.hasMoreElements
    aCell = oCells.nextElement
    'This next Function is defined elsewhere in this document!
    MsgBox PrintableAddressOfCell(aCell) + " = " + aCell.String
Wend
End Sub

```

6.15. Using Outline (Grouping) Methods

Ryan Nelson [ryan@aubelius-mfg.com] told me about the outline capability in Calc and then asked how to do this in a macro. There are two things to keep in mind. The first is that it is the sheet that adds and removes grouping, and the second is that the parameters must be correct.

<http://api.openoffice.org/docs/common/ref/com/sun/star/sheet/XSheetOutline.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/table/TableOrientation.html>

```

Option Explicit
Sub CalcGroupingExample
    Dim oDoc As Object, oRange As Object, oSheet As Object
    oDoc = ThisComponent
    If Not oDoc.SupportsService("com.sun.star.sheet.SpreadsheetDocument") Then
        MsgBox "This macro must be run from a spreadsheet document", 64, "Error"
    End If
    oSheet=oDoc.Sheets.getByName("Sheet1")
    ' Parmes are (left, top, right, bottom)
    oRange = oSheet.getCellRangeByPosition(2,1,3,2)
    'Could also use COLUMNS
    oSheet.group(oRange.getRangeAddress(),
com.sun.star.table.TableOrientation.ROWS)
    Print "I just grouped the range"
    oSheet.unGroup(oRange.getRangeAddress(),
com.sun.star.table.TableOrientation.ROWS)
    Print "I just ungrouped the range"
End Sub

```

6.16. Protecting your data

It is easy to protect your spreadsheets, you only need to get your sheet and then protect it. My experiments indicate that although you do not generate an error when you choose to protect an entire document, it does not protect the entire document.

```

Sub ProtectSpreadsheet
    Dim oDoc As Object, oSheet As Object

```

```

oDoc = ThisComponent
oSheet=oDoc.Sheets.getByName("Sheet1")
oSheet.protect("password")
Print "Protect value = " & oSheet.isProtected()
oSheet.unprotect("password")
Print "Protect value = " & oSheet.isProtected()
End Sub

```

6.17. Setting header and footer text

This macro will set the header for every sheet to “Sheet: <sheet_name>”. The headers and footers are set using identical methods, just change “Header” to “Footer” in the calls. Special thanks to Oliver Brinzing [OliverBrinzing@t-online.de] for filling in the holes that I did not know, namely that I had to write the header back into the document.

```

Sub SetHeaderTextInSpreadSheet
    Dim oDoc, oSheet, oPstyle, oHeader
    Dim oText, oCursor, oField
    Dim oStyles
    Dim sService$
    oDoc = ThisComponent
    ' Get the pagestyle for the currently active sheet.
    oSheet = oDoc.CurrentController.getActiveSheet
    oStyles = oDoc.StyleFamilies.getByName("PageStyles")
    oPstyle = oStyles.getByName(oSheet.PageStyle)

    ' Turn headers on and then make them shared!
    oPstyle.HeaderOn = True
    oPstyle.HeaderShared = True

    ' The is also a RightText and a LeftText
    oHeader = oPstyle.RightPageHeaderContent
    oText = oHeader.CenterText

    ' You may now set the text object to be anything you desire
    ' Use setString() from the text object to set simple text.
    ' Use a cursor to insert a field (such as the current sheet name).
    ' First, clear any existing text!
    oText.setString("")
    oCursor = oText.createTextCursor()
    oText.insertString(oCursor, "Sheet: ", False)
    ' This will have the sheet name of the current sheet!
    sService = "com.sun.star.text.TextField.SheetName"
    oField = oDoc.createInstance(sService)
    oText.insertTextContent(oCursor, oField, False)

    ' And now for the part that holds the entire thing together,
    ' You must write the header object back because we have been

```



```

' modifying a temporary object
oPstyle.RightPageHeaderContent = oHeader
End Sub

```

6.18. Copying spreadsheet cells

6.18.1. Copy entire sheet to a new document

The following macro copies the contents of a given sheet into a newly created of a second document.

```

'Author: Stephan Wunderlich [stephan.wunderlich@sun.com]
Sub CopySpreadsheet
    Dim doc1
    Dim doc2
    doc1 = ThisComponent
    selectSheetByName(doc1, "Sheet2")
    dispatchURL(doc1, ".uno:SelectAll")
    dispatchURL(doc1, ".uno:Copy")
    doc2 = StarDesktop.loadComponentFromUrl("private:factory/scalc" , _
        "_blank", 0, dimArray())
    doc2.getSheets().insertNewByName("inserted", 0)
    selectSheetByName(doc2, "inserted")
    dispatchURL(doc2, ".uno:Paste")
End Sub

Sub selectSheetByName(oDoc, sheetName)
    oDoc.getCurrentController().select(oDoc.getSheets().getByName(sheetName))
End Sub

Sub dispatchURL(oDoc, aURL)
    Dim noProps()
    Dim URL As New com.sun.star.util.URL
    Dim frame
    Dim transf
    Dim disp

    frame = oDoc.getCurrentController().getFrame()
    URL.Complete = aURL
    transf = createUnoService("com.sun.star.util.URLTransformer")
    transf.parseStrict(URL)

    disp = frame.queryDispatch(URL, "", _
        com.sun.star.frame.FrameSearchFlag.SELF _
        OR com.sun.star.frame.FrameSearchFlag.CHILDREN)
    disp.dispatch(URL, noProps())
End Sub

```

6.19. Select a named range

Use Insert | Names | Define to open the Define Names dialog. The More button expands the dialog so that you can provide more information about the range. A named range is not the same as a database range defined using Data | Define Range. The following macro displays all of the named ranges contained in a document.

Listing 6.29: Display all named ranges in a Calc document.

```
oRanges = ThisComponent.NamedRanges
MsgBox "Current named ranges = " & CHR$(10) & _
    Join(oRanges.getElementNames(), CHR$(10))
```

A named range supports the `getReferencePosition()` method, which returns a cell address used as a base for relative references in the content. The `getContent()` method of the named range returns a string representation of the named range. The `GetGlobalRangeByName` macro by Rob Gray, simplifies porting Excel VBA macros to OpenOffice.org.

```
REM Author: Rob Gray
REM Email: robberbaron@optusnet.com.au
REM Modified from a macro contained in Andrew Pitonyak's document.
REM This makes it easier to transition from VBA global ranges and to
REM separate parameters onto a Config sheet
Function GetGlobalRangeByName(rngname As String)
    Dim oSheet          'Sheet containing the named range
    Dim oNamedRange     'The named range object
    Dim oCellAddr       'Address of the upper left cell in the named range

    If NOT ThisComponent.NamedRanges.hasByName(rngname) Then
        MsgBox "Sorry, the named range !" & rngname & _
            "! does not exist", "MacroError", 0+48
        Exit function
    End If

    REM The oNamedRange object supports the XNamedRange interface
    oNamedRange = ThisComponent.NamedRanges.getByName(rngname)
    'Print "Named range content = " & oNamedRange.getContent()

    REM Get the com.sun.star.table.CellAddress service
    oCellAddr = oNamedRange.getReferencePosition()

    REM Now, get the sheet that matters!
    oSheet = ThisComponent.Sheets.getByIndex(oCellAddr.Sheet+1)
    GetGlobalRangeByName = oSheet.getCellRangeByName(rngname)

    REM now can apply .GetCellByPosition(0,0).string etc
End function
```

My original exposure to named ranges, was to select a named range. I modified my original macro and created SelectNamedRange instead.

```
Sub SelectNamedRange(rngname As String, Optional oDoc)
    REM Author: Andrew Pitonyak
    Dim oSheet          'Sheet containing the named range
    Dim oNamedRange     'The named range object
    Dim oCellAddr       'Address of the upper left cell in the named range
    Dim oRanges         'All of the named ranges

    If IsMissing(oDoc) Then oDoc = ThisComponent
    oRanges = oDoc.NamedRanges
    If NOT oRanges.hasByName(rngname) Then
        MsgBox "Sorry, the named range " & rngname & _
            " does not exist" & CHR$(10) & _
            "Current named ranges = " & CHR$(10) & _
            Join(oRanges.getElementNames(), CHR$(10))
        Exit Sub
    End If

    REM The oNamedRange object supports the XNamedRange interface
    oNamedRange = oRanges.getByNamed(rngname)
    'Print "Named range content = " & oNamedRange.getContent()

    oCellAddr = oNamedRange.getReferencePosition()

    REM Now, get the sheet that matters!
    oSheet = oDoc.Sheets.getByIndex(oCellAddr.Sheet)

    REM You can then use the current controller
    REM to select what must be selected.
    REM select ( VARIANT )
    REM setActiveSheet ( OBJECT )
    REM setFirstVisibleColumn ( LONG )
    REM setFirstVisibleRow ( LONG )
    oDoc.getCurrentController().setActiveSheet(oSheet)

    REM The sheet can return the range based on the name
    REM oSheet.getCellRangeByName(rngname)
    REM The sheet can also return a range by position, if you know it.

    REM This selects the ENTIRE range
    Dim oRange
    oRange = oSheet.getCellRangeByName(rngname)
    oDoc.getCurrentController().select(oRange)
End Sub
```

6.19.1. Select an entire column

Select an entire column by obtaining the column from the sheet. Use the current controller to select the column. (This macro is for you John Ward <http://digiassn.blogspot.com/>).

Listing 6.30: *Select the third column in the first sheet*

```
Sub SelCol()  
    Dim oSheet  
    Dim oCol  
    oSheet = ThisComponent.getSheets().getByIndex(0)  
    oCol = oSheet.getColumns().getByIndex(2)  
    ThisComponent.getCurrentController().select(oCol)  
End Sub
```

6.19.2. Select an entire row

Select an entire row by obtaining the row from the sheet. Use the current controller to select the row. (This macro is for you John Ward <http://digiassn.blogspot.com/>).

Listing 6.31: *Select the third column in the first sheet*

```
Sub SelCol()  
    Dim oSheet  
    Dim oRow  
    oSheet = ThisComponent.getSheets().getByIndex(0)  
    oRow = oSheet.getRows().getByIndex(2)  
    ThisComponent.getCurrentController().select(oRow)  
End Sub
```

6.20. Convert data in column format into rows

Cut and paste data from a web page containing name and address information formatted in a column as follows:

```
customer1  
address1a  
address1b  
  
customer2  
address2a  
address2b
```

The macro shown below converts these records into a row format so that they can be used to merge data into a form letter.

```
Customer  address1  address2  
-----  
customer1 address1a address1b  
customer2 address2a address2b
```

[Andy notes:] The following macro assumes that the data is located in the first sheet with the data starting in the first row and the first column. The data is located in the first row and column afterwards. I always run my macros using Option Explicit, but the variables are not declared in this macro. Although this might not be the best method, it certainly works.

```
REM ---Arrange Calc colum data into rows.
REM Author: David Kwok
REM Email: dkwok@iware.com.au
Sub ColumnsToRows
    Dim oDoc

    oDoc = ThisComponent
    int_col = 0
    int_row = 0
    osheet = oDoc.Sheets.getByIndex(0)
    oDoc.CurrentController.Select(osheet.GetCellByPosition(0,0))
    Cellstring = oDoc.getCurrentSelection.getstring
    loop_col = int_col
    loop_row = int_row
    row_cnt = int_row
    Do While Cellstring <> ""
        col_cnt = 1
        REM the end number, 3, depends on the number of fields in a record
        For xx = 1 To 3

            oRangeOrg = osheet.GetCellByPosition(loop_col,loop_row).Rangeaddress
            oRangecpy = osheet.GetCellByPosition(col_cnt,row_cnt).Rangeaddress
            oCellCpy = osheet.GetCellByPosition(oRangecpy.StartColumn, _
                oRangecpy.StartRow).CellAddress
            osheet.MoveRange(oCellcpy, oRangeOrg)
            col_cnt = col_cnt + 1
            loop_row = loop_row + 1

        Next xx
        oDoc.CurrentController.Select(osheet.GetCellByPosition(loop_col,loop_row))
        cellstring = oDoc.getCurrentSelection.getstring
        row_cnt = row_cnt + 1
    Loop
    osheet.Columns.removeByIndex(0,1) 'tidying up
    osheet.Rows.insertByIndex(0,1)
    osheet.GetCellByPosition(0,0).string = "Customer"
    osheet.GetCellByPosition(1,0).string = "Address1"
    osheet.GetCellByPosition(2,0).string = "Address2"
    osheet.Columns(0).OptimalWidth = true 'adjusting the width
    osheet.Columns(1).OptimalWidth = true
    osheet.Columns(2).OptimalWidth = true
End Sub
```

6.21. Toggle Automatic Calculation

Chris Clementson sent me a macro to toggle a spreadsheet automatic calculation on and off using the dispatch functionality. I only included enough to show the argument:

```
Dim oProp(0) As New com.sun.star.beans.PropertyValue
oProp(0).Name = "AutomaticCalculation"
oProp(0).Value = False
dispatcher.executeDispatch(document, ".uno:AutomaticCalculation", "", 0,
oProp())
```

Calc documents implement the XCalculatable interface, which defines the methods shown in the table below.

Method	Description
calculate()	Recalculate all dirty cells.
calculate all()	Recalculate all cells.
isAutomaticCalculationEnabled()	Return True if automatic calculation is enabled.
enableAutomaticCalculation(boolean)	Enable or disable automatic calculation.

The following code disables and then enables automatic calculation:

```
ThisComponent.enableAutomaticCalculation(False)
ThisComponent.enableAutomaticCalculation(True)
```

6.22. Which cells are used in a sheet?

A SheetCellCursor implements the methods gotoStartOfUsedArea(boolean) and gotoEndOfUsedArea(boolean), which move the cursor to the start and end of the area used on a spreadsheet. Thanks to Johnny Rosenberg for an improved version of these functions.

```
Sub testEndColRow
    Dim oSheet
    Dim oCell
    Dim nEndCol As Integer
    Dim nEndRow As Integer
    oSheet = ThisComponent.Sheets.getByIndex( 0 )
    nEndCol = getLastUsedColumn(oSheet)
    nEndRow = getLastUsedRow(oSheet)
    oCell = oSheet.GetCellByPosition( nEndCol + 1, nEndRow + 1 )
    oCell.String = "test"
End Sub
```

```
Function GetLastUsedColumn(oSheet) As Integer
    Dim oCursor
    oCursor = oSheet.createCursor
    oCursor.GotoEndOfUsedArea(True)
    GetLastUsedColumn = oCursor.RangeAddress.EndColumn
End Function
```

```

Function GetLastUsedRow(oSheet) As Integer
    Dim oCursor
    oCursor = oSheet.createCursor
    oCursor.GotoEndOfUsedArea(True)
    GetLastUsedRow = oCursor.RangeAddress.EndRow
End Function

```

6.23. Searching a Calc document

Gerrit Jasper provided a macro to search a Calc document that first determines the range of cells that are used, and then obtains each cell and checks it against a string. I made several modifications:

```

REM Return the cell that contains the text
Function uFindString(sString$, oSheet) As Variant
    Dim nCurCol As Integer
    Dim nCurRow As Integer
    Dim nEndCol As Integer
    Dim nEndRow As Integer
    Dim oCell As Object
    Dim oCursor As Object
    Dim aAddress As Variant
    Dim sFind As String

    oCell = oSheet.GetCellbyPosition( 0, 0 )
    oCursor = oSheet.createCursorByRange(oCell)
    oCursor.GotoEndOfUsedArea(True)
    aAddress = oCursor.RangeAddress
    nEndRow = aAddress.EndRow
    nEndCol = aAddress.EndColumn

    For nCurCol = 0 To nEndCol      'Go through the range column by column,
        For nCurRow = 0 To nEndRow  'row by row.
            oCell = oSheet.GetCellByPosition( nCurCol, nCurRow )
            sFind = oCell.String      'Get cell contents.
            If sFind = sString then
                uFindString = oCell
                Exit Function
            End If
        Next
    Next
End Function

```

In a small sheet, I was able to find the cell that contained the text in around 1184 clock ticks. Next, I modified the macro to use a data array. Using a data array takes the time down to closer to 54 clock ticks – much faster.

```

REM Return the cell that contains the text

```

```

Function uFindString_2(sString$, oSheet) As Variant
    Dim nCurCol As Integer
    Dim nCurRow As Integer
    Dim oCell As Object
    Dim oCursor As Object
    Dim oData
    Dim oRow

    oCell = oSheet.GetCellbyPosition( 0, 0 )
    oCursor = oSheet.createCursorByRange(oCell)
    oCursor.GotoEndOfUsedArea(True)
    oData = oCursor.getDataArray()
    For nCurRow = LBound(oData) To UBound(oData)
        oRow = oData(nCurRow)
        For nCurCol = LBound(oRow) To UBound(oRow)
            If (oRow(nCurCol) = sString$) Then
                uFindString_2 = oSheet.GetCellbyPosition( nCurCol, nCurRow )
                Exit Function
            End If
        Next
    Next
End Function

```

Searching the sheet directly is much faster at 34 ticks!

```

REM Find the first cell that contains sString$
REM If bWholeWord is True, then the cell must contain ONLY the text
REM as indicated. If bWholeWord is False, then the cell must only contain
REM the requested string.
Function SimpleSheetSearch(sString$, oSheet, bWholeWord As Boolean) As Variant
    Dim oDescriptor
    Dim oFound
    REM Create a descriptor from a searchable document.
    oDescriptor = oSheet.createSearchDescriptor()
    REM Set the text for which to search and other
    REM
    http://api.openoffice.org/docs/common/ref/com/sun/star/util/SearchDescriptor.htm
    1
    With oDescriptor
        .SearchString = sString$
        REM These all default to false
        REM SearchWords forces the entire cell to contain only the search string
        .SearchWords = bWholeWord
        .SearchCaseSensitive = False
    End With
    REM Find the first one
    oFound = oSheet.findFirst(oDescriptor)
    SimpleSheetSearch = oFound

```



```

REM Do you really want to find more instances
REM You can continue the search using a cell if you want!
'Do While Not IsNull(oFound)
'  Print oFound.GetString()
'  oFound = oSheet.FindNext( oFound, oDescriptor)
'Loop
End Function

```

As usual, the built in functionality is much faster than a macro coded solution. This does, however also clearly demonstrate that it usually better to obtain a chunk of data using the data array methods than to operate on one cell at a time. This is the macro that I used to check the run time:

```

Sub SimpleSheetSearchTest
    Dim nItCount As Integer
    Dim nMaxIt As Integer
    Dim lTick1 As Long
    Dim lTick2 As Long
    Dim oSheet
    Dim oCell
    Dim s As String

    nMaxIt = 10
    oSheet = ThisComponent.getSheets().getByIndex(0)
    lTick1 = GetSystemTicks()
    For nItCount = 1 To nMaxIt
        oCell = SimpleSheetSearch("hello", oSheet, True)
        'oCell = uFindString("hello", oSheet)
        'oCell = uFindString_2("hello", oSheet)
    Next
    lTick2 = GetSystemTicks()

    s = s & "Search took " & (lTick2 - lTick1) & " ticks for " & _
        nMaxIt & " iterations " & CHR$(10) & _
        CStr((lTick2 - lTick1) / nMaxIt) & _
        " ticks per iteration" & CHR$(10)

    If IsEmpty(oCell) OR IsNull(oCell) Then
        s = s & "Text not found" & CHR$(10)
    Else
        s = s & "col = " & oCell.CellAddress.Column & _
            " row = " & oCell.CellAddress.Row & CHR$(10)
    End If
    MsgBox s, 0, "Compare Search Times"
End Sub

```

Tip

By default, the built in search capability searches based on the cell's formula (SearchType = 0). Set the SearchType property to 1 to search based on the

cell's value.

Tip

You can search a sheet, and you can also directly search a sheet cell range.

Gerrit Jasper pointed out that the SimpleSheetSearch function works equally well on a range as it does on a sheet – kind of embarrassing that I did not noticed this when I wrote the rest of this section. The following example demonstrates this capability by searching a specified range.

```
Sub SearchARange
    REM Author: Andrew Pitonyak
    REM Author: Martin Wilkinson
    Dim oSheet
    Dim oRange
    Dim oFoundCell
    Dim sSearchString$      : sSearchString = "ae"
    Dim bWholeWord As Boolean : bWholeWord = False
    Dim oConv

    ' Use to get a string representation of the cell address
    oConv =
ThisComponent.CreateInstance("com.sun.star.table.CellAddressConversion")

    ' Get the first sheet, or, use the next line instead
    ' to get the sheet named "Work1".
    oSheet = ThisComponent.getSheets().getByIndex(0)
    'oSheet = ThisComponent.getSheets().getByname("Work1")

    oRange = oSheet.getCellRangeByName("A1:H11")
    oFoundCell = SimpleSheetSearch(sSearchString, oRange, False)
    If NOT IsNull(oFoundCell) and NOT IsEmpty(oFoundCell) Then
        oConv.Address = oFoundCell.CellAddress
        MsgBox ("Found: "" & oFoundCell.getString & _
                "" in Cell: " & oConv.UserInterfaceRepresentation )
    Else
        Print "Text not found"
    End If
End Sub
```

6.24. Print a Calc range

The official method for printing a range in a Calc document is to set the print area for each sheet and then use the print method on the document. Although XPrintAreas interface, which is used to set the print areas, has been marked as deprecated for a while, it is still the officially sanctioned method for printing sections of a Calc document (Thanks Niklas Nebel).

David French <dfrench(at)xtra.co.nz>, a moderator on the oooforum, pointed out that when a Calc document prints, all print areas in all sheets are printed. In other words, when you set a print area, you should first clear the print areas from all sheets. Cor Nouws produced the

solution shown below (based on a solution by David French). He mentions that he obtains sheets using `getByName()` because `getByIndex()` did not work well when hidden sheets were used.

```
'Author: Cor Nouws <cno@nouenoff.nl>
'-----
Sub PrintSelectedArea (sSht$, nStC&, nStR&, nEndC&, nEndR&)
'-----

    Dim selArea(0) as new com.sun.star.table.CellRangeAddress
    Dim oDoc as object
    Dim oSheet as object
    Dim oSheets
    Dim i%
    oDoc = ThisComponent
    oSheets = ThisComponent.Sheets
    For i = 0 to oSheets.getCount() - 1
        oSheet = ThisComponent.Sheets.getByIndex(i)
        oSheet.setPrintareas(array())
    Next

    selArea(0).StartColumn = nStC
    selArea(0).StartRow = nStR
    selArea(0).EndColumn = nEndC
    selArea(0).EndRow = nEndR
    oSheet=ThisComponent.Sheets.getByName(sSht)
    oSheet.setPrintareas(selArea())
    oDoc.Print(Array())
End Sub
```

6.25. Is a cell merged?

Note: my book has some coverage on page 342. That said, When cells are merged, the merged area acts as a single cell as identified by the upper left hand corner of the merged range. The entire range will report as merged using the `isMerged()` function. The individual cell will also report as merged. The individual cell or the originally merged range can be used to set the merged property to false. I do not know off hand how to quickly find cells that are merged.

```
Sub MergeTest
    Dim oCell 'Holds a cell temporarily
    Dim oRange 'The primary range
    Dim oSheet 'The fourth sheet

    oSheet = ThisComponent.getSheets().getByIndex(0)
    oRange = oSheet.getCellRangeByName("B2:D7")
    oRange.merge(True)
    Print "Range merged = " & oRange.isMerged() ' True
```

```

REM Now obtain a cell that was merged
REM and I can do it!
oCell = oSheet.getCellByPosition(2, 3) 'C4
Print "Cell C4 merged = " & oCell.isMerged() ' False
oCell = oSheet.getCellByPosition(1, 1) 'B2
Print "Cell B2 merged = " & oCell.isMerged() ' True
oCell.merge(False)
End Sub

```

6.26. Write your own Calc functions

This is just a brief introduction to writing your own Calc functions in Basic. I am writing many things from memory without testing, so be certain to inform me of any errors.

6.26.1. User defined Calc functions

Be certain to check the OOO documentation site for more information on this topic; I wrote a chapter on macros for Calc. Also, many thanks to Rob Gray (robberbaron@optusnet.com.au) for material in this section.

If you define a function in Basic, you can call it from Calc. For example, I created a Calc document, and added the NumberTwo macro in the Module1 in the Standard library of the Calc document. I added the function call =NumberTwo() in a cell and it displayed the value 2.

Listing 6.32: *Very simple function that returns 2.0.*

```

Function NumberTwo() As Double
    NumberTwo() = 2.0
End Function

```

[Thanks to Rob Gray (robberbaron@optusnet.com.au)] For a User Defined Functions (UDF) to be available in Calc, the library must be loaded. Only the Standard Library of the current document and My Macros are loaded/available at startup. A UDF can be stored in any library but that library must be loaded or error #NAME? results. You can load a macro using the macro organizer (**Tools > Macros > Organize Macros > OpenOffice.org Basic**).

Distributions using the Novell changes recognizes VBA code and retains the correct UDF calls in worksheets. For example, =Test1(34) is retained rather than converting it to =(test1;34); although code changes are usually required, this is much easier.

Rob Gray has the Document Open or Start Application event call BWStart in Listing 6.33. BWStart loads the Tools library, and two of his own his macro libraries on startup.

Listing 6.33: *Very simple function that returns 2.0.*

```

Sub BWStart
    Dim oLibs As Object
    oLibs = GlobalScope.BasicLibraries
    'load useful / required libraries

```

```

LibName="Tools"      'from OO Macros & Dialogs collection
If oLibs.HasByName (LibName) AND (Not oLibs.isLibraryLoaded(LibName)) Then
    oLibs.LoadLibrary(LibName)
End If
LibName="VBA_Compat" 'from My Macros & Dialogs
If oLibs.HasByName (LibName) AND (Not oLibs.isLibraryLoaded(LibName)) Then
    oLibs.LoadLibrary(LibName)
End If

LibName="RecordedMacros" 'from current document
If oLibs.HasByName (LibName) AND (Not oLibs.isLibraryLoaded(LibName)) Then
    oLibs.LoadLibrary(LibName)
End If
End sub

```

6.26.2. Evaluating the argument

The arguments to a user defined Calc function are determined by your own needs. It is your responsibility to either use the correct arguments, or to test for them in your code. For example, I can define a function that accepts a single numeric value, but then I will have a problem if multiple values are passed as an argument. Inspect the arguments to determine what action to take.

Listing 6.34: *Inspect the argument as a Calc function.*

```

Function EvalArgs(Optional x) As String
    Dim i As Integer
    Dim s As String
    If IsMissing(x) Then
        s = "No Argument"
    ElseIf NOT IsArray(x) Then
        s = "One argument of type " & TypeName(x)
    Else
        s = "Array with bounds x(" & _
            LBound(x, 1) & " To " & UBound(x, 1) & ", " & _
            LBound(x, 2) & " To " & UBound(x, 2) & ")"
    End If
    EvalArgs() = s
End Function

```

A little experimentation demonstrates that the call =EVALARGS(A2:B4; A4:C7) returns the string “Array with bounds x(1 To 3, 1 To 2)”. In other words, access the argument as a two dimensional array.

The single argument x corresponds to the range A2:B4. The second range is completely ignored. Add a second optional argument to handle the second range.

6.26.3. What is the return type

It is not possible to know the expected return type. Usually, this is not a problem. For most types, Calc can properly handle the value. If the function is called as an array function, however, the function must return a two dimensional array.

Listing 6.35: Function, callable in an array context.

```
Function ArrayCalc(a as Single)
  Dim x(0 To 4, 0 To 4) As Single
  Dim nC As Integer
  Dim n As Integer
  Dim m As Integer
  nC = 2
  For n = 0 To 4
    For m = 0 To 4
      x(n, m) = nC * a
      nC = nC + 1
    Next
  Next
  ArrayCalc = x()
End Function
```

I placed my cursor into cell B8 and entered =ArrayCalc(3) and then I pressed the keys *Shift+Ctrl+Enter*, which tells Calc to enter an array formula. If an array formula is not used, then the value in the upper left hand corner is used. The following values then appear in cells B8:F12.

	B	C	D	E	F
8	6	9	12	15	18
9	21	24	27	30	33
10	36	39	42	45	48
11	51	54	57	60	63
12	66	69	72	75	78

The number of cells used to display the returned data was determined by the size of the returned array. The size of the returned array is not determined by the number of rows and columns that need to be filled. It is not possible to evaluate the calling context to know how many rows and columns should be returned; at least I do not believe so.

6.26.4. Do not modify other cells in the sheet

If a function is called from a cell in Sheet1, then the called function can not change any other cell in sheet1. The changes will be ignored.

6.26.5. Add digits in a number

How can I add the digits in a number? The first solution was rejected because it is a new function.

```
number = 12890000
while number > 0
    total = total + number MOD 10
    number = (number - (number MOD 10)) / 10
wend
msgbox total
```

The next solution by Ken Johnson in Sydney, Australia, uses an array formula. This assumes that the number in question is in Cell A1 Enter the formula and then use Ctrl+Shift+Enter (to tell Calc that this is an array formula).

```
=SUM(VALUE(MID(A1;ROW(INDIRECT("A1:A"&LEN(A1)) );1)))
```

If the number is in a different cell, change the first and third A1s. The A1 in "A1:A" inside the INDIRECT function can stay as is because it generates an array of values from 1 to the number of digits in the number having its digits summed.

My primary purpose for mentioning this is to demonstrate that a function is not always the best solution.

6.27. Add a chart

Although OOO contains extensive charting capabilities, a simple example is shown to demonstrate how to create a chart.

Listing 6.36: *Insert a simple chart.*

```
Sub CreateChart
    Dim oSheet      'Sheet containing the chart
    Dim oRect       'How big is the chart
    Dim oCharts     'Charts in the sheet
    Dim oChart      'Created chart
    Dim oAddress    'Address of data to plot
    Dim sName$      'Chart name
    Dim oChartDoc   'Embedded chart object
    Dim oTitle      'Chart title object
    Dim oDiagram    'Inserted diagram (data).
    Dim sDataRng$   'Where is the data

    sName = "ADP_Chart"
    sDataRng = "A1:D6"

    oSheet = ThisComponent.sheets(0)
    oAddress = oSheet.getCellRangeByName( sDataRng ).getRangeAddress()
    oCharts = oSheet.getCharts()
```

```

If NOT oCharts.hasByName(sName) Then
    oRect      = createObject("com.sun.star.awt.Rectangle")
    oRect.X     = 10000
    oRect.Y     = 1000
    oRect.width = 10000
    oRect.Height= 10000

    ' The rectangle identifies the dimensions in 1/100 mm.
    ' The address is the location of the data.
    ' True indicates that column headings should be used.
    ' False indicates that Row headings should not be used.
    oCharts.addNewByName(sName, oRect, Array(oAddress), True, False)
End If

oChart = oCharts.getByName( sName )
oChart.setRanges(Array(oAddress))
oChartDoc = oChart.getEmbeddedObject()
'oChartDoc.attachData(oAddress)
oTitle = oChartDoc.getTitle()
oTitle.String = "Andy - " & Now

' Create a diagram.
oDiagram = oChartDoc.createInstance( "com.sun.star.chart.LineDiagram" )
oChartDoc.setDiagram( oDiagram )
oDiagram = oChartDoc.getDiagram()
oDiagram.DataCaption = com.sun.star.chart.ChartDataCaption.VALUE
oDiagram.DataRowSource = com.sun.star.chart.ChartDataRowSource.COLUMNS
End Sub

```

6.28. Use FunctionAccess to call Calc Functions

Most functions are easy to call, create a FunctionAccess method. The first argument is the function name, and the second argument is an array containing the arguments. I will start with a "difficult" example - VLookup.

```

Dim x, oFuncAcc
x = Array(Array("?", "1", "2"), Array("X", "21", "22"))
oFuncAcc = createunodoservice("com.sun.star.sheet.FunctionAccess")
Print oFuncAcc.callFunction("VLookup", array("?", x, 2, 0))
Print oFuncAcc.callFunction("VLookup", array("X", x, 2, 0))

```

I can also pass a cell range, or the data array from a cell range.

```

Dim oSheet, oRange
oSheet = ThisComponent.Sheets.getByName("Sheet1")
oRange = oSheet.getCellRangeByPosition(0, 1, 9, 200)
Print oFuncAcc.callFunction("VLookup", array("x", oRange.getDataArray(), 2, 0))
Print oFuncAcc.callFunction("VLookup", array("x", oRange, 2, 0))

```


7. Writer Macros

7.1. Selected Text, What Is It?

Selected text is essentially a text range, nothing more. After a selection is obtained, it is possible to get the text [getString()] and set the text [setString()]. Although strings are limited to 64K in size, selections are not. There are some instances, therefore, when the getString() and setString() methods have results that are not understood by myself. It is, therefore, probably better to use a cursor to traverse the selected text and then use the insertString() and insertControlCharacter() method of the text object, which supports the XText interface. The documentation specifically states that the following white space characters are supported by insertString() method: blank, tab, cr (which will insert a paragraph break), and lf (which will insert a line break).

Text may be manually selected such that the cursor is on either the left or the right of the selection. A selection has both a start and an end point and you can not know ahead of time which is on the left and which is on the right of the selected text. A method is shown below to address this problem.

See Also:

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/TextRange.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XTextRange.html>

7.1.1. Is the cursor in a text table?

It is a common misconception that you use the selected text functions to determine if the cursor is located inside of a text table. A Writer document's controller returns a view cursor that identifies the location of the cursor. The view cursor contains two properties of interest, TextTable, and Cell – each text cursor also contains the properties Textfield, TextFrame, and TextSection. Use IsNull() to check these properties to determine if the cursor is contained in a table. Be certain to experiment with multiple cells and areas selected.

Listing 7.1: Is the cursor in a text table?

```
Sub TableStuff
    Dim oVCurs      'The view cursor
    Dim oTable      'The text table that contains the text cursor.
    Dim oCurCell    'The text table cell that contains the text cursor.

    oVCurs = ThisComponent.getCurrentController().getViewCursor()

    If IsEmpty(oVCurs.TextTable) Then
        Print "The cursor is NOT in a table"
    Else
        oTable = oVCurs.TextTable
        oCurCell = oVCurs.Cell
        Print "The cursor is in cell " & oCurCell.CellName
    End If
End Sub
```

7.1.2. Can I check the current selection for a TextTable or Cell?

[Andy notes: this section is premature so be certain to read the rest of section 7.1]

If I place the cursor inside of a text table and I do not select the entire cell, then the current selection is a TextRanges object. The selected text from the cell is represented by a TextRange object. The TextRange object has a TextTable property and a Cell property. These two properties are not empty if the text range is contained in a text table cell.

Normally, I can select multiple pieces of text. When I select one or more cells in a text table, however, I can have only one selection and the returned current selection is a TextTableCursor. The following macro illustrates these possibilities.

Listing 7.2: Does the current selection contain a text table?

```
Sub IsACellSelected
    Dim oSels          'All of the selections
    Dim oSel           'A single selection
    Dim i As Integer
    Dim sTextTableCursor$

    sTextTableCursor$ = "com.sun.star.text.TextTableCursor"

    oSels = ThisComponent.getCurrentController().getSelection()
    If oSels.supportsService("com.sun.star.text.TextRanges") Then
        For i=0 to oSels.getCount()-1
            oSel = oSels.getByIndex(i)
            If oSel.supportsService("com.sun.star.text.TextRange") Then
                If ( Not IsEmpty(oSel.TextTable) ) Then
                    Print "The text table property is NOT empty"
                End If
                If ( Not IsEmpty(oSel.Cell) ) Then
                    Print "The Cell property is NOT empty"
                End If
            End If
        Next
    ElseIf oSels.supportsService(sTextTableCursor) Then
        REM At least one entire cell is selected
        Print oSels.getRangeName()
    End If
End Sub
```

7.2. Text Cursors, What Are They?

A regular TextCursor is an invisible cursor, which is independent of the view cursor. You can have more than one at a time and you can move them around without affecting the view cursor. The view cursor is also a text cursor, that also supports the XTextViewCursor interface. There is only one view cursor and it is visible on the screen.

A `TextCursor` is a `TextRange` which can be moved within a `Text` object. The standard movements include `goLeft`, `goRight`, `goUp`, and `goDown`. The first parameter is an integer indicating how many characters or lines to move. The second parameter is a boolean directing the selected text range to be expanded (`True`) or not. The value of `True` is returned as long as the move occurs. If a cursor has been selecting text by moving left and you now want it to start moving right, you probably want to use `oCursor.goRight(0, False)` to tell the cursor to start moving right and do not select text. This will leave no text selected.

A `TextCursor` has both a start and an end. If the start position is the same as the end position then no text is selected and the `IsCollapsed` property will be true.

A `TextCursor` implements interfaces that allow moves and recognizing positions specific to words, sentences, and paragraphs. This can save a lot of time.

Warning

`Cursor.gotoStart()` and `Cursor.gotoEnd()` go to the start and end of the document even if the cursor is created over a range.

See Also:

<http://api.openoffice.org/docs/common/ref/com/sun/star/view/XViewCursor.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/TextCursor.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XWordCursor.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XSentenceCursor.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XParagraphCursor.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/TextRange.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XTextRange.html>

7.2.1. You can not move a cursor to a `TextTable` anchor.

Text content is anchored into the text. The anchor is available using the method `getAnchor()`. Unfortunately, text cursors are not always compatible with an anchor. The following example demonstrates methods that fail with a text table anchor.

```
oAnchor = oTable.getAnchor()
oCurs.gotoRange(oAnchor.getStart(), False)           ' Error
oCurs = oAnchor.getText().createTextCursorByRange(oAnchor.getStart()) ' Error
oText.insertTextContent(oAnchor.getEnd(), oTable2, False) ' Error
```

I was recently asked how to delete an existing table and then create another table in its place. You can use a little trick and move the cursor to the table location using the current controller.

```
ThisComponent.getCurrentController().select(oTable)
```

The following macro demonstrates how to create a table and then replace it with another.

Listing 7.3: *Create a text table at the location of another.*

```
Sub Main
    Dim sName$
    Dim oTable
    Dim oAnchor
    Dim oCurs
    Dim oText

    oText = ThisComponent.getText()
    oTable = ThisComponent.CreateInstance("com.sun.star.text.TextTable")
    oTable.initialize(3, 3)
    'oTable.setName("wow")
    oCurs = ThisComponent.getCurrentController().getViewCursor()
    oText.insertTextContent(oCurs, oTable, False)
    oTable.setDataArray(Array(Array(1,2,3), Array(4,5,6), Array(7,8,9)))
    sName = oTable.getName()

    Print "Created table named " & sName
    oTable = ThisComponent.getTextTables().getByName(sName)
    oAnchor = oTable.getAnchor()

    REM I will now move the view cursor to the start of the document
    REM so that I can demonstrate that this works.
    oCurs = ThisComponent.getCurrentController().getViewCursor()
    oCurs.gotoStart(False)

    REM I would Love to be able to move the cursor to the anchor,
    REM but I can not create a crusor based on the anchor, move to
    REM the anchor, etc. So, I use a trick and let the controller
    REM move the view cursor to the table.
    REM Unfortunately, you can not move the cursor to the anchor...
    ThisComponent.getCurrentController().select(oTable)
    oTable.dispose()

    oTable = ThisComponent.CreateInstance("com.sun.star.text.TextTable")
    oTable.initialize(2, 2)
    oCurs = ThisComponent.getCurrentController().getViewCursor()
    oText.insertTextContent(oCurs, oTable, False)
    oTable.setDataArray(Array(Array(1, 2), Array(3, 4)))
    sName = oTable.getName()
    Print "Created table named " & sName
End Sub
```

You can use the idiosyncrasies of the API to move the text cursor immediately before a text table.

Listing 7.4: Move the text cursor BEFORE a text table.

```
Dim oTable
Dim oCurs

oTable = ThisComponent.getTextTables().getByIndex(2)

REM Move the cursor to the first row and column
ThisComponent.getCurrentController().select(oTable)
oCurs = ThisComponent.getCurrentController().getViewCursor()
oCurs.goLeft(1, False)
```

7.2.2. Inserting something before (or after) a text table.

If you get a text table as the first or last thing in your document, then you can not easily insert something before (or after) the table. Using the GUI, you can move the cursor into the start of the first cell and press enter to insert a leading new line. Using the API, you need to use `insertTextContentBefore` (or after).

Listing 7.5: Insert a new paragraph before a text table.

```
Sub InsertParBeforeTable
Dim oTable
Dim oText
Dim oPar
oTable = ThisComponent.getTextTables().getByIndex(0)
oText = ThisComponent.getText()
oCurs = oText.createTextCursor()
oPar = ThisComponent.createInstance("com.sun.star.text.Paragraph")
oText.insertTextContentBefore ( oPar, oTable )
End Sub
```

You can use this method for a paragraph, text section, or a text table.

7.2.3. You can move a cursor to a Bookmark anchor.

Bookmark anchors are compatible with a regular text cursor.

Listing 7.6: Select a bookmark anchor directly.

```
Dim oAnchor 'Bookmark anchor
Dim oCursor 'Cursor at the left most range.
Dim oMarks

oMarks = ThisComponent.getBookmarks()
oAnchor = oMarks.getByName("MyMark").getAnchor()
oCursor = ThisComponent.getCurrentController().getViewCursor()
oCursor.gotoRange(oAnchor, False)
```

You can, also compare a cursor to see if it is before or after an anchor; this fails with a `TextTable` anchor.

Listing 7.7: Compare the view cursor to an anchor.

```
Sub CompareViewCursorToAnchor()  
    Dim oAnchor 'Bookmark anchor  
    Dim oCursor 'Cursor at the left most range.  
    Dim oMarks  
  
    oMarks = ThisComponent.getBookmarks()  
    oAnchor = oMarks.getByName("MyMark").getAnchor()  
    oCursor = ThisComponent.getCurrentController().getViewCursor()  
  
    If NOT EqualUNOObjects(oCursor.getText(), oAnchor.getText()) Then  
        Print "The view cursor and the anchor use a different text object"  
        Exit Sub  
    End If  
  
    Dim oText, oEnd1, oEnd2  
    oText = oCursor.getText()  
    oEnd1 = oCursor.getEnd() : oEnd2 = oAnchor.getEnd()  
    If oText.compareRegionStarts(oEnd1, oEnd2) >= 0 Then  
        Print "Cursor END is Left of the anchor end"  
    Else  
        Print "Cursor END is Right of the anchor end"  
    End If  
End Sub
```

You need to experiment with the different anchor types to see which play well with cursors.

7.2.4. Insert Text At Bookmark

Listing 7.8: Insert text at a bookmark.

```
oBookMark = oDoc.getBookmarks().getByName("<yourBookmarkName>")  
oBookMark.getAnchor.setString("What you want to insert")
```

7.3. Andrew's Selected Text Framework

Most problems using selected text look the same at an abstract level.

```
If nothing is selected then  
    do work on entire document  
else  
    for each selected area  
        do work on selected area
```

The difficult part that will change each time is writing a worker macro that will iterate over a selection or between two cursors.

7.3.1. Is Text Selected?

The documentation states that if there is no current controller, then `GetCurrentSelection()` will return a null rather than the selections; perhaps OOo is running in headless mode.

If the text cursor is sitting in the document with nothing selected, this is considered an “empty” selection. As such, the selection count should never be zero. For my purposes, therefore, no text is selected when I have one selection of zero length. One might argue that if all of my selections are of zero length, then nothing is selected. I have seen examples where a zero length selection is determined as follows:

Listing 7.9: Wrong way to check for an empty selection.

```
If Len(oSel.GetString()) = 0 Then nothing is selected
```

Selected text can be longer than 64K characters, but a string can not contain more than 64K characters. Also, this is not efficient because the selected text must be converted to a single string. The better solution is to create a text cursor from the selected range and then check to see if the start and end points are the same.

Listing 7.10: See if a selection is collapsed.

```
oCursor = oSel.GetText().CreateTextCursorByRange(oSel)
If oCursor.IsCollapsed() Then nothing is selected
```

Here is the function that will perform the entire check. The code in Listing 7.10 uses the text object from the document. If the selection is not in the document's primary text object then this will fail. Listing 7.11 obtains the text object from the selection object, which is safer.

Listing 7.11: Is text selected?

```
Function IsAnythingSelected(oDoc As Object) As Boolean
    Dim oSels 'All of the selections
    Dim oSel 'A single selection
    Dim oCursor 'A temporary cursor

    IsAnythingSelected = False
    If IsNull(oDoc) Then Exit Function
    ' The current selection in the current controller.
    'If there is no current controller, it returns NULL.
    oSels = oDoc.GetCurrentSelection()
    If IsNull(oSels) Then Exit Function

    REM I have never seen a selection count of zero
    If oSels.GetCount() = 0 Then Exit Function

    REM If there are multiple selections, then assume
    REM something is selected
    If oSels.GetCount() > 1 Then
        IsAnythingSelected = True
    Else
```



```

REM If only one thing is selected, however, then check to see
REM if the selection is collapsed. In other words, see if the
REM end location is the same as the starting location.
REM Notice that I use the text object from the selection object
REM because it is safer than assuming that it is the same as the
REM documents text object.
oSel = oSels.getByIndex(0)
oCursor = oSel.getText().CreateTextCursorByRange(oSel)
If Not oCursor.IsCollapsed() Then IsAnythingSelected = True
End If
End Function

```

7.3.2. How To Get A Selection

Obtaining a selection is complicated because it is possible to have multiple non-contiguous selections. Some selections are empty and some are not. Code written to handle text selection should handle all of these cases. The following example iterates through all of the selected sections printing them.

Listing 7.12: *It is possible to have multiple simultaneous selections.*

```

'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub MultipleTextSelectionExample
    Dim oSels As Object, oSel As Object
    Dim lSelCount As Long, lWhichSelection As Long

    ' The current selection in the current controller.
    'If there is no current controller, it returns NULL.
    oSels = ThisComponent.getCurrentSelection()
    If Not IsNull(oSels) Then
        lSelCount = oSels.getCount()
        For lWhichSelection = 0 To lSelCount - 1
            oSel = oSels.getByIndex(lWhichSelection)
            MsgBox oSel.getString()
        Next
    End If
End Sub

```

See Also:

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XTextRange.html>

7.3.3. Selected Text, Which End Is Which

Selections are essentially text ranges with a start and an end. Although selections have both a start and an end, which side of the text is which is determined by the selection method. The text object provides methods to compare starting and ending positions of text ranges. The

method “short compareRegionStarts (XTextRange R1, XTextRange R2)” returns 1 if R1 starts before R2 , 0 if R1 starts at the same position as R2 and -1 if R1 starts after R2. The method “short compareRegionEnds (XTextRange R1, XTextRange R2)” returns 1, if R1 ends before R2 , 0, if R1 ends at the same position as R2 and -1, if R1 ends behind R2. I use the following two methods to find the leftmost and rightmost cursor position of selected text.

Listing 7.13: Determine if the start or end comes first.

```

' *****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'oSel is a text selection or cursor range
Function GetLeftMostCursor(oSel As Object) As Object
    Dim oRange 'Left most range.
    Dim oCursor 'Cursor at the left most range.

    If oSel.GetText().compareRegionStarts(oSel.getEnd(), oSel) >= 0 Then
        oRange = oSel.getEnd()
    Else
        oRange = oSel.getStart()
    End If
    oCursor = oSel.GetText().CreateTextCursorByRange(oRange)
    oCursor.goRight(0, False)
    GetLeftMostCursor = oCursor
End Function

' *****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'oSel is a text selection or cursor range
Function GetRightMostCursor(oSel As Object) As Object
    Dim oRange 'Right most range.
    Dim oCursor 'Cursor at the right most range.

    If oSel.GetText().compareRegionStarts(oSel.getEnd(), oSel) >= 0 Then
        oRange = oSel.getStart()
    Else
        oRange = oSel.getEnd()
    End If
    oCursor = oSel.GetText().CreateTextCursorByRange(oRange)
    oCursor.goLeft(0, False)
    GetRightMostCursor = oCursor
End Function

```

Tip

I modified these sections to obtain the text object from the selection objects rather than using the document's text object.

See Also:

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XTextCursor.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XSimpleText.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XTextRangeCompare.html>

7.3.4. The Selected Text Framework Macro

It took me a long time to understand how to iterate over selected text using cursors so I have written many macros that do things in what I consider the wrong way. I now use a high level framework to do this. The idea is that if no text is selected, then it asks if the macro should be run against the entire document. If the answer is yes, then a cursor is created at the start and the end of the document and then the worker macro is called. If text is selected, then each selection is retrieved, a cursor is obtained at the start and end of selection, and then the worker macro is called for each of these selections.

7.3.4.1. The Rejected Framework

I ultimately rejected the framework that follows because it is just too long and cumbersome to repeat every time that I wanted to iterate over text. It is, however, tenable. You may prefer this framework and choose to use it.

Listing 7.14: A cumbersome selected text framework that works.

```
Sub IterateOverSelectedTextFramework
    Dim oSels As Object, oSel As Object, oText As Object
    Dim lSelCount As Long, lWhichSelection As Long
    Dim oLCurs As Object, oRCurs As Object

    oText = ThisComponent.Text
    If Not IsAnythingSelected(ThisComponent) Then
        Dim i%
        i% = MsgBox("No text selected!" + Chr(13) + _
            "Call worker for the ENTIRE document?", _
            1 OR 32 OR 256, "Warning")
        If i% <> 1 Then Exit Sub
        oLCurs = oText.createTextCursor()
        oLCurs.gotoStart(False)
        oRCurs = oText.createTextCursor()
        oRCurs.gotoEnd(False)
        CallYourWorkerMacroHere(oLCurs, oRCurs, oText)
    Else
        oSels = ThisComponent.getCurrentSelection()
        lSelCount = oSels.getCount()
        For lWhichSelection = 0 To lSelCount - 1
            oSel = oSels.getByIndex(lWhichSelection)
            'If I want to know if NO text is selected, I could
            'do the following:
```

```

        'oLCurs = oText.CreateTextCursorByRange(oSel)
        'If oLCurs.isCollapsed() Then ...
        oLCurs = GetLeftMostCursor(oSel, oText)
        oRCurs = GetRightMostCursor(oSel, oText)
        CallYourWorkerMacroHere(oLCurs, oRCurs, oText)
    Next
End If
End Sub

```

7.3.4.2. The Accepted Framework

I opted to create the framework that follows. It returns a two dimensional array of start and end cursors over which to iterate. It allows for a very minimal code base to be used to iterate over selected text or the entire document.

Listing 7.15: Create cursors around the selected ranges.

```

'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'sPrompt : how to ask if should iterate over the entire text
'oCurs() : Has the return cursors
'Returns true if should iterate and false if should not
Function CreateSelectedTextIterator(oDoc, sPrompt$, oCurs()) As Boolean
    Dim lSelCount As Long          'Number of selected sections.
    Dim lWhichSelection As Long    'Current selection item.

    Dim oSels 'All of the selections
    Dim oSel  'A single selection.
    Dim oLCurs 'Cursor to the left of the current selection.
    Dim oRCurs 'Cursor to the right of the current selection.

    CreateSelectedTextIterator = True
    If Not IsAnythingSelected(ThisComponent) Then
        Dim i%
        i% = MsgBox("No text selected!" + Chr(13) + sPrompt, _
            1 OR 32 OR 256, "Warning")
        If i% = 1 Then
            oLCurs = oDoc.getText().createTextCursor()
            oLCurs.gotoStart(False)
            oRCurs = oDoc.getText().createTextCursor()
            oRCurs.gotoEnd(False)
            oCurs = DimArray(0, 1)
            oCurs(0, 0) = oLCurs
            oCurs(0, 1) = oRCurs
        Else
            oCurs = DimArray()
            CreateSelectedTextIterator = False
        End If
    End If
End Function

```

```

Else
    oSels = ThisComponent.getCurrentSelection()
    lSelCount = oSels.getCount()
    oCurs = DimArray(lSelCount - 1, 1)
    For lWhichSelection = 0 To lSelCount - 1
        oSel = oSels.getByIndex(lWhichSelection)
        REM If I want to know if NO text is selected, I could
        REM do the following:
        REM oLCurs = oSel.getText().CreateTextCursorByRange(oSel)
        REM If oLCurs.isCollapsed() Then ...
        oLCurs = GetLeftMostCursor(oSel)
        oRCurs = GetRightMostCursor(oSel)
        oCurs(lWhichSelection, 0) = oLCurs
        oCurs(lWhichSelection, 1) = oRCurs
    Next
End If
End Function

```

7.3.4.3. The Main Worker

This is an example that then calls a worker routine.

Listing 7.16: *Call the routine PrintEachCharacterWorker for each selected range.*

```

Sub PrintExample
    Dim oCurs(), i%
    If Not CreateSelectedTextIterator(ThisComponent, _
        "Print characters for the entire document?", oCurs()) Then Exit Sub
    For i% = LBound(oCurs()) To UBound(oCurs())
        PrintEachCharacterWorker(oCurs(i%, 0), oCurs(i%, 1))
    Next i%
End Sub

Sub PrintEachCharacterWorker(oLCurs As Object, oRCurs As Object)
    Dim oText
    oText = oLCurs.getText()
    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oText) Then Exit Sub
    If oText.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Sub
    oLCurs.goRight(0, False)
    Do While oLCurs.goRight(1, True) AND _
        oText.compareRegionEnds(oLCurs, oRCurs) >= 0
        Print "Character = '" & oLCurs.getString() & "'"
        REM This will cause the currently selected text to become
        REM no longer selected
        oLCurs.goRight(0, False)
    Loop
End Sub

```

7.3.5. Counting Sentences

I threw this together quickly and with little thought. Use at your own risk! I found some bugs in the sentence cursor while writing my book, check my Macro book page 286 for more information.

Listing 7.17: Count sentences using a sentence cursor.

```
REM This will probably fail if there text tables because the sentence
REM cursor will not be able to enter the table, but I have not checked
REM this to verify.
Sub CountSentences
    Dim oCursor          'A text cursor.
    Dim oSentenceCursor  'A text cursor.
    Dim oText
    Dim i

    oText = ThisComponent.Text
    oCursor = oText.CreateTextCursor()
    oSentenceCursor = oText.CreateTextCursor()

    'Move the cursor to the start of the document
    oCursor.GoToStart(False)
    Do While oCursor.gotoNextParagraph(True)
        'At this point, you have the entire paragraph highlighted
        oSentenceCursor.gotoRange(oCursor.getStart(), False)
        Do While oSentenceCursor.gotoNextSentence(True) AND_
            oText.compareRegionEnds(oSentenceCursor, oCursor) >= 0
            oSentenceCursor.goRight(0, False)
            i = i + 1
        Loop
        oCursor.goRight(0, False)
    Loop
    MsgBox i, 0, "Number of Sentences"
End Sub
```

7.3.6. Remove Empty Spaces And Lines, A Larger Example

This set of macros replaces all runs of white space characters with a single white space character. It is easily modifiable to delete different types of white space. The different types of spaces are ordered by importance so if you have a regular space followed by a new paragraph, the new paragraph will stay and the single space will be removed. This will cause leading and trailing white space to be removed from a line.

7.3.6.1. Define “White Space”

In solving this problem, my first task was to determine what characters are white space characters. You can trivially change the definition of white space to ignore certain characters.

'Usually, this is done with an array lookup which would probably be
'faster, but I do not know how to use static initializers in .

```
Function IsWhiteSpace(iChar As Integer) As Boolean
    Select Case iChar
        Case 9, 10, 13, 32, 160
            IsWhiteSpace = True
        Case Else
            IsWhiteSpace = False
    End Select
End Function
```

7.3.6.2. Rank Characters For Deletion

Next, I needed to define what to remove and what to leave. I opted to do this with the following routine.

```
'-1 means delete the previous character
' 0 means ignore this character
' 1 means delete this character
' Rank from highest to lowest is: 0, 13, 10, 9, 160, 32
Function RankChar(iPrevChar, iCurChar) As Integer
    If Not IsWhiteSpace(iCurChar) Then 'Char is not WS, ignore it
        RankChar = 0
    ElseIf iPrevChar = 0 Then 'Line start and char is WS
        RankChar = 1 ' so delete the character.
    ElseIf Not IsWhiteSpace(iPrevChar) Then 'Char is WS but previous was not
        RankChar = 0 ' so ignore the current character.
    ElseIf iPrevChar = 13 Then 'Previous char is highest ranked WS
        RankChar = 1 ' so delete the current character.
    ElseIf iCurChar = 13 Then 'Character is highest ranked WS
        RankChar = -1 ' so delete the previous character.
    ElseIf iPrevChar = 10 Then 'No new Paragraph. Check prev for new line
        RankChar = 1 'so delete the current character.
    ElseIf iCurChar = 10 Then 'No new Pars. Check current for new line.
        RankChar = -1 ' so delete the previous character.
    ElseIf iPrevChar = 9 Then 'No new Line! Check previous for tab
        RankChar = 1 ' so delete the current character.
    ElseIf iCurChar = 9 Then 'No new Line. Check current char for tab
        RankChar = -1 ' so delete the previous character.
    ElseIf iPrevChar = 160 Then 'No Tabs! Check previous char for hard space
        RankChar = 1 ' so delete the current character.
    ElseIf iCurChar = 160 Then 'No Tabs. Check current char for hard space
        RankChar = -1 'so delete the previous character.
    ElseIf iPrevChar = 32 Then 'No hard space, check previous for a space
        RankChar = 1 'so delete the current character.
    ElseIf iCurChar = 32 Then 'No hard spaces so check current for a space
        RankChar = -1 'so delete the previous character.
    Else 'Should probably not get here
```

```

        RankChar = 0                                'so simply ignore it!
    End If
End Function

```

7.3.6.3. The Standard Selected Text Iterator

This is the standard format to decide if work should be done on the entire document or just a portion.

Listing 7.18: *Remove runs of white space.*

```

'Remove all runs of empty space!
'If text is selected, then white space is only removed from the
'selected text.
Sub RemoveEmptySpace
    Dim oCurs() As Integer
    If Not CreateSelectedTextIterator(ThisComponent, _
        "ALL empty space will be removed from the ENTIRE document?", _
        oCurs()) Then Exit Sub
    For i% = LBOUND(oCurs()) To UBOUND(oCurs())
        RemoveEmptySpaceWorker (oCurs(i%, 0), oCurs(i%, 1))
    Next i%
End Sub

```

7.3.6.4. The Worker Macro

This is where the real work happens.

Listing 7.19: *Enumerate across the cursors and remove white space.*

```

Sub RemoveEmptySpaceWorker(oLCurs As Object, oRCurs As Object)
    Dim sParText As String, i As Integer
    Dim oText

    oText = oLCurs.GetText()
    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oText) Then Exit Sub
    If oText.CompareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Sub

    Dim iLastChar As Integer, iThisChar As Integer, iRank As Integer
    iLastChar = 0
    iThisChar = 0
    oLCurs.GoRight(0, False)
    Do While oLCurs.GoRight(1, True)
        iThisChar = Asc(oLCurs.GetString())
        i = oText.CompareRegionEnds(oLCurs, oRCurs)
        'If at the last character!
        'Then always remove white space
        If i = 0 Then
            If IsWhiteSpace(iThisChar) Then oLCurs.SetString("")
        End If
    End Do

```



```

End If
'If went past the end then get out
If i < 0 Then Exit Do
iRank = RankChar(iLastChar, iThisChar)
If iRank = 1 Then
    'I am about to delete this character.
    'I do not change iLastChar because it did not change!
    'Print "Deleting Current with " + iLastChar + " and " + iThisChar
    oLCurs.setString("")
ElseIf iRank = -1 Then
    'This will deselect the selected character and then select one
    'more to the left.
    oLCurs.goLeft(2, True)
    'Print "Deleting to the left with " + iLastChar + " and " + iThisChar
    oLCurs.setString("")
    oLCurs.goRight(1, False)
    iLastChar = iThisChar
Else
    oLCurs.goRight(0, False)
    iLastChar = iThisChar
End If
Loop
End Sub

```

7.3.7. Removing Empty Paragraphs, Yet Another Example

It is better to set up “AutoFormat” to remove blank paragraphs, then apply it to the document in question. Click on “Tools=>AutoCorrect/AutoFormat...” and then choose the “Options” tab. One of the options is “Remove Blank Paragraphs.” Make certain that this is checked. Now, you can auto format the document and all of the empty paragraphs are gone.

If you only want to remove selected empty paragraphs, then you will need to use a macro. If text is selected, then the empty paragraphs are removed from within the selected text. If no text is selected, then text is removed from the entire document. This first macro iterates through all of the selected text. If no text is selected, it creates a cursor at the beginning and the end of the document and then works on the entire document. The primary thing to see in this macro is how to traverse the text based on paragraphs. The removing empty space macro is the safer macro because it does not extract a string to work.

Listing 7.20: Remove empty paragraphs.

```

Sub RemoveEmptyParsWorker(oLCurs As Object, oRCurs As Object)
    Dim sParText As String, i As Integer
    Dim oText

    oText = oLCurs
    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oText) Then Exit Sub
    If oText.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Sub

```

```

oLCurs.goRight(0, False)
Do While oLCurs.gotoNextParagraph(TRUE) AND _
    oText.compareRegionEnds(oLCurs, oRCurs) > 0
    'Yes, I know, limited to 64K here
    'If we have one paragraph that is over 64K
    'Then I am in trouble!
    sParText = oLCurs.getString()
    i = Len(sParText)
    'We do not have short circuit logical. Drat!
    Do While i > 0
        If (Mid(sParText,i,1) = Chr(10)) OR (Mid(sParText,i,1) = Chr(13)) Then
            i = i - 1
        Else
            i = -1
        End If
    Loop
    If i = 0 Then
        oLCurs.setString("")
    Else
        oLCurs.goLeft(0,FALSE)
    End If
loop
End Sub

```

7.3.8. Selected Text, Timing Considerations And Counting Words

Anyone who has studied algorithms will tell you that a better algorithm is usually better than a faster computer. When I first wrote the macros that manipulate blank lines and spaces, I wrote them using strings. This introduced the possibility of losing formatting information and failure when the strings exceeded 64K in size. I then wrote the macros using cursors and I received complaints that they were too slow. The question arises, is there a better way?

7.3.8.1. Searching Selected Text To Count Words

Andrew Brown, the maintainer of <http://www.darwinwars.com> (contains useful macro information), asked about performing searches inside of a selected region. ??See the section on searching selected text.??

I found out that this was used to count words in a document and that it was very slow; too slow.

7.3.8.2. Using Strings To Count Words

The existing code counted the number of spaces in the selected region and used that to determine the number of words. I then wrote my own version that was a bit more general, slightly faster, and produced the correct answer.

Listing 7.21: Count words in a safe but slow way.

```
Function ADPWordCountStrings(vDoc) As String
    REM Place what ever characters you want as word separators here!
    Dim sSeps$
    sSeps = Chr$(9) & Chr$(13) & Chr$(10) & " ,;."
    Dim bSeps(256) As Boolean, i As Long
    For i = LBound(bSeps()) To UBound(bSeps())
        bSeps(i) = False
    Next
    For i = 1 To Len(sSeps)
        bSeps(Asc(Mid(sSeps, i, 1))) = True
    Next

    Dim nSelChars As Long, nSelwords As Long
    Dim nSel%, nNonEmptySel%, j As Long, s$
    Dim vSelections, vSel, oText, oCursor
    ' The current selection in the current controller.
    'If there is no current controller, it returns NULL.
    vSelections = vDoc.GetCurrentSelection()
    If IsNull(vSelections) Then
        nSel = 0
    Else
        nSel = vSelections.getCount()
    End If
    nNonEmptySel = 0
    Dim lTemp As Long, bBetweenWords As Boolean, bIsSep As Boolean
    On Local Error Goto BadOutOfRange
    Do While nSel > 0
        nSel = nSel - 1
        s = vSelections.GetByIndex(nSel).getString()
        REM See if this is an empty selection
        lTemp = Len(s)
        If lTemp > 0 Then
            nSelChars = nSelChars + lTemp
            nNonEmptySel = nNonEmptySel + 1
            REM Does this start on a word?
            If bSeps(Asc(Mid(s, 1, 1))) Then
                bBetweenWords = True
            Else
                bBetweenWords = False
                nSelWords = nSelWords + 1
            End If
            For j = 2 To lTemp
                bIsSep = bSeps(Asc(Mid(s, j, 1)))
                If bBetweenWords <> bIsSep Then
                    If bBetweenWords Then
                        REM Only count a new word if I was between words
```

```

        REM and I am no longer between words!
        bBetweenWords = False
        nSelWords = nSelWords + 1
    Else
        bBetweenWords = True
    End If
End If
Next
End If
Loop
On Local Error Goto 0

Dim nAllChars As Long, nAllWords As Long, nAllPars As Long
' access document statistics
nAllChars = vDoc.CharacterCount
nAllWords = vDoc.WordCount
nAllPars = vDoc.ParagraphCount

Dim sRes$
sRes = "Document count:" & chr(13) & nAllWords & " words." & _
    chr(13) & "(" & nAllChars & " Chars." & chr(13) & nAllPars & _
    " Paragraphs.)" & chr(13) & chr(13)
If nNonEmptySel > 0 Then
    sRes = sRes & "Selected text count:" & chr(13) & nSelWords & _
        " words" & chr(13) & "(" & nSelChars & " chars)" & _
        chr(13) & "In " & str(nNonEmptySel) & " selection"
    If nNonEmptySel > 1 Then sRes = sRes & "s"
    sRes=sRes & "." & chr(13) & chr(13) & "Document minus selected:" & _
        chr(13)& str(nAllWords-nSelWords) & " words."
End If
'MsgBox(sRes,64,"ADP Word Count")
ADPWordCountStrings = sRes
Exit Function

BadOutOfRange:
    bIsSep = False
    Resume Next
End Function

```

Each selected range is extracted as a string. This fails if the selected text is greater than 64K in size. The ASCII value of each character is checked to see if it is considered a word separator. This is done by an array lookup. This was efficient but failed if there was a special character that had an ASCII value larger than the array so an error handling routine is used. Special handling is done so that the correct values are obtained with various selections. This took about 2.7 seconds to check 8000 words.

7.3.8.3. Using A Character Cursor To Count Words

In an attempt to avoid the 64K limit, I wrote a version the used cursors to traverse the text one character at a time. This version took 47 seconds to check the same 8000 words. This uses the same algorithm as the string method, but the overhead of using a cursor to find each character is prohibitive.

Listing 7.22: Count words using a character cursor.

```
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Function ADPWordCountCharCursor(vDoc) As String
    Dim oCurs(), i%, lNumWords As Long
    lNumWords = 0
    If Not CreateSelectedTextIterator(vDoc, _
        "Count Words in the entire document?", oCurs()) Then Exit Function
    For i% = LBound(oCurs()) To UBound(oCurs())
        lNumWords = lNumWords + _
            WordCountCharCursor(oCurs(i%, 0), oCurs(i%, 1), vDoc.Text)
    Next
    ADPWordCountCharCursor = "Total Words = " & lNumWords
End Function
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Function WordCountCharCursor(oLCurs, oRCurs, oText)
    Dim lNumWords As Long
    lNumWords = 0
    WordCountCharCursor = lNumWords
    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oText) Then Exit Function
    If oText.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Function

    Dim sSeps$
    sSeps = Chr$(9) & Chr$(13) & Chr$(10) & " ,;. "
    Dim bSeps(256) As Boolean, i As Long
    For i = LBound(bSeps()) To UBound(bSeps())
        bSeps(i) = False
    Next
    For i = 1 To Len(sSeps)
        bSeps(Asc(Mid(sSeps, i, 1))) = True
    Next

    On Local Error Goto BadOutOfRange
    Dim bBetweenWords As Boolean, bIsSep As Boolean
    oLCurs.goRight(0, False)
    oLCurs.goRight(1, True)
    REM Does this start on a word?
```

```

If bSeps(Asc(oLCurs.getString())) Then
    bBetweenWords = True
Else
    bBetweenWords = False
    lNumWords = lNumWords + 1
End If
oLCurs.goRight(0, False)

Do While oLCurs.goRight(1, True) AND _
    oText.compareRegionEnds(oLCurs, oRCurs) >= 0
    bIsSep = bSeps(Asc(oLCurs.getString()))
    If bBetweenWords <> bIsSep Then
        If bBetweenWords Then
            REM Only count a new word if I was between words
            REM and I am no longer between words!
            bBetweenWords = False
            lNumWords = lNumWords + 1
        Else
            bBetweenWords = True
        End If
    End If
    oLCurs.goRight(0, False)
Loop
WordCountCharCursor = lNumWords
Exit Function

BadOutOfRange:
    bIsSep = False
    Resume Next
End Function

```

7.3.8.4.

7.3.8.5. Using A Word Cursor To Count Words

This is the fastest method yet. This uses a word cursor and lets OOo figure out where words start and end. This will check the 8000 words in 1.7 seconds. This macro moves from word to word counting how many word breaks it finds. Because of this, the result may be off by one. I found bugs with word cursors while writing my book.

Listing 7.23: *count words using a word cursor.*

```

'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Function ADPWordCountWordCursor(vDoc) As String
    Dim oCurs(), i%, lNumWords As Long
    lNumWords = 0
    If Not CreateSelectedTextIterator(vDoc, _

```

```

    "Count Words in the entire document?", oCurs()) Then Exit Function
For i% = LBound(oCurs()) To UBound(oCurs())
    lNumWords = lNumWords + _
        WordCountWordCursor(oCurs(i%, 0), oCurs(i%, 1), vDoc.Text)
Next
ADPWordCountWordCursor = "Total Words = " & lNumWords
End Function
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Function WordCountWordCursor(oLCurs, oRCurs, oText)
    Dim lNumWords As Long
    lNumWords = 0
    WordCountWordCursor = lNumWords
    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oText) Then Exit Function
    If oText.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Function
    oLCurs.goRight(0, False)
    Do While oLCurs.gotoNextWord(False) AND _
        oText.compareRegionEnds(oLCurs, oRCurs) >= 0
        lNumWords = lNumWords + 1
    Loop
    WordCountWordCursor = lNumWords
End Function

```

7.3.8.6. Final Thoughts On Counting Words And Timing

If your solution to a problem is too slow, then perhaps there is another way. In OOo, cursors can move based on characters, words, or paragraphs. The cursor that you use makes a significant difference in the runtime.

If you want to count the number of words in a selected region, I recommend that you take a look at Andrew Brown's website, <http://www.darwinwars.com>, because he is actively working on counting words. In fact, he has provided a macro that I consider the correct way to do it... See the next section.

7.3.9. Counting Words, You Should Use This Macro!

The following macro was sent to me by Andrew Brown, as already mentioned. You really should check his web site, it has a lot of very nice things; and in case you missed it, he wrote a book. It is not about macro programming, but I really enjoyed parts of it. Check it out!

Listing 7.24: *Count words safely.*

```

Sub acbwc
' v2.0.1
' 5 sept 2003
' does footnotes and selections of all sizes
' still slow with large selections, but I blame Hamburg :-)
' v 2.0.1 slightly faster with improved cursorcount routine

```

```

' not unendurably slow with lots of footnotes, using cursors.

' acb, June 2003
' rewritten version of the
' dvwc macro by me and Daniel Vogelheim

' september 2003 changed the selection count to use a word cursor for large
selections
' following hints from Andrew Pitonyak.
' this is not perfect, either, largely because move-by-word is erratic.
' it will slightly exaggerate the number of words in a selection, counting extra
' for paragraph ends and some punctuation.
' but it is still much quicker than the old method.

Dim xDoc, xSel, nSelcount
Dim nAllChars
Dim nAllWords
Dim nAllPars
Dim thisrange, sRes
Dim nSelChars, nSelwords, nSel
Dim atext, bigtext
Dim fnotes, thisnote, nfnotes, fnotecount
Dim oddthing, startcursor, stopcursor
    xDoc = thiscomponent
    xSel = xDoc.getCurrentSelection()
    nSelCount = xSel.getCount()
    bigText=xDoc.getText()

    ' by popular demand ...
    fnotes=xdoc.getFootNotes()
    If fnotes.hasElements() Then
        fnotecount=0
        For nfnotes=0 To fnotes.getCount()-1
            thisnote=fnotes.getbyIndex(nfnotes)
            startcursor=thisnote.getStart()
            stopcursor=thisnote.getEnd()
            Do While thisnote.getText().compareRegionStarts(startcursor,stopcursor) AND
startcursor.gotoNextWord(FALSE)
                fnotecount=fnotecount+1
            Loop
        Next nfnotes
    End If

    ' this next "If" works around the problem that If you have made a selection,
then

```



```

' collapse it, and count again, the empty selection is still counted,
' which was confusing and ugly
If nSelCount=1 and xSel.GetByIndex(0).GetString()="" Then
    nSelCount=0
End If

' access document statistics
nAllChars = xDoc.CharacterCount
nAllWords = xDoc.WordCount
nAllPars = xDoc.ParagraphCount

' initialize counts
nSelChars = 0
nSelWords = 0
' the fancy bit starts here
' iterate over multiple selection
For nSel = 0 To nSelCount - 1
    thisrange=xSel.GetByIndex(nSel)
    atext=thisrange.GetString()
    If len(atext)< 220 Then
        nselwords=nSelWords+stringcount(atext)
    Else
        nselwords=nSelWords+Cursorcount(thisrange)
    End If
    nSelChars=nSelChars+len(atext)
Next nSel

' dialog code rewritten for legibility
If fnotes.hasElements() Then
    sRes="Document count (with footnotes): " + nAllWords + _
        " words. " + chr(13)
    sRes= sRes + "Word count without footnotes: " + _
        str(nAllWords-fnotecount) + _
        " words. " + chr(13)+"(Total: " +nAllChars +" Chars in "
Else
    sRes= "Document count: " + nAllWords +" words. " + chr(13)+"(" + _
        nAllChars +" Chars in "
End If
sRes=sRes + nAllPars +" Paragraphs.)"+ chr(13)+ chr(13)
If nselCount>0 Then
    sRes=sRes + "Selected text count: " + nSelWords + _
        " words" + chr(13) + _
        "(" + nSelChars + " chars"
    If nSelcount=1 Then
        sRes=sRes + " In " + str(nselCount) + " selection.)"
    Else
        REM I don't know why, but need this adjustment

```

```

        sRes=sRes + " In " + str(nselCount-1) + " selections.)"
    End If
    sRes=sRes+chr(13)+chr(13)+"Document minus selected:" + chr(13)+ _
        str(nAllWords-nSelWords) + " words." +chr(13) +chr(13)
End If
If fnotes.HasElements() Then
    sRes=sRes+"There are "+ str(fnotecount) + " words in "+ fnotes.getCount()
+_
        " footnotes." +chr(13) +chr(13)
End If
msgbox(sRes,64,"acb Word Count")
End Sub

```

```

function Cursorcount(aRange)
' acb September 2003
' quick count for use with large selections
' based on Andrew Pitonyak's WordCountWordCursor() function
' but made cruder, in line with my general tendency.
Dim lnumwords as long
Dim atext
Dim startcursor, stopcursor as object
atext=arange.GetText()
lnumwords=0
If not atext.CompareRegionStarts(aRange.getStart(),aRange.getEnd()) Then
    startcursor=atext.CreateTextCursorByRange(aRange.getStart())
    stopcursor=atext.CreateTextCursorByRange(aRange.getEnd())
Else
    startcursor=atext.CreateTextCursorByRange(aRange.getEnd())
    stopcursor=atext.CreateTextCursorByRange(aRange.getStart())
End If
Do while aText.CompareRegionEnds(startCursor, stopcursor) >= 0 and _
    startCursor.GotoNextWord(False)
    lnumwords=lnumwords+1
Loop
CursorCount=lnumwords-1
end function

```

```

Function stringcount(astring)
' acb June 2003
' slower, more accurate word count
' for use with smaller strings
' sharpened up by David Hammerton (http://crazney.net/) in September 2003
' to allow those who put two spaces after punctuation to escape their
' just deserts
Dim nspaces,i,testchar,nextchar
nspaces=0
For i= 1 To len(astring)-1

```

```

testchar=mid(astring,i,1)
select Case testchar
Case " ",chr(9),chr(13)
    nextchar = mid(astring,i+1,1)
    select Case nextchar
        Case " ",chr(9),chr(13),chr(10)
            nspaces=nspaces
        Case Else
            nspaces=nspaces+1
    end select
end select
Next i
stringcount=nspaces+1
End Function

```

7.4. Replacing Selected Space Using Strings

In general, you should not remove extra space by reading the selected text and then writing new values back. One reason is that strings are limited to 64K in size, and the other is that it is possible to lose formatting information. I left these examples in place because they work for the problems they were written to solve before I learned how I could do the same thing with cursors, and because they demonstrate techniques of inserting special characters. This first macro replaces all new paragraphs and new lines with a space character. These are also the examples that demonstrate how to insert control characters (new paragraphs, line breaks, etc.) into the text.

Listing 7.25: *Modify the text in an unsafe way.*

```

Sub SelectedNewLinesToSpaces
    Dim lSelCount&, oSels As Object
    Dim iWhichSelection As Integer, lIndex As Long
    Dim s$, bSomethingChanged As Boolean

    oSels = ThisComponent.getCurrentSelection()
    lSelCount = oSels.getCount()
    For iWhichSelection = 0 To lSelCount - 1
        bSomethingChanged = False
        REM What If the string is bigger than 64K? Oops
        s = oSels.getByIndex(iWhichSelection).getString()
        lIndex = 1
        Do While lIndex < Len(s)
            Select Case Asc(Mid(s, lIndex, 1))
            Case 13
                'We found a new paragraph marker.
                'The next character will be a 10!
                If lIndex < Len(s) AND Asc(Mid(s, lIndex+1, 1)) = 10 Then
                    Mid(s, lIndex, 2, " ")
                Else

```

```

        Mid(s, lIndex, 1, " ")
    End If
    lIndex = lIndex + 1
    bSomethingChanged = True
Case 10
    'New line unless the previous charcter is a 13
    'Remove this entire case statement to ignore only new lines!
    If lIndex > 1 AND Asc(Mid(s, lIndex-1, 1)) <> 13 Then
        'This really is a new line and NOT a new paragraph.
        Mid(s, lIndex, 1, " ")
        lIndex = lIndex + 1
        bSomethingChanged = True
    Else
        'Nope, this one really was a new paragraph!
        lIndex = lIndex + 1
    End If
Case Else
    'Do nothing If we do not match something else
    lIndex = lIndex + 1
End Select
Loop
If bSomethingChanged Then
    oSels.GetByIndex(iWhichSelection).setString(s)
End If
Next
End Sub

```

I was also asked to convert new paragraphs to new lines. Using cursors is clearly a better idea, but I did not know how to do it. I think that this example is still instructive, so I left it in. I first delete the selected text and then start adding the text back.

Listing 7.26: Use text cursors to replace space.

```

Sub SelectedNewParagraphsToNewLines
    Dim lSelCount&, oSels As Object, oSelection As Object
    Dim iWhichSelection As Integer, lIndex As Long
    Dim oText As Object, oCursor As Object
    Dim s$, lLastCR As Long, lLastNL As Long

    oSels = ThisComponent.getCurrentSelection()
    lSelCount = oSels.getCount()
    oText=ThisComponent.Text

    For iWhichSelection = 0 To lSelCount - 1
        oSelection = oSels.GetByIndex(iWhichSelection)
        oCursor=oText.createTextCursorByRange(oSelection)
        s = oSelection.getString()
        'Delete the selected text!
        oCursor.setString("")
    Next
End Sub

```

```

lIndex = 1
Do While lIndex <= Len(s)
    Select Case Asc(Mid(s, lIndex, 1))
    Case 13
        oText.insertControlCharacter(oCursor, _
com.sun.star.text.ControlCharacter.LINE_BREAK, False)
        'I wish I had short circuit booleans!
        'Skip the next LF If there is one. I think there
        'always will be but I can not verify this
        If (lIndex < Len(s)) Then
            If Asc(Mid(s, lIndex+1, 1)) = 10 Then lIndex = lIndex + 1
        End If
    Case 10
        oText.insertControlCharacter(oCursor, _
com.sun.star.text.ControlCharacter.LINE_BREAK, False)
    Case Else
        oCursor.setString(Mid(s, lIndex, 1))
        oCursor.GoRight(1, False)
    End Select
    lIndex = lIndex + 1
Loop
Next
End Sub

```

7.4.1. Compare Cursors And String Examples

Here are some macros that I wrote using the cursor methods and following them, the same way I had done them before I had my framework!

Listing 7.27: I wrote these a long time ago!

```

' *****
' Author: Andrew Pitonyak
' email: andrew@pitonyak.org
' The purpose of this macro is to make it easier to use the Text<-->Table
' method which wants trailing and leading white space removed.
' It also wants new paragraphs and NOT new lines!
Sub CRTToNLMain
    Dim oCurs(), i%, sPrompt$
    sPrompt$ = "Convert New Paragraphs to New Lines for the ENTIRE document?"
    If Not CreateSelectedTextIterator(ThisComponent, sPrompt$, oCurs()) Then Exit Sub
    For i% = LBOUND(oCurs()) To UBOUND(oCurs())
        CRTToNLWorker(oCurs(i%, 0), oCurs(i%, 1), ThisComponent.Text)
    Next i%
End Sub
Sub CRTToNLWorker(oLCurs As Object, oRCurs As Object, oText As Object)
    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oText) Then Exit Sub
    If oText.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Sub

```

```

oLCurs.goRight(0, False)
Do While oLCurs.gotoNextParagraph(False) AND oText.compareRegionEnds(oLCurs,
oRCurs) >= 0
    oLCurs.goLeft(1, True)
    oLCurs.setString("")
    oLCurs.goRight(0, False)
    oText.insertControlCharacter(oLCurs, _
        com.sun.star.text.ControlCharacter.LINE_BREAK, True)
Loop
End Sub
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'The real purpose of this macro is to make it easier to use the Text<-->Table
'method which wants trailing and leading white space removed.
'It also wants new paragraphs and NOT new lines!
Sub SpaceToTabsInWordsMain
    Dim oCurs(), i%, sPrompt$
    sPrompt$ = "Convert Spaces to TABS for the ENTIRE document?"
    If Not CreateSelectedTextIterator(ThisComponent, sPrompt$, oCurs()) Then Exit
Sub
    For i% = LBOUND(oCurs()) To UBOUND(oCurs())
        SpaceToTabsInWordsWorker(oCurs(i%, 0), oCurs(i%, 1), ThisComponent.Text)
    Next i%
End Sub
Sub SpaceToTabsInWordsWorker(oLCurs As Object, oRCurs As Object, oText As
Object)
    Dim iCurrentState As Integer, iChar As Integer, bChanged As Boolean
    Const StartLineState = 0
    Const InWordState = 1
    Const BetweenWordState = 2

    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oText) Then Exit Sub
    If oText.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Sub
    oLCurs.goRight(0, False)
    iCurrentState = StartLineState
    bChanged = False
    Do While oLCurs.goRight(1, True) AND oText.compareRegionEnds(oLCurs, oRCurs)
    >= 0
        iChar = Asc(oLCurs.getString())
        If iCurrentState = StartLineState Then
            If IsWhiteSpace(iChar) Then
                oLCurs.setString("")
            Else
                iCurrentState = InWordState
            End If
        ElseIf iCurrentState = InWordState Then
            bChanged = True

```

```

Select Case iChar
Case 9
    REM It is already a tab, ignore it
    iCurrentState = BetweenWordState
Case 32, 160
    REM Convert the space to a tab
    oLCurs.setString(Chr(9))
    oLCurs.goRight(1, False)
    iCurrentState = BetweenWordState
Case 10
    REM Remove the new line and insert a new paragraph
    oText.insertControlCharacter(oLCurs, _
        com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, True)
    oLCurs.goRight(1, False)
    iCurrentState = StartLineState
Case 13
    iCurrentState = StartLineState
Case Else
    oLCurs.gotoEndOfWord(True)
End Select
ElseIf iCurrentState = BetweenWordState Then
    Select Case iChar
    Case 9, 32, 160
        REM We already added a tab, this is extra stuff!
        oLCurs.setString("")
    Case 10
        REM Remove the new line and insert a new paragraph
        REM and be sure to delete the leading TAB that we already
        REM added in and that should come out!
        oText.insertControlCharacter(oLCurs, _
            com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, True)
        oLCurs.goLeft(0, False)
        REM and over the TAB for deletion
        oLCurs.goLeft(1, True)
        oLCurs.setString("")
        oLCurs.goRight(1, False)
        iCurrentState = StartLineState
    Case 13
        REM first, backup over the CR, then select the TAB and delete it
        oLCurs.goLeft(0, False)
        oLCurs.goLeft(1, False)
        oLCurs.goLeft(1, True)
        oLCurs.setString("")
        REM finally, move back over the CR that we will ignore
        oLCurs.goRight(1, True)
        iCurrentState = StartLineState
    Case Else

```

```

        iCurrentState = InWordState
        oLCurs.gotoEndOfWord(False)
    End Select
End If
oLCurs.goRight(0, False)
Loop
If bChanged Then
    REM To get here, we went one character too far right
    oLCurs.goLeft(1, False)
    oLCurs.goLeft(1, True)
    If Asc(oLCurs.getString()) = 9 Then oLCurs.setString("")
End If
End Sub
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub TabsToSpacesMain
    Dim oCurs(), i%, sPrompt$
    sPrompt$ = "Convert TABS to Spaces for the ENTIRE document?"
    If Not CreateSelectedTextIterator(ThisComponent, sPrompt$, oCurs()) Then Exit Sub
    For i% = LBOUND(oCurs()) To UBOUND(oCurs())
        TabsToSpacesWorker(oCurs(i%, 0), oCurs(i%, 1), ThisComponent.Text)
    Next i%
End Sub
Sub TabsToSpacesWorker(oLCurs As Object, oRCurs As Object, oText As Object)
    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oText) Then Exit Sub
    If oText.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Sub
    oLCurs.goRight(0, False)
    Do While oLCurs.goRight(1, True) AND oText.compareRegionEnds(oLCurs, oRCurs)
    >= 0
        If Asc(oLCurs.getString()) = 9 Then
            oLCurs.setString("    ") 'Change a tab into 4 spaces
        End If
        oLCurs.goRight(0, False)
    Loop
End Sub
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'sPrompt : how to ask if should iterate over the entire text
'oCurs() : Has the return cursors
'Returns true if should iterate and false if should not
Function CreateSelectedTextIterator(oDoc As Object, sPrompt As String, oCurs())
As Boolean
    Dim oSels As Object, oSel As Object, oText As Object
    Dim lSelCount As Long, lWhichSelection As Long
    Dim oLCurs As Object, oRCurs As Object

```



```

CreateSelectedTextIterator = True
oText = oDoc.Text
If Not IsAnythingSelected(ThisComponent) Then
    Dim i%
    i% = MsgBox("No text selected!" + Chr(13) + sPrompt, _
        1 OR 32 OR 256, "Warning")
    If i% = 1 Then
        oLCurs = oText.createTextCursor()
        oLCurs.gotoStart(False)
        oRCurs = oText.createTextCursor()
        oRCurs.gotoEnd(False)
        oCurs = DimArray(0, 1)
        oCurs(0, 0) = oLCurs
        oCurs(0, 1) = oRCurs
    Else
        oCurs = DimArray()
        CreateSelectedTextIterator = False
    End If
Else
    oSels = ThisComponent.getCurrentSelection()
    lSelCount = oSels.getCount()
    oCurs = DimArray(lSelCount - 1, 1)
    For lWhichSelection = 0 To lSelCount - 1
        oSel = oSels.getByIndex(lWhichSelection)
        'If I want to know if NO text is selected, I could
        'do the following:
        'oLCurs = oText.CreateTextCursorByRange(oSel)
        'If oLCurs.isCollapsed() Then ...
        oLCurs = GetLeftMostCursor(oSel, oText)
        oRCurs = GetRightMostCursor(oSel, oText)
        oCurs(lWhichSelection, 0) = oLCurs
        oCurs(lWhichSelection, 1) = oRCurs
    Next
End If
End Function
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'oDoc is a writer object
Function IsAnythingSelected(oDoc As Object) As Boolean
    Dim oSels As Object, oSel As Object, oText As Object, oCursor As Object
    IsAnythingSelected = False
    If IsNull(oDoc) Then Exit Function
    ' The current selection in the current controller.
    'If there is no current controller, it returns NULL.
    oSels = oDoc.getCurrentSelection()

```

```

If IsNull(oSels) Then Exit Function
If oSels.getCount() = 0 Then Exit Function
If oSels.getCount() > 1 Then
    IsAnythingSelected = True
Else
    oSel = oSels.getByIndex(0)
    oCursor = oDoc.Text.CreateTextCursorByRange(oSel)
    If Not oCursor.IsCollapsed() Then IsAnythingSelected = True
End If
End Function
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'oSelection is a text selection or cursor range
'oText is the text object
Function GetLeftMostCursor(oSel As Object, oText As Object) As Object
    Dim oRange As Object, oCursor As Object

    If oText.compareRegionStarts(oSel.getEnd(), oSel) >= 0 Then
        oRange = oSel.getEnd()
    Else
        oRange = oSel.getStart()
    End If
    oCursor = oText.CreateTextCursorByRange(oRange)
    oCursor.goRight(0, False)
    GetLeftMostCursor = oCursor
End Function
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'oSelection is a text selection or cursor range
'oText is the text object
Function GetRightMostCursor(oSel As Object, oText As Object) As Object
    Dim oRange As Object, oCursor As Object

    If oText.compareRegionStarts(oSel.getEnd(), oSel) >= 0 Then
        oRange = oSel.getStart()
    Else
        oRange = oSel.getEnd()
    End If
    oCursor = oText.CreateTextCursorByRange(oRange)
    oCursor.goLeft(0, False)
    GetRightMostCursor = oCursor
End Function
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org

```

```

'oSelection is a text selection or cursor range
'oText is the text object
Function IsWhiteSpace(iChar As Integer) As Boolean
    Select Case iChar
        Case 9, 10, 13, 32, 160
            IsWhiteSpace = True
        Case Else
            IsWhiteSpace = False
    End Select
End Function

'*****
'Here starts the OLD macros!
'Author: Andrew Pitonyak
Sub ConvertSelectedNewParagraphToNewLine
    Dim lSelCount&, oSels As Object, oSelection As Object
    Dim iWhichSelection As Integer, lIndex As Long
    Dim oText As Object, oCursor As Object
    Dim s$, lLastCR As Long, lLastNL As Long

    'There may be multiple selections present!
    oSels = ThisComponent.getCurrentSelection()
    lSelCount = oSels.getCount()
    oText=ThisComponent.Text

    For iWhichSelection = 0 To lSelCount - 1
        oSelection = oSels.getByIndex(iWhichSelection)
        oCursor=oText.createTextCursorByRange(oSelection)
        s = oSelection.getString()
        oCursor.setString("")
        lLastCR = -1
        lLastNL = -1
        lIndex = 1
        Do While lIndex <= Len(s)
            Select Case Asc(Mid(s, lIndex, 1))
                Case 13
                    oText.insertControlCharacter(oCursor, _
                        com.sun.star.text.ControlCharacter.LINE_BREAK, False)
                    'Boy I wish I had short circuit booleans!
                    'Skip the next LF if there is one. I think that there
                    'always will be but I can not verify this
                    If (lIndex < Len(s)) Then
                        If Asc(Mid(s, lIndex+1, 1)) = 10 Then lIndex = lIndex + 1
                    End If
                Case 10

```

```

        oText.insertControlCharacter(oCursor, _
            com.sun.star.text.ControlCharacter.LINE_BREAK, False)
    Case Else
        oCursor.setString(Mid(s, lIndex, 1))
        oCursor.GoRight(1, False)
    End Select
    lIndex = lIndex + 1
Loop
Next
End Sub

'I decided to write this as a finite state machine
'Finite state machines are a wonderful thing :-)
Sub ConvertSelectedSpaceToTabsBetweenWords
    Dim lSelCount&, oSels As Object, oSelection As Object
    Dim iWhichSelection As Integer, lIndex As Long
    Dim oText As Object, oCursor As Object
    Dim s$, lLastCR As Long, lLastNL As Long
    REM What states are supported
    Dim iCurrentState As Integer
    Const StartLineState = 0
    Const InWordState = 1
    Const BetweenWordState = 2
    REM Transition Points
    Dim iWhatFound As Integer
    Const FoundWhiteSpace = 0
    Const FoundNewLine = 1
    Const FoundOther = 2
    Const ActionIgnoreChr = 0
    Const ActionDeleteChr = 1
    Const ActionInsertTab = 2
    REM Define the state transitions
    Dim iNextState(0 To 2, 0 To 2, 0 To 1) As Integer
    iNextState(StartLineState, FoundWhiteSpace, 0) = StartLineState
    iNextState(StartLineState, FoundNewLine, 0) = StartLineState
    iNextState(StartLineState, FoundOther, 0) = InWordState

    iNextState(InWordState, FoundWhiteSpace, 0) = BetweenWordState
    iNextState(InWordState, FoundNewLine, 0) = StartLineState
    iNextState(InWordState, FoundOther, 0) = InWordState

    iNextState(BetweenWordState, FoundWhiteSpace, 0) = BetweenWordState
    iNextState(BetweenWordState, FoundNewLine, 0) = StartLineState
    iNextState(BetweenWordState, FoundOther, 0) = InWordState

    REM Define the state actions
    iNextState(StartLineState, FoundWhiteSpace, 1) = ActionDeleteChr

```

```

iNextState(StartLineState, FoundNewLine, 1)      = ActionIgnoreChr
iNextState(StartLineState, FoundOther, 1)         = ActionIgnoreChr

iNextState(InWordState, FoundWhiteSpace, 1)       = ActionDeleteChr
iNextState(InWordState, FoundNewLine, 1)          = ActionIgnoreChr
iNextState(InWordState, FoundOther, 1)            = ActionIgnoreChr

iNextState(BetweenWordState, FoundWhiteSpace, 1) = ActionDeleteChr
iNextState(BetweenWordState, FoundNewLine, 1)    = ActionIgnoreChr
iNextState(BetweenWordState, FoundOther, 1)       = ActionInsertTab

'There may be multiple selections present!
oSels = ThisComponent.getCurrentSelection()
lSelCount = oSels.getCount()
oText=ThisComponent.Text

For iWhichSelection = 0 To lSelCount - 1
    oSelection = oSels.getByIndex(iWhichSelection)
    oCursor=oText.createTextCursorByRange(oSelection)
    s = oSelection.getString()
    oCursor.setString("")
    lLastCR = -1
    lLastNL = -1
    lIndex = 1
    iCurrentState = StartLineState
    Do While lIndex <= Len(s)
        Select Case Asc(Mid(s, lIndex, 1))
            Case 9, 32, 160
                iWhatFound = FoundWhiteSpace
            Case 10
                iWhatFound = FoundNewLine
                lLastNL = lIndex
            Case 13
                iWhatFound = FoundNewLine
                lLastCR = lIndex
            Case Else
                iWhatFound = FoundOther
        End Select
        Select Case iNextState(iCurrentState, iWhatFound, 1)
            Case ActionDeleteChr
                'By choosing to not insert, it is deleted!
            Case ActionIgnoreChr
                'This really means that I must add the character Back!
                If lLastCR = lIndex Then
                    'Inserting a control character seems to move the
                    'cursor around
                    oText.insertControlCharacter(oCursor, _

```

```

        com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
'    oText.insertControlCharacter(oCursor, _
'        com.sun.star.text.ControlCharacter.APPEND_PARAGRAPH, False)
'    oCursor.goRight(1, False)
'    Print "Inserted a CR"
    ElseIf lLastNL = lIndex Then
        If lLastCR + 1 <> lIndex Then
            oText.insertControlCharacter(oCursor, _
                com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
            'com.sun.star.text.ControlCharacter.LINE_BREAK, False)
            oCursor.goRight(1, False)
            'Print "Inserted a NL"
        End If
        'Ignore this one
    Else
        oCursor.setString(Mid(s, lIndex, 1))
        oCursor.GoRight(1, False)
        'Print "Inserted Something"
    End If
Case ActionInsertTab
    oCursor.setString(Chr$(9) + Mid(s, lIndex, 1))
    oCursor.GoRight(2, False)
    'Print "Inserted a tab"
End Select
lIndex = lIndex + 1
'MsgBox "index = " + lIndex + Chr(13) + s
iCurrentState = iNextState(iCurrentState, iWhatFound, 0)
Loop
Next
End Sub

Sub ConvertAllTabsToSpace
    DIM oCursor As Object, oText As Object

    Dim nSpace%, nTab%, nPar%, nRet%, nTot%
    Dim justStarting As Boolean

    oText=ThisComponent.Text           'Get the Text component
    oCursor=oText.createTextCursor()   'Create a cursor in the text
    oCursor.gotoStart(FALSE)           'Goto the start but do NOT select the text
as you go
    Do While oCursor.GoRight(1, True) 'Move right one chracter and select it
        If Asc(oCursor.getString()) = 9 Then
            oCursor.setString("    ") 'Change a tab into 4 spaces
        End If
        oCursor.goRight(0,FALSE)       'Deselect text!
    Loop
End Sub

```

```

Sub ConvertSelectedTabsToSpaces
    Dim lSelCount&, oSels As Object
    Dim iWhichSelection As Integer, lIndex As Long
    Dim s$, bSomethingChanged As Boolean

    'There may be multiple selections present!
    'There will probably be one more than expected because
    'it will count the current cursor location as one piece
    'of selected text, just so you know!
    oSels = ThisComponent.getCurrentSelection()
    lSelCount = oSels.getCount()
    'Print "total selected = " + lSelCount
    For iWhichSelection = 0 To lSelCount - 1
        bSomethingChanged = False
        s = oSels.getByIndex(iWhichSelection).getString()
        'Print "Text group " + iWhichSelection + " is of length " + Len(s)
        lIndex = 1
        Do While lIndex < Len(s)
            'Print "ASCII at " + lIndex + " = " + Asc(Mid(s, lIndex, 1))
            If Asc(Mid(s, lIndex, 1)) = 9 Then
                s = ReplaceInString(s, lIndex, 1, " ")
                bSomethingChanged = True
                lIndex = lIndex + 3
            End If
            lIndex = lIndex + 1
            'Print ":" + lIndex + "(" + s + ")"
        Loop
        If bSomethingChanged Then
            oSels.getByIndex(iWhichSelection).setString(s)
        End If
    Next
End Sub

Function ReplaceInString(s$, index&, num&, replaces$) As String
    If index <= 1 Then
        'Place this in front of the string
        If num < 1 Then
            ReplaceInString = replaces + s
        ElseIf num > Len(s) Then
            ReplaceInString = replaces
        Else
            ReplaceInString = replaces + Right(s, Len(s) - num)
        End If
    ElseIf index + num > Len(s) Then
        ReplaceInString = Left(s, index - 1) + replaces
    Else

```

```

        ReplaceInString = Left(s,index - 1) + replaces + _
                        Right(s, Len(s) - index - num + 1)

    End If
End Function

```

7.5. Setting Text Attributes

When this macro is run, it affects the paragraph containing the cursor. The font and the size is set. The CharPosture attribute controls italics, CharWeight controls bold, and CharUnderline controls the underline type. Valid values are found at:

<http://api.openoffice.org/docs/common/ref/com/sun/star/style/CharacterProperties.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/FontWeight.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/FontSlant.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/FontUnderline.html>

Listing 7.28: *Demonstrate how to set text attributes.*

```

'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub SetTextAttributes
    Dim document As Object
    Dim Cursor
    Dim oText As Object
    Dim mySelection As Object
    Dim Font As String

    document=ThisComponent
    oText = document.Text
    Cursor = document.currentcontroller.getViewCursor()
    mySelection = oText.createTextCursorByRange(Cursor.getStart())
    mySelection.gotoStartOfParagraph(false)
    mySelection.gotoEndOfParagraph(true)
    mySelection.CharFontName="Courier New"
    mySelection.CharHeight="10"
    'Time to set Italic or NOT italic as the case with
    'NONE, OBLIQUE, ITALIC, DONTKNOW, REVERSE_OBLIQUE, REVERSE_ITALIC
    mySelection.CharPosture = com.sun.star.awt.FontSlant.ITALIC
    'So you want BOLD text?
    'DONTKNOW, THIN, ULTRALIGHT, LIGHT, SEMILIGHT,
    'NORMAL, SEMIBOLD, BOLD, ULTRABOLD, BLACK
    'These are really only constants where THIN is 50, NORMAL is 100
    ' BOLD is 150, and BLACK is 200.
    mySelection.CharWeight = com.sun.star.awt.FontWeight.BOLD
    'If underlining is your thing
    'NONE, SINGLE, DOUBLE, DOTTED, DONTKNOW, DASH, LONGDASH,
    'DASHDOT, DASHDOTDOT, SMALLWAVE, WAVE, DOUBLEWAVE, BOLD,
    'BOLDDOTTED, BOLDDASH, BOLDLONGDASH, BOLDDASHDOT,

```



```

'BOLDDASHDOTDOT, BOLDWAVE
mySelection.CharUnderline = com.sun.star.awt.FontUnderline.SINGLE
'I have not experimented with this enough to know what the true
'implications of this really is, but I do know that it seems to set
'the character locale to German.
Dim aLanguage As New com.sun.star.lang.Locale
aLanguage.Country = "de"
aLanguage.Language = "de"
mySelection.CharLocale = aLanguage
End Sub

```

7.6. Insert text

Listing 7.29: *Demonstrate how to insert strings into a text object.*

```

'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub InsertSimpleText
    Dim oDoc As Object
    Dim oText As Object
    Dim oVCurs As Object
    Dim oTCurs As Object

    oDoc = ThisComponent
    oText = oDoc.Text
    oVCurs = oDoc.CurrentController.getViewCursor()
    oTCurs = oText.createTextCursorByRange(oVCurs.getStart())
    ' Place the text to insert here
    oText.insertString(oTCurs, "—", FALSE)
End Sub

```

7.6.1. Insert new paragraph

Use insertString() to insert regular text. use insertControlCharacter() to insert special characters such as a new paragraph. The text range object, the example uses a text cursor, identifies where the text is inserted. The final boolean value specifies if the cursor should be expanded to include the newly inserted text. I use the value of False to not expand the cursor.

Listing 7.30: *Insert a line break control character.*

```

'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub InsertTextWithABreak
    Dim oDoc
    Dim oText
    Dim oVCurs
    Dim oCursor

    oText = ThisComponent.getText()

```

```

oVCurs = ThisComponent.CurrentController.getViewCursor()
oCursor = oText.createTextCursorByRange(oVCurs.getStart())
' Place the text to insert here
oText.insertString(oCursor, _
    "- I am new text before the break -", FALSE)
oText.insertControlCharacter(oCursor, _
    com.sun.star.text.ControlCharacter.LINE_BREAK, False)
oText.insertString(oCursor, "- I am new text after the break -", FALSE)
End Sub

```

7.7. Fields

7.7.1. Insert a formatted date field into a Write document

This will insert the text “Today is <date> ” where the date is formatted as “DD. MMM YYYY”. This will create the date format if it does not exist. For more information on valid formats, see the help contents on topic “number formats; formats”.

Listing 7.31: *Insert a formatted date field into a Write document.*

```

'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
'uses: FindCreateNumberFormatStyle
Sub InsertDateField
    Dim oDoc
    Dim oText
    Dim oVCurs
    Dim oTCurs
    Dim oDateTime
    Dim s$

    oDoc = ThisComponent
    If oDoc.SupportsService("com.sun.star.text.TextDocument") Then
        oText = oDoc.Text
        oVCurs = oDoc.CurrentController.getViewCursor()
        oTCurs = oText.createTextCursorByRange(oVCurs.getStart())
        oText.insertString(oTCurs, "Today is ", FALSE)
        ' Create the DateTime type.
        s = "com.sun.star.text.TextField.DateTime"
        oDateTime = oDoc.createInstance(s)
        oDateTime.IsFixed = TRUE
        oDateTime.NumberFormat = FindCreateNumberFormatStyle(_
            "DD. MMMM YYYY", oDoc)

        oText.insertTextContent(oTCurs,oDateTime,FALSE)
        oText.insertString(oTCurs, " ",FALSE) Else
            MsgBox "Sorry, this macro requires a TextDocument"
    End If
End Sub

```

```
End Sub
```

7.7.2. Inserting a Note (Annotation)

Listing 7.32: *Add a note at the cursor.*

```
Sub AddNoteAtCursor
    Dim vDoc, vViewCursor, oCurs, vTextField
    Dim s$

    'Lets lie and say that this was added ten days ago!
    Dim aDate As New com.sun.star.util.Date
    With aDate
        .Day    = Day(Now - 10)
        .Month  = Month(Now - 10)
        .Year   = Year(Now - 10)
    End With

    vDoc = ThisComponent
    vViewCursor = vDoc.getCurrentController().getViewCursor()
    oCurs=vDoc.getText().createTextCursorByRange(vViewCursor.getStart())
    s = "com.sun.star.text.TextField.Annotation"
    vTextField = vDoc.createInstance(s)
    With vTextField
        .Author   = "AP"
        .Content  = "It sure is fun to insert notes into my document"
        'Ommmit the date and it defaults to today!
        .Date     = aDate
    End With
    vDoc.Text.insertTextContent(oCurs, vTextField, False)
End Sub
```

7.8. Inserting A New Page

In my quest to insert a new page into a document, I stumbled across the following link:

<http://api.openoffice.org/docs/common/ref/com/sun/star/style/ParagraphProperties.html>

which discusses two properties. The *PageNumberOffset* states: “If a page break property is set at a paragraph, this property contains the new value for the page number.” The *PageDescName* property states: “If this property is set, it creates a page break before the paragraph it belongs to and assigns the value as the name of the new page style sheet to use.” I reasoned that if I set the *PageDescName*, then I could create a new page and set the page number. What was not said is that the *PageDescName* is the name of the new page style to use after the page break. If you do not use an existing page style, then this will fail!

Listing 7.33: *Insert a page break.*

```
Sub ExampleNewPage
    Dim oSels As Object, oSel As Object, oText As Object
```

```

Dim lSelCount As Long, lWhichSelection As Long
Dim oLCurs As Object, oRCurs As Object

oText = ThisComponent.Text
oSels = ThisComponent.getCurrentSelection()
lSelCount = oSels.getCount()
For lWhichSelection = 0 To lSelCount - 1
    oSel = oSels.getByIndex(lWhichSelection)
    oLCurs = oText.CreateTextCursorByRange(oSel)
    oLCurs.gotoStartOfParagraph(false)
    oLCurs.gotoEndOfParagraph(true)
    REM Preserve the existing page style!
    oLCurs.PageDescName = oLCurs.PageStyleName
    oLCurs.PageNumberOffset = 7
Next
End Sub

```

7.8.1. Removing Page Breaks

I have not thoroughly researched this, but I have done a few experiments. Setting the PageDescName causes a page break. It is also possible that the BreakType is set. The following macro detects and removes page breaks. I have only performed minimal testing. I have not concerned myself with new page offsets.

***Listing 7.34:** Find and remove page breaks.*

```

Sub FindPageBreaks
    REM Author: Andrew Pitonyak
    Dim iCnt As Long
    Dim oCursor as Variant
    Dim oText As Variant
    Dim s As String

    oText = ThisComponent.Text
    oCursor = oText.CreateTextCursor()
    oCursor.GoToStart(False)
    Do
        If NOT oCursor.gotoEndOfParagraph(True) Then Exit Do
        iCnt = iCnt + 1
        If NOT IsEmpty(oCursor.PageDescName) Then
            s = s & "Paragraph " & iCnt & " has a new page to style " & _
                oCursor.PageDescName & CHR$(10)
            oCursor.PageDescName = ""
        End If
        If oCursor.BreakType <> com.sun.star.style.BreakType.NONE Then
            s = s & "Paragraph " & iCnt & " has a page break" & CHR$(10)
            oCursor.BreakType = com.sun.star.style.BreakType.NONE
        End If
    Loop
End Sub

```

```

    Loop Until NOT oCursor.gotoNextParagraph(False)
    MsgBox s
End Sub

```

7.9. Set the document page style

The page style is set by modifying the Page Description name. This is very similar to starting a new page.

Listing 7.35: *Set the page style for the entire document.*

```

Sub SetDocumentPageStyle
    Dim oCursor As Object
    oCursor = ThisComponent.Text.createTextCursor()
    oCursor.gotoStart(False)
    oCursor.gotoEnd(True)
    Print "Current style = " & oCursor.PageStyleName
    oCursor.PageDescName = "Wow"
End Sub

```

7.10. Toggle a header or footer on or off

Each header and footer is associated with a page style. This means that you turn a header or footer on or off in the page style. The following macro turns a page header on or off at the current cursor. This will set the header on or off for every page that uses this page style.

Listing 7.36: *Set a page header on or off*

```

Sub HeaderOnAtCursor(oDoc, bHeaderState As boolean)
    Dim oVC
    Dim sName$
    Dim oStyle

    REM Get the page style name in use that the view cursor
    sName = oDoc.getCurrentController().getViewCursor().PageStyleName
    oStyle = oDoc.StyleFamilies.getByStyleName("PageStyles").getByName(sName)
    REM Use FooterIsOn to toggle the footer state.
    If oStyle.HeaderIsOn <> bHeaderState Then
        oStyle.HeaderIsOn = bHeaderState
    End If
End Sub

```

The page style is obtained from the list of page styles. The HeaderIsOn attribute is toggled on or off. To toggle a footer on or off, you should set the FooterIsOn property.

7.11. Insert An OLE Object

The rumor is that with OpenOffice version 1.1, the following code will insert an OLE object into a write document. The CLSID may be an external OLE object.

Listing 7.37: Insert an OLE object.

```
SName = "com.sun.star.text.TextEmbeddedObject"  
obj = ThisComponent.createInstance(sName)  
obj.CLSID = "47BBB4CB-CE4C-4E80-A591-42D9AE74950F"  
obj.attach(ThisComponent.currentController().Selection.getByIndex(0))
```

If you select an embedded object in writer, you can access its API with:

```
oModel = ThisComponent.currentController().Selection.Model
```

This provides the same interface to the object as if you created the object by loading a document with `loadComponentFromURL`

7.12. Setting Paragraph Style

Many styles can be set directly to the selected text including the paragraph style.

Listing 7.38: Set the paragraph style for selected paragraphs to “Heading 2”.

```
Sub SetParagraphStyle  
    Dim oSels As Object, oSel As Object, oText As Object  
    Dim lSelCount As Long, lWhichSelection As Long  
    Dim oLCurs As Object, oRCurs As Object  
  
    oText = ThisComponent.Text  
    oSels = ThisComponent.getCurrentSelection()  
    lSelCount = oSels.getCount()  
    For lWhichSelection = 0 To lSelCount - 1  
        oSel = oSels.getByIndex(lWhichSelection)  
        oSel.ParagraphStyleName = "Heading 2"  
    Next  
End Sub
```

The following example will set all paragraphs to use the same style.

Listing 7.39: Set all paragraphs to use the same style.

```
'Author: Marc Messeant  
'email: marc.liste@free.fr  
Sub AppliquerStyle()  
    Dim oText, oVCurs, oTCurs  
    oText = ThisComponent.Text  
    oVCurs = ThisComponent.CurrentController.getViewCursor()  
    oTCurs = oText.createTextCursorByRange(oVCurs.getStart())  
  
    While oText.compareRegionStarts(oTCurs.getStart(), oVCurs.getEnd())=1  
        oTCurs.paragraphStyleName = "YourStyle"  
        oTCurs.gotoNextParagraph(false)  
    Wend  
End Sub
```

7.13. Create Your Own Style

I did not test this code, I have not had time, but I have believe that it works.

Listing 7.40: Create a paragraph style.

```
vFamilies = oDoc.StyleFamilies
vStyle = oDoc.CreateInstance("com.sun.star.style.ParagraphStyle")
vParaStyles = vFamilies.getByName("ParagraphStyles")
vParaStyles.insertByName("MyStyle", vStyle)
```

7.14. Search And Replace

Searchable components support the ability to create a search descriptor. A searchable component can also find the first, next, and all occurrences of the search text. See:

<http://api.openoffice.org/docs/common/ref/com/sun/star/util/XSearchable.html>

A simple example demonstrates how to search (see Listing 7.41).

Listing 7.41: Perform a simple search based on words.

```
Sub SimpleSearchExample
    Dim vDescriptor, vFound
    ' Create a descriptor from a searchable document.
    vDescriptor = ThisComponent.createSearchDescriptor()
    ' Set the text for which to search and other
    '
    With vDescriptor
        .SearchString = "hello"
        ' These all default to false
        .SearchWords = true
        .SearchCaseSensitive = False
    End With
    ' Find the first one
    vFound = ThisComponent.findFirst(vDescriptor)
    Do While Not IsNull(vFound)
        Print vFound.getString()
        vFound.CharWeight = com.sun.star.awt.FontWeight.BOLD
        vFound = ThisComponent.findNext(vFound.End, vDescriptor)
    Loop
End Sub
```

The object returned from findFirst and findNext behave very similarly to a cursor so most things that you can do to a cursor, such as setting attributes, can also be done to this object.

7.14.1. Replacing Text

Replacing text is very similar to searching text except that it must support:

<http://api.openoffice.org/docs/common/ref/com/sun/star/util/XReplaceable.html>

The only useful method that this provides that a searchable document does not have is the ability to replace all occurrences of the found text with something else. The idea being that if

you search one at a time, then you can manually update each occurrence of the found text with the replacement text. The following example searches the text and replaces things such as “a@” with the Unicode character 257.

Listing 7.42: Replace multiple characters

```
'Author: Birgit Kellner
'email: birgit.kellner@univie.ac.at
Sub AtToUnicode
    'Andy says that sometime in the future these may have to be Variant types to
    work with Array()
    Dim numbered(5) As String, accented(5) As String
    Dim n as long
    Dim oDoc as object, oReplace as object
    numbered() = Array("A@", "a@", "I@", "i@", "U@", "u@", "Z@", "z@", _
        "O@", "o@", "H@", "h@", "D@", "d@", "L@", "l@", "M@", "m@", _
        "G@", "g@", "N@", "n@", "R@", "r@", _
        "Y@", "y@", "S@", "s@", "T@", "t@", "C@", "c@", "j@", "J@")
    accented() = Array(Chr$(256), Chr$(257), Chr$(298), Chr$(299), _
        Chr$(362), Chr$(363), Chr$(377), Chr$(378), Chr$(332), _
        Chr$(333), Chr$(7716), Chr$(7717), Chr$(7692), Chr$(7693), _
        Chr$(7734), Chr$(7735), Chr$(7746), Chr$(7747), Chr$(7748), _
        Chr$(7749), Chr$(7750), Chr$(7751), Chr$(7770), Chr$(7771), _
        Chr$(7772), Chr$(7773), Chr$(7778), Chr$(7779), Chr$(7788), _
        Chr$(7789), Chr$(346), Chr$(347), Chr$(241), Chr$(209))
    oReplace = ThisComponent.createReplaceDescriptor()
    oReplace.SearchCaseSensitive = True
    For n = LBound(numbered()) To UBound(accented())
        oReplace.SearchString = numbered(n)
        oReplace.ReplaceString = accented(n)
        ThisComponent.ReplaceAll(oReplace)
    Next n
End Sub
```

7.14.2. Searching Selected Text

The trick to searching only a selected range of text is to notice that a cursor may be used in the findNext routine. You can then check the end points of the find to see if the search went too far. This will also allow you to start searching from any cursor location. The findFirst method is not required if you already have a cursor type object to specify the starting search location with findNext. The example below uses my selected text framework and contains some enhancements suggested by Bernard Marcelly.

See Also:

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/XTextRangeCompare.html>

Listing 7.43: Search selected text

```
*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub SearchSelectedText
    Dim oCurs(), i%
    If Not CreateSelectedTextIterator(ThisComponent, _
        "Search text in the entire document?", oCurs()) Then Exit Sub
    For i% = LBound(oCurs()) To UBound(oCurs())
        SearchSelectedWorker(oCurs(i%, 0), oCurs(i%, 1), ThisComponent)
    Next i%
End Sub

*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub SearchSelectedWorker(oLCurs, oRCurs, oDoc)
    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oDoc) Then Exit Sub
    If oDoc.Text.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Sub
    oLCurs.goRight(0, False)
    Dim vDescriptor, vFound
    vDescriptor = oDoc.createSearchDescriptor()
    With vDescriptor
        .SearchString = "Paragraph"
        .SearchCaseSensitive = False
    End With
    ' There is no reason to perform a findFirst.
    vFound = oDoc.findNext(oLCurs, vDescriptor)
    REM Would you kill for short-circuit evaluation?
    Do While Not IsNull(vFound)
        REM If Not vFound.hasElements() Then Exit Do
        'See if we searched past the end
        'Not really safe because this assumes that vFound and oRCurs
        'are in the same text object (warning).
        If -1 = oDoc.Text.compareRegionEnds(vFound, oRCurs) Then Exit Do
        Print vFound.getString()
        vFound = ThisComponent.findNext(vFound.End, vDescriptor)
    Loop
End Sub
```

7.14.3. Complicated Search And Replace

Listing 7.44: Delete between two delimiters with search and replace.

```
'Deleting text between two delimiters is actually very easy
Sub deleteTextBetweenDlimeters
```

```

Dim vOpenSearch, vCloseSearch 'Open and Close descriptors
Dim vOpenFound, vCloseFound   'Open and Close find objects
Dim oDoc

oDoc = ThisComponent
' Create descriptors from the searchable document.
vOpenSearch = oDoc.createSearchDescriptor()
vCloseSearch = oDoc.createSearchDescriptor()

' Set the text for which to search and other
vOpenSearch.SearchString = "["
vCloseSearch.SearchString = "]"

' Find the first open delimiter
vOpenFound = oDoc.findFirst(vOpenSearch)
Do While Not IsNull(vOpenFound)

    'Search for the closing delimiter starting from the open delimiter
    vCloseFound = oDoc.findNext( vOpenFound.End, vCloseSearch)
    If IsNull(vCloseFound) Then
        Print "Found an opening bracket but no closing bracket!"
        Exit Do
    Else
        ' Clear the open bracket, if I do not do this, then I end up
        ' with only the text inside the brackets
        vOpenFound.setString("")
        ' select the text inside the brackets
        vOpenFound.gotoRange(vCloseFound, True)
        Print "Found " & vOpenFound.getString()
        ' clear the text inside the brackets
        vOpenFound.setString("")
        ' Clear the close bracket
        vCloseFound.setString("")
        ' Do you really want to delete ALL of the spaces?
        ' If so, then do it here!
        If vCloseFound.goRight(1, True) Then
            If vCloseFound.getString() = " " Then
                vCloseFound.setString("")
            End If
        End If
        vOpenFound = oDoc.findNext( vOpenFound.End, vOpenSearch)
    End If
Loop
End Sub

```

7.14.4. Search and Replace with Attributes and Regular Expressions

The macro surrounds all **BOLD** elements with curly brackets “{ { } }” and changes the **Bold** attribute to Normal. A regular expression is used to specify the search text.

Listing 7.45: Replace formatting with a regular expression.

```
Sub ReplaceFormatting
    'original code : Alex Savitsky
    'modified by : Laurent Godard
    'The purpose of this macro is to surround all
    'BOLD elements with { { } }
    'and change the Bold attribute to NORMAL
    'This uses regular expressions
    'The styles have to be searched too

    Dim oDoc As Object
    Dim oReplace As Object
    Dim SrchAttributes(0) As New com.sun.star.beans.PropertyValue
    Dim ReplAttributes(0) As New com.sun.star.beans.PropertyValue

    oDoc = ThisComponent
    oReplace = oDoc.createReplaceDescriptor

    'Regular expression. Match any text
    oReplace.SearchString = ".*"
    'Note the & places the found text back
    oReplace.ReplaceString = "{ { & } }"
    oReplace.SearchRegularExpression=True    'Use regular expressions
    oReplace.searchStyles=True              'We want to search styles
    oReplace.searchAll=True                 'Do the entire document

    REM This is the attribute to find
    SrchAttributes(0).Name = "CharWeight"
    SrchAttributes(0).Value =com.sun.star.awt.FontWeight.BOLD

    REM This is the attribute to replace it with
    ReplAttributes(0).Name = "CharWeight"
    ReplAttributes(0).Value =com.sun.star.awt.FontWeight.NORMAL

    REM Set the attributes in the replace descriptor
    oReplace.SetSearchAttributes(SrchAttributes())
    oReplace.SetReplaceAttributes(ReplAttributes())

    REM Now do the work!
    oDoc.replaceAll(oReplace)
End Sub
```

7.14.5. Search only the first text table

Your first find can be with `findNext` (you do not need `findFirst`), but you must specify the initial start position. The start position can be almost any text range.

Listing 7.46: Search in the first text table.

```
Sub SearchInFirstTable
    Dim oDescriptor, oFound
    Dim oTable
    Dim oCell
    Dim oDoc

    oDoc = ThisComponent
    REM Get cell A1 in the first text table
    oTable = oDoc.getTextTables().getByIndex(0)
    oCell = oTable.getCellByName("A1")

    REM Create a search descriptor
    oDescriptor = oDoc.createSearchDescriptor()
    With oDescriptor
        .SearchString = "one"
        .SearchWords = False
        .SearchCaseSensitive = False
    End With

    REM Start searching from the start of the text object in cell A1 of
    REM the first text table.
    REM oFound = ThisComponent.findFirst(oDescriptor)
    oFound = oDoc.findNext(oCell.getText().getStart(), oDescriptor)
    Do While Not IsNull(oFound)
        REM If the found text is not in a text table then finished.
        If IsNull(oFound.TextTable) Then
            Exit Sub
        End If
        REM If the found text is not in the same text table then finished.
        REM This is not fool proof, because the text table may contain
        REM another text table or some other object such as a frame,
        REM but this is close enough for a simple example.
        If NOT EqualUnoObjects(oTable, oFound.TextTable) Then
            Exit Sub
        End If
        oFound.CharWeight = com.sun.star.awt.FontWeight.BOLD
        oFound = oDoc.findNext(oFound.End, oDescriptor)
    Loop
End Sub
```

7.15. Changing The Case Of Words

OOo has a character property that causes text to be displayed as upper case, lower case, title case, or small caps. The property changes how text is displayed, not the content of the text. Although you can select an entire document and set the character case, this is only possible if all of the contained portions support the character case property. As a compromise between speed and possible problems, I use a word cursor to traverse the text setting the case of each word individually; setting the case on one word at a time was an arbitrary decision.

The macro causes an error if the text does not support the charCasemap property. Avoid the errors by adding the statement “On Local Error Resume Next” to SetWordCase().

Warning

Setting the case does not change the character, only how it is displayed. If you set the lower case attribute, you can not manually enter an upper case letter.

See Also:

<http://api.openoffice.org/common/ref/com/sun/star/style/CaseMap.html>

```
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Sub ADPSetWordCase()
    Dim oCurs() As Integer, i As Integer, sMapType As String, iMapType As Integer
    iMapType = -1
    Do While iMapType < 0
        sMapType = InputBox("To what case shall I set the words?" & Chr(10) & _
            "None, UPPER, lower, Title, or small caps?", "Change Case Type", "Title")
        sMapType = UCase(Trim(sMapType))
        If sMapType = "" Then sMapType = "EXIT"
        Select Case sMapType
            Case "EXIT"
                Exit Sub
            Case "NONE"
                iMapType = com.sun.star.style.CaseMap.NONE
            Case "UPPER"
                iMapType = com.sun.star.style.CaseMap.UPPERCASE
            Case "LOWER"
                iMapType = com.sun.star.style.CaseMap.LOWERCASE
            Case "TITLE"
                iMapType = com.sun.star.style.CaseMap.TITLE
            Case "SMALL CAPS"
                iMapType = com.sun.star.style.CaseMap.SMALLCAPS
            Case Else
                Print "Sorry, " & sMapType & " is not a recognized valid type"
        End Select
    Loop
    If Not CreateSelectedTextIterator(ThisComponent, _
        "Change the entire document?", oCurs()) Then Exit Sub
```

```

For i% = LBound(oCurs()) To UBound(oCurs())
    SetWordCase(oCurs(i%, 0), oCurs(i%, 1), ThisComponent.Text, iMapType%)
Next
End Sub
'*****
'Author: Andrew Pitonyak
'email: andrew@pitonyak.org
Function SetWordCase(vLCursor, vRCursor, oText, iMapType%)
    If IsNull(vLCursor) OR IsNull(vRCursor) OR IsNull(oText) Then Exit Function
    If oText.compareRegionEnds(vLCursor, vRCursor) <= 0 Then Exit Function
    vLCursor.goRight(0, False)
    Do While vLCursor.gotoNextWord(True)
        If oText.compareRegionStarts(vLCursor, vRCursor) > 0 Then
            vLCursor.charCasemap = iMapType%
            vLCursor.goRight(0, False)
        Else
            Exit Function
        End If
    Loop
    REM If the LAST word ends the document with no punctuation and new lines,
    REM it is not possible to goto the next word. I will now check for this case!
    If oText.compareRegionStarts(vLCursor, vRCursor) > 0 AND _
        vLCursor.gotoEndOfWord(True) Then
        vLCursor.charCasemap = iMapType%
    End If
End Function

```

I have a solution from Alan Lindsey (virtualdal@gmail.com) that modifies the text rather than by setting properties.

```

Rem -----
Rem Macro to cycle case between lower, UPPER and Title case
Rem 09/02/14 DAL
Rem 09/02/14 Doesn't work in tables
Rem 09/02/15 sort of works in tables - but changes all text in cell
Rem 09/02/15 Now works in tables as well
Rem 09/02/16 minimalism rules OK, recognizes space and tab as a separator
Rem -----
Sub Charcase
    Dim oVC

    Rem get access to the document
    oVC = thiscomponent.currentselection(0) ' get the current selection
    ctext = oVC.string                      ' get selected text
    If len(ctext) = 0 Then Exit Sub          ' nothing to do if no selection

    Rem ring the changes!
    If UCase(ctext) = ctext Then            ' if in UPPER CASE

```

```

        ctext = LCase(ctext)                ' change to lower case
    ElseIf LCase(ctext) = ctext Then        ' if lower case, make Title Case
        c1 = UCase(Mid(ctext, 1, 1))        ' get the first character
        Mid(ctext, 1, 1, c1)               ' first character to upper case
        For i=2 To Len(ctext)              ' scan the string look for
            c1 = Mid(ctext, i, 1)           ' separators
            If ( c1 = " " OR c1 = Chr(9) ) Then ' spaces and tabs
                c1 = UCase(Mid(ctext, i+1,1)) ' Change next character.
                mid(ctext, i+1, 1, c1)
            EndIf
        Next
    Else
        ctext = UCase(ctext)                ' title case, change to UPPER
    EndIf

    oVC.string = ctext                      ' replace with new text
    ThisComponent.CurrentController.select(oVC) ' reselect the text
End Sub

```

7.16. Traverse paragraphs (text cursor behavior)

Internally, a paragraph is represented something like “Blah blah.<cr><lf>”. Using the GUI to manually select the paragraph selects the text and not the trailing carriage return or line feed characters.

Listing 7.47: *Move the cursor zero spaces to clear a selection.*

```

oVCurs = ThisComponent.getCurrentController().getViewCursor()
MsgBox "(" & oVCurs.getString() & ")"
oVCurs.goRight(0, False)

```

Moving the cursor zero spaces to the right clears the selection, and leaves the cursor sitting directly before the <cr><lf>; and can therefore not be used to move to the next paragraph. It is possible, however, to move between paragraphs.

Listing 7.48: *Select the entire next paragraph.*

```

Dim oVCurs
Dim oTCurs
oVCurs = ThisComponent.getCurrentController().getViewCursor()
oTCurs = oVCurs.getText().createTextCursorByRange(oVCurs)
oTCurs.gotoNextParagraph(False) 'goto start of next paragraph.
oTCurs.gotoEndOfParagraph(True) 'Select entire paragraph.
MsgBox "(" & oTCurs.getString() & ")"

```

TIP The method gotoEndOfParagraph does not select the separating <cr><lf> characters.

The following macro demonstrates a method to visit each paragraphs and print the current paragraph style name. I more than one paragraph is selected at a time, you can not retrieve the paragraph style. Conveniently, this completely skips inserted text tables.

Listing 7.49: *Print all paragraph styles.*

```
Sub PrintAllStyles
    Dim s As String
    Dim oCurs as Variant
    Dim sCurStyle As String

    oCurs = ThisComponent.Text.CreateTextCursor()
    oCurs.GoToStart(False)
    Do
        If NOT oCurs.gotoEndOfParagraph(True) Then Exit Do
        sCurStyle = oCurs.ParaStyleName
        s = s & """" & sCurStyle & """" & CHR$(10)
    Loop Until NOT oCurs.gotoNextParagraph(False)
    MsgBox s, 0, "Styles in Document"
End Sub
```

7.16.1. Formatting macro paragraphs (an example)

I wrote a macro to inspect a document and force all code segments to use the proper paragraph styles.

Table 7.1. Paragraph styles used to format macros.

Description	Original Style	New Style
Macro with one line.	_code_one_line	_OooComputerCodeLastLine
First line	_code_first_line	_OooComputerCode
Last line	_code_last_line	_OooComputerCodeLastLine
Middle lines	_code	_OooComputerCode

I wanted to inspect the entire document, identify code segments (based on their paragraph style) and force them to use the proper convention.

Listing 7.50: *Format macro paragraphs.*

```
Sub user_CleanUpCodeSections
    worker_CleanUpCodeSections("_code_first_line", "_code", _
                                "_code_last_line", "_code_one_line")
End Sub

Sub worker_CleanUpCodeSections(firstStyle$, midStyle$, _
                                lastStyle$, onlyStyle$)
    Dim vCurCurs as Variant    'Current cursor
```



```

Dim vPrevCurs as Variant      'Previous cursor is one paragraph behind
Dim sPrevStyle As String      'Previous style
Dim sCurStyle As String      'Current style

REM Position the current cursor at the start of the second paragraph
vCurCurs = ThisComponent.Text.CreateTextCursor()
vCurCurs.GoToStart(False)
If NOT vCurCurs.gotoNextParagraph(False) Then Exit Sub

REM Position the previous cursor to select the first paragraph
vPrevCurs = ThisComponent.Text.CreateTextCursor()
vPrevCurs.GoToStart(False)
If NOT vPrevCurs.gotoEndOfParagraph(True) Then Exit Sub
sPrevStyle = vPrevCurs.ParaStyleName

Do
    If NOT vCurCurs.gotoEndOfParagraph(True) Then Exit Do
    sCurStyle = vCurCurs.ParaStyleName

    REM do the work here.
    If sCurStyle = firstStyle$ Then
        REM Current style is the first style
        REM See if the previous style was also one of these!
        Select Case sPrevStyle
            Case onlyStyle$, lastStyle$
                sCurStyle = midStyle$
                vCurCurs.ParaStyleName = sCurStyle
                vPrevCurs.ParaStyleName = firstStyle$

            Case firstStyle$, midStyle$
                sCurStyle = midStyle$
                vCurCurs.ParaStyleName = sCurStyle
        End Select

    ElseIf sCurStyle = midStyle$ Then
        REM Current style is the mid style
        REM See if the previous style was also one of these!
        Select Case sPrevStyle
            Case firstStyle$, midStyle$
                REM do nothing!

            Case onlyStyle$
                REM last style was an only style, but it comes before a mid!
                vPrevCurs.ParaStyleName = firstStyle$

            Case lastStyle$
                vPrevCurs.ParaStyleName = midStyle$

```

```

        Case Else
            sCurStyle = firstStyle$
            vCurCurs.ParaStyleName = sCurStyle
        End Select

    ElseIf sCurStyle = lastStyle$ Then
        Select Case sPrevStyle
            Case firstStyle$, midStyle$
                REM do nothing!

            Case onlyStyle$
                REM last style was an only style, but it comes before a mid!
                vPrevCurs.ParaStyleName = firstStyle$

            Case lastStyle$
                vPrevCurs.ParaStyleName = midStyle$

            Case Else
                sCurStyle = firstStyle$
                vCurCurs.ParaStyleName = sCurStyle
            End Select

        ElseIf sCurStyle = onlyStyle$ Then
            Select Case sPrevStyle
                Case firstStyle$, midStyle$
                    sCurStyle = midStyle$
                    vCurCurs.ParaStyleName = sCurStyle

                Case lastStyle$
                    sCurStyle = lastStyle$
                    vCurCurs.ParaStyleName = sCurStyle
                    vPrevCurs.ParaStyleName = midStyle$

                Case onlyStyle$
                    sCurStyle = lastStyle$
                    vCurCurs.ParaStyleName = sCurStyle
                    vPrevCurs.ParaStyleName = firstStyle$
            End Select

        Else
            Select Case sPrevStyle
                Case firstStyle$
                    vPrevCurs.ParaStyleName = onlyStyle$

                Case midStyle$
                    vPrevCurs.ParaStyleName = lastStyle$
            End Select
        End Select
    End If
End Sub

```

```

        End Select
    End If

    REM Done with the work so advance the trailing cursor
    vPrevCurs.gotoNextParagraph(False)
    vPrevCurs.gotoEndOfParagraph(True)
    sPrevStyle = vPrevCurs.ParStyleName
    Loop Until NOT vCurCurs.gotoNextParagraph(False)
End Sub

```

It is easy to modify the macro in *Listing 7.50* to format, and the macro would be much smaller. I lack the time, so I will not.

7.16.2. Is the cursor in the last paragraph

The cursor movements can be used to determine if a cursor is in the last paragraph. The following macro assumes that the answer is no unless the cursor is in the documents primary text object.

Listing 7.51: Is a cursor in the last paragraph?

```

Function IsCursorInlastPar(oCursor, oDoc)
    Dim oTC

    IsCursorInlastPar = False
    If EqualUNOObjects(oCursor.getText(), oDoc.getText()) Then
        oTC = oCursor.getText().createTextCursorByRange(oCursor)
        IsCursorInlastPar = NOT oTC.gotoNextParagraph(false)
    End If
End Function

```

7.16.3. What does it mean to enumerate text content?

Enumerating objects means that you visit each object. A Text object can enumerate its contained text content. A text object enumerates paragraphs and text tables. In other words, a text object can individually return each paragraph and text table that it contains (see *Listing 7.52*).

Listing 7.52: Enumerate paragraph level text content.

```

Sub EnumerateParagraphs
    REM Author: Andrew Pitonyak
    Dim oParEnum      'Enumerator used to enumerate the paragraphs
    Dim oPar          'The enumerated paragraph

    REM Enumerate the paragraphs.
    REM Tables are enumerated along with paragraphs
    oParEnum = ThisComponent.getText().createEnumeration()

```

```

Do While oParEnum.hasMoreElements()
    oPar = oParEnum.nextElement()

    REM This avoids the tables. Add an else statement if you want to
    REM process the tables.
    If oPar.supportsService("com.sun.star.text.Paragraph") Then
        MsgBox oPar.getString(), 0, "I found a paragraph"
    ElseIf oPar.supportsService("com.sun.star.text.TextTable") Then
        Print "I found a TextTable"
    Else
        Print "What did I find?"
    End If
Loop
End Sub

```

Each text paragraph is able to enumerate its content. Enumeration can be used to find fields, bookmarks, and all sorts of other text content. It can, however, only enumerate content anchored in the paragraph. A graphic that is anchored to a page, is not included in the enumeration. “Consider a **simple** text document that contains only *this* paragraph.”

Listing 7.53: Enumerate paragraph level text content.

```

Sub EnumerateContent
    REM Author: Andrew Pitonyak
    Dim oParEnum      'Enumerator used to enumerate the paragraphs
    Dim oPar          'The enumerated paragraph
    Dim oSectionEnum  'Enumerator used to enumerate the text sections
    Dim oSection      'The enumerated text section
    Dim s As String   'Contains the enumeration
    Dim i As Integer  'Count the paragraphs

    REM Enumerate the paragraphs.
    REM Tables are enumerated along with paragraphs
    oParEnum = ThisComponent.getText().createEnumeration()
    Do While oParEnum.hasMoreElements()
        oPar = oParEnum.nextElement()

        REM This avoids the tables. Add an else statement if you want to
        REM process the tables.
        If oPar.supportsService("com.sun.star.text.Paragraph") Then

            i = i + 1 : s = ""
            REM Now, enumerate the text sections and look for graphics that
            REM are anchored to a character, or as a character.
            oSectionEnum = oPar.createEnumeration()
            Do While oSectionEnum.hasMoreElements()
                oSection = oSectionEnum.nextElement()

                If oSection.TextPortionType = "Text" Then

```

```

    REM This is a simply text object!
    s = s & oSection.TextPortionType & " : "
    s = s & oSection.getString() & CHR$(10)
Else
    s = s & oSection.TextPortionType & " : " & CHR$(10)
End If
Loop
MsgBox s, 0, "Paragraph " & i
End If
Loop
End Sub

```

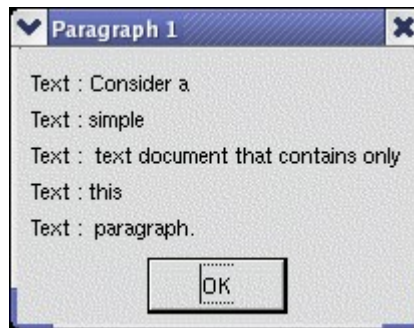


Figure 7.1: Enumerated paragraph.

A more complicated example is shown in Listing 7.54 in the next section, which enumerates graphics content. I also have more coverage in my book!

Can you replace a section text without affecting the attributes? Unfortunately, this is difficult. Inserted text uses the attributes of the text to the left of the insertion point. I tried to be clever:

1. Create a text cursor covering the section.
2. Insert text at the end.
3. Set the cursor text to be empty.

When the new text was inserted, the cursor expanded its range to include all of the text. An obvious solution is to then move the text cursor left by the length of the inserted text (exercise). Another solution, is as follows:

```

Dim oStart : oStart = oSection.getStart()
Dim oEnd   : oEnd   = oSection.getEnd()
Dim oText  : oText  = oStart.getText()
Dim oCurs  : oCurs  = oText.createTextCursorByRange(oStart)
Dim SecLen%: SecLen = Len(oSection.getString())
oText.insertString ( oEnd, "XyzzyX", False)
oCurs.goRight (SecLen, True)
oCurs.setString("")

```

I do not like this solution, but it works.

7.16.4. Enumerating text and finding text content

The primary reason to enumerate text content is to export the document. I was recently asked how to recognize graphics objects embedded in the text. The FindGraphics macro finds graphics objects that are anchored to a paragraph, anchored to a character, and inserted as a character. This does not find images anchored to the page.

The FindGraphics routine finds TextGraphicObjects and the GraphicObjectShapes. The TextGraphicObject is designed to be embedded into a text object and is used with **Insert > Graphics** for a Writer document. I can double click on a TextGraphicObject and numerous properties are presented for the object; this is not the case for a GraphicObjectShape.

Listing 7.54: Find graphics embedded in the text.

```
Sub FindGraphics
    REM Author: Andrew Pitonyak
    Dim oParEnum      'Enumerator used to enumerate the paragraphs
    Dim oPar          'The enumerated paragraph
    Dim oSectEnum     'Enumerator used to enumerate the text sections
    Dim oSect         'The enumerated text section
    Dim oCEnum        'Enum content, such as graphics objects
    Dim oContent      'The enumerated content
    Dim msg1$, msg2$, msg3$, msg4$
    Dim textGraphService$, graphicService$, textCService$

    textGraphService$ = "com.sun.star.text.TextGraphicObject"
    graphicService$    = "com.sun.star.drawing.GraphicObjectShape"
    textCService$      = "com.sun.star.text.TextContent"

    msg1$ = "Found a TextGraphicObject anchored to a Paragraph: URL = "
    msg2$ = "Found a GraphicObjectShape anchored to a Paragraph: URL = "
    msg3$ = "Found a TextGraphicObject anchored " & _
            "to or as a character: URL = "
    msg4$ = "Found a GraphicObjectShape anchored " & _
            "to or as a character: URL = "

    REM Enumerate the paragraphs.
    REM Tables are enumerated along with paragraphs
    oParEnum = ThisComponent.getText().createEnumeration()
    Do While oParEnum.hasMoreElements()
        oPar = oParEnum.nextElement()

        REM This avoids the tables. Add an else statement if you want to
        REM process the tables.
        If oPar.supportsService("com.sun.star.text.Paragraph") Then

            REM If you want to see the types that are available for
            REM enumeration as content associated with this paragraph,
```

```

REM then look at the available service names.
REM MsgBox Join(oPar.getAvailableServiceNames(), CHR$(10)

REM Typically, I use an empty string to enumerate ALL content,
REM but this causes a runtime error here. If any graphics
REM images are present, then they are enumerated as TextContent.
oCEnum = oPar.createContentEnumeration(textCService$)
Do While oCEnum.hasMoreElements()
    oContent = oCEnum.nextElement()
    If oContent.supportsService(textGraphService$) Then
        Print msg1$ & oContent.GraphicURL
    ElseIf oContent.supportsService(graphicService$) Then
        Print msg2$ & oContent.GraphicURL
        ' EmbedLinkedGraphic(oContent)
    'Else
    ' Inspect(oContent)
    End If
Loop

REM Now, enumerate the text sections and look for graphics that
REM are anchored to a character, or as a character.
oSectEnum = oPar.createEnumeration()
Do While oSectEnum.hasMoreElements()
    oSect = oSectEnum.nextElement()

    If oSect.TextPortionType = "Text" Then
        REM This is a simply text object!
        'MsgBox oSect.TextPortionType & " : " & _
        '     CHR$(10) & oSect.getString()
    ElseIf oSect.TextPortionType = "Frame" Then

        REM Use an empty string to enumerate ALL of the content
        oCEnum = oSect.createContentEnumeration(textGraphService$)
        Do While oCEnum.hasMoreElements()
            oContent = oCEnum.nextElement()
            If oContent.supportsService(textGraphService$) Then
                Print msg3$ & oContent.GraphicURL
            ElseIf oContent.supportsService(graphicService$) Then
                Print msg4$ & oContent.GraphicURL
                ' EmbedLinkedGraphic(oContent)
            'Else
            ' Inspect(oContent)
            End If
        Loop
    End If
Loop
End If

```

```

    Loop
End Sub

```

7.16.5. But I only want to find the graphics objects

I tested this on a Writer document and it iterates through the graphics objects. This is certainly the fastest way to find graphic objects.

Listing 7.55: Find all items on a draw page.

```

Dim i As Integer
Dim oGraph
For i=0 To ThisComponent.Drawpage.getCount()-1
    oGraph = ThisComponent.Drawpage.getByIndex(i)
Next i

```

Tilman Kranz used this to write the following macro:

Listing 7.56: Export all graphics in current write document.

```

REM Author: tilde@tk-sls.de (Tilman Kranz)
Sub iterateGraphics
    Dim i As Integer
    Dim c As Object
    Dim g As Object
    Dim sPath As String

    ' FIXME: hardcoded export path
    sPath = "/home/tilman/test/"

    c = ThisComponent.currentController

    For i=0 To ThisComponent.Drawpage.getCount()-1
        g = ThisComponent.Drawpage.getByIndex(i)
        saveGraphic(g.Graphic, ConvertToURL(sPath & g.Name & ".jpg"))
    Next
End Sub

' save specific graphic arg1
' to url arg2
Sub saveGraphic(in_obj, out_url)
    Dim i(0) As New com.sun.star.beans.PropertyValue
    Dim a(1) As New com.sun.star.beans.PropertyValue
    Dim p As Object
    Dim g As Object

    ' One provider is sufficient, just pass it in as an argument.
    p = createUnoService("com.sun.star.graphic.GraphicProvider")

```



```

a(0).Name = "MimeType"
a(0).Value = "image/jpeg"
a(1).Name = "URL"
a(1).Value = out_url

p.storeGraphic(in_obj, a)
End Sub

```

7.16.6. Find a text field contained in the current paragraph?

CPH (Mr. Hennessy), wanted the text field at the start of the current paragraph.

1. It is possible to enumerate text fields, checking to see if it is anchored in the current paragraph.
2. A text cursor can enumerate content, but text fields are not included in the enumeration.
3. It is possible to move a cursor one character at a time through the current paragraph looking for fields (see Listing 7.57). A similar method is used to determine if the current cursor is contained in a text table or a cell (see Listing 7.1 and Listing 7.2).

Listing 7.57: *Use the cursor to find a text field.*

```

oTCurs.goToStartOfParagraph(False)
Do While oTCurs.goRight(1, False) AND NOT oTCurs.isEndOfParagraph()
  If NOT IsEmpty(oTCurs.TextField) Then
    Print "Found a field by moving the cursor through the text."
  End If
Loop

```

Enumeration of the text content from a text cursor will enumerate graphics, but it will NOT enumerate a text field.

Listing 7.58: *Enumerate text content.*

```

sTContentService = "com.sun.star.text.TextContent"
oEnum = oTCurs.createContentEnumeration(sTContentService)
Do While oEnum.hasMoreElements()
  oSect = oEnum.nextElement()
  Print "Enumerating TextContent: " & oSect.ImplementationName
Loop

```

The current paragraph can be enumerated from the text cursor.

Listing 7.59: *Start an enumeration from a text cursor.*

```

oEnum = oTCurs.createEnumeration()
Do While oEnum.hasMoreElements()
  v = oEnum.nextElement()
  oSecEnum = v.createEnumeration()

```

```

Do While oSecEnum.hasMoreElements()
    oSubSection = oSecEnum.nextElement()
    If oSubSection.TextPortionType = "TextField" Then
        REM Notice that the Textfield is accessed,
        REM you can also access a
        REM TextFrame, TextSection, or a TextTable.
        'Text field here
    ElseIf oSubSection.TextPortionType = "Frame" Then
        'Graphics here!
    End If
Loop
Loop

```

Putting it all together in a single program yields the following:

Listing 7.60: Find the graphics and text fields in the current paragraph.

```

Sub GetTextFieldFromParagraph
    Dim oEnum    'Cursor enumerator.
    Dim oSect    'Current Section.
    Dim s$       'Generic string variable.
    Dim oVCurs   'Holds the view cursor.
    Dim oTCurs   'Created text cursor.
    Dim oText    'Text object that contains the view cursor
    Dim sTContentService$

    sTContentService = "com.sun.star.text.TextContent"
    REM Only the view cursor knows where a line ends.
    oVCurs = ThisComponent.CurrentController.getViewCursor()

    REM Use the text object that contains the view cursor.
    oText = oVCurs.Text

    REM Require a text cursor so that you know where the paragraph ends.
    REM Too bad the view cursor is not a paragraph cursor.
    oTCurs = oText.createTextCursorByRange(oVCurs)
    oTCurs.gotoStartOfParagraph(False)
    oTCurs.gotoEndOfParagraph(True)

    REM This does NOT work to enumerate text fields,
    REM but it enumerates graphics.
    oEnum = oTCurs.createContentEnumeration(sTContentService)
    Do While oEnum.hasMoreElements()
        oSect = oEnum.nextElement()
        Print "Enumerating TextContent: " & oSect.ImplementationName
    Loop

    REM focus the cursor over the paragraph again.
    oTCurs.gotoStartOfParagraph(False)

```

```

oTCurs.gotoEndOfParagraph(True)
REM And this provides the paragraph!
oEnum = oTCurs.createEnumeration()
Dim v
Do While oEnum.hasMoreElements()
    v = oEnum.nextElement()
    Dim oSubSection
    Dim oSecEnum
    oSecEnum = v.createEnumeration()
    s = "Enumerating section type: " & v.ImplementationName
    Do While oSecEnum.hasMoreElements()
        oSubSection = oSecEnum.nextElement()
        s = s & CHR$(10) & oSubSection.TextPortionType
        If oSubSection.TextPortionType = "TextField" Then
            REM Notice how a Textfield is accessed, you can also access a
            REM TextFrame, TextSection, or a TextTable.
            s = s & " <== here is a text field "
            s = s & oSubSection.TextField.ImplementationName
        ElseIf oSubSection.TextPortionType = "Frame" Then
            s = s & " <== here is a Frame "
        End If
    Loop
    MsgBox s, 0, "Enumerate Single Paragraph"
Loop

REM Move the cursor one character at a time looking
REM for a text field.
oTCurs.gotoStartOfParagraph(False)
Do While oTCurs.goRight(1, False) AND NOT oTCurs.isEndOfParagraph()
    If NOT IsEmpty(oTCurs.TextField) Then
        Print "Found a field by moving the cursor through the text."
    End If
Loop
End Sub

```

7.17. Where is the Display Cursor?

No time to be descriptive, but here is the e-mail with Giuseppe Castagno [castagno@tecsa-srl.it] who had the ideas.

You do a lot of interesting things but I think that it is not correct. First of all, the get position looks like it is relative to the first position on the top that can contain text. If there is a header, it is relative to that and if there is no header then it is from the top of the text frame. It looks like the top margin will be from the top of the page to the first position that can contain text.

Your measurements of the footer position are a pretty neat idea because it tells you the offset from the top of the footer to the cursor. I was impressed; had not thought about it. On the

other hand, what if you increase the size of the footer? You do not take that into account I think.

You can probably do something more like:

Page height - top margin - cursor position

No need to move the cursor around.

```
Sub PrintCursorLocation
    Dim xDoc
    Dim xViewCursor
    Dim s As String

    xDoc = ThisComponent
    xViewCursor = xDoc.CurrentController.getViewCursor()
    s = xViewCursor.PageStyleName

    Dim xFamilyNames As Variant, xStyleNames As Variant
    Dim xFamilies
    Dim xStyle, xStyles

    xFamilies = xDoc.StyleFamilies
    xStyles = xFamilies.getByName("PageStyles")
    xStyle = xStyles.getByName(xViewCursor.PageStyleName)
    ' RunSimpleObjectBrowser(xViewCursor)

    Dim lHeight As Long
    Dim lWidth As Long
    lHeight = xStyle.Height
    lWidth = xStyle.Width

    s = "Page size is " & CHR$(10) & _
        "      " & CStr(lWidth / 100.0) & " mm By " & _
        "      " & CStr(lHeight / 100.0) & " mm" & CHR$(10) & _
        "      " & CStr(lWidth / 2540.0) & " inches By " & _
        "      " & CStr(lHeight / 2540.0) & " inches" & CHR$(10) & _
        "      " & CStr(lWidth * 72.0 / 2540.0) & " picas By " & _
        "      " & CStr(lHeight * 72.0 / 2540.0) & " picas" & CHR$(10)

    Dim dCharHeight As Double
    Dim iCurPage As Integer

    Dim dXCursor As Double
    Dim dYCursor As Double
    Dim dXRight As Double
    Dim dYBottom As Double
```

```

Dim dBottomMargin As Double
Dim dLeftMargin As Double

dCharHeight = xViewCursor.CharHeight / 72.0
iCurPage = xViewCursor.getPage()

Dim v
v = xViewCursor.getPosition()
dYCursor = (v.Y + xStyle.TopMargin)/2540.0 + dCharHeight / 2
dXCursor = (v.X + xStyle.LeftMargin)/2540.0
dXRight = (lWidth - v.X - xStyle.LeftMargin)/2540.0
dYBottom = (lHeight - v.Y - xStyle.TopMargin)/2540.0 - dCharHeight / 2
' Print "Left margin = " & xStyle.LeftMargin/2540.0

dBottomMargin = xStyle.BottomMargin / 2540.0
dLeftMargin = xStyle.LeftMargin / 2540.0
s = s & "Cursor is " & Format(dXCursor, "0.##") & " inches from left " &
CHR$(10)
s = s & "Cursor is " & Format(dXRight, "0.##") & " inches from right " &
CHR$(10)
s = s & "Cursor is " & Format(dYCursor, "0.##") & " inches from top " &
CHR$(10)
s = s & "Cursor is " & Format(dYBottom, "0.##") & " inches from bottom " &
CHR$(10)
s = s & "Left margin = " & dLeftMargin & " inches" & CHR$(10)
s = s & "Bottom margin = " & dBottomMargin & " inches" & CHR$(10)
s = s & "Char height = " & Format(dCharHeight, "0.####") & " inches" &
CHR$(10)

' RunSimpleObjectBrowser(xStyle)

' Dim dFinalX As Double
' Dim dFinalY As Double
' dFinalX = dXCursor + dLeftMargin
' dFinalY = (v.Y + xStyle.TopMargin)/2540 + dCharHeight / 2
' s = s & "Cursor in page is at (" & Format(dFinalX, "0.####") & ", " & _
' Format(dFinalY, "0.####") & ") in inches" & CHR$(10)

REM now check the footer!
' If xStyle.FooterIsOn Then
' v = IIF(iCurPage MOD 2 = 0, xStyle.FooterTextLeft, xStyle.FooterTextRight)
' If IsNull(v) Then v = xStyle.FooterText
' If Not IsNull(v) Then
' REM Save the postion
' Dim xOldCursor
' xOldCursor = xViewCursor.getStart()

```

```

'      xViewCursor.gotoRange(v.getStart(), false)
'      Print "footer position = " & CStr(xViewCursor.getPosition().Y/2540.0)
'      dFinalY = xViewCursor.getPosition().Y/2540.0 - dYCursor + dBottomMargin
'      xViewCursor.gotoRange(xOldCursor, false)
'      End If
'      Else
'      Print "No footer"
'      End If
'      s = s & "Cursor in page is at (" & Format(dFinalX, "0.####") & ", " & _
'      Format(dFinalY, "0.####") & ") in inches" & CHR$(10)
'      MsgBox s, 0, "Page information"
End Sub

```

As a side note, do not try this with lock controllers, or you will not obtain the correct position.

7.17.1. Which cursor should I use to delete a line or a paragraph

The view cursor knows about lines and pages, the regular cursors know about words, sentences, and paragraphs.

```

Sub DeleteCurrentLine
    Dim oVCurs
    oVCurs = ThisComponent.getCurrentController().getViewCursor()
    oVCurs.gotoStartOfLine(False)
    oVCurs.gotoEndOfLine(True)
    oVCurs.setString("")
End Sub

```

To delete the current paragraph, you require a paragraph cursor, which is not the view cursor. You only use the view cursor to find the current cursor position.

```

Sub DeleteParagraph
    Dim oCurs
    Dim oText
    Dim oVCurs
    oVCurs = ThisComponent.getCurrentController().getViewCursor()

    REM Get the text object from the cursor, it is safer than assuming
    REM that the high level document Text object contains the
    REM view cursor.
    oText = oVCurs.getText()
    oCurs = oText.createTextCursorByRange(oVCurs)
    oCurs.gotoStartOfParagraph(False)
    If oCurs.gotoNextParagraph(True) Then
        oCurs.setString("")
    Else
        REM Then we were already AT the last paragraph
        If oCurs.gotoPreviousParagraph(False) Then

```

```

        oCurs.gotoEndOfParagraph(False)
        oCurs.gotoNextParagraph(True)
        oCurs.gotoEndOfParagraph(True)
        oCurs.setString("")
    Else
        Rem There is one, and only one paragraph here
        REM Remove it
        oCurs.gotoStartOfParagraph(False)
        oCurs.gotoEndOfParagraph(True)
        oCurs.setString("")
    End If
End If
End Sub

```

7.17.2. Delete the current page

To delete an entire page, requires the view cursor, because only the view cursor knows where a page starts and ends. Consider the following simplistic example:

```

Sub removeCurrentPage()
    REM Author: Andrew Pitonyak
    Dim oVCurs
    Dim oCurs

    oVCurs = ThisComponent.getCurrentController().getViewCursor()
    If oVCurs.jumpToStartOfPage() Then
        oCurs = ThisComponent.getText().CreateTextCursorByRange(oVCurs)
        If (oVCurs.jumpToEndOfPage()) Then
            oCurs.gotoRange(oVCurs, True)
            oCurs.setString("")
        Else
            Print "Unable to jump to the end of the page"
        End If
    Else
        Print "Unable to jump to the start of the page"
    End If
End Sub

```

I have performed limited testing with this macro. If the page starts with a paragraph and ends with a paragraph, this may leave an extra empty paragraph in the text. This is most likely because the `jumpToEndOfPage()` method probably does not include the ending paragraph marker.

7.18. Insert an index or table of contents

The table of contents is simply another type of index. I have inserted sufficient comments that they should answer most of your questions as to how to do this.

```

Sub InsertATOC

```

```

REM Author: Andrew Pitonyak
Dim oCurs 'Used to insert the text content.
Dim oIndexes 'All of the existing indexes
Dim oIndex 'TOC if it exists and a new one if not
Dim i As Integer 'Find an existing TOC
Dim bIndexFound As Boolean 'Flag to track if the TOC was found
Dim s$

REM First, find an existing TOC if it exists. If so,
REM then this will simply be updated.
oIndexes = ThisComponent.getDocumentIndexes()
bIndexFound = False
For i = 0 To oIndexes.getCount() - 1
    oIndex = oIndexes.getByIndex(i)
    If oIndex.supportsService("com.sun.star.text.ContentIndex") Then
        bIndexFound = True
        Exit For
    End If
Next

If Not bIndexFound Then
    Print "I did not find an existing content index"
    REM Perhaps you should create and insert a new one!
    REM Notice that this MUST be created by the document that
    REM will contain the index.
    S = "com.sun.star.text.ContentIndex"
    oIndex = ThisComponent.createInstance(s)

    REM On my system, these are the default values
    REM How do you want to create the index?
    REM CreateFromChapter = False
    REM CreateFromLevelParagraphStyles = False
    REM CreateFromMarks = True
    REM CreateFromOutline = False
    oIndex.CreateFromOutline = True

    REM You can set all sorts of other things such as the
    REM Title or Level

    oCurs = ThisComponent.getText().createTextCursor()
    oCurs.gotoStart(False)
    ThisComponent.getText().insertTextContent(oCurs, oIndex, False)
End If

REM Even the newly inserted index is not updated until right HERE!
oIndex.update()
End Sub

```


7.19. Inserting a URL into a Write document

Although a Calc document stores URLs in a URL text field, as shown elsewhere in this document, Write documents identify contained URLs based on character properties. A link becomes a link, when the HyperLinkURL property is set.

```
Sub InsertURLAtTextCursor
    Dim oText      'Text object for the current object
    Dim oVCursor   'Current view cursor

    oVCursor = ThisComponent.getCurrentController().getViewCursor()
    oText = oVCursor.getText()
    oText.insertString(oVCursor, "andrew@pitonyak.org", True)
    oVCursor.HyperLinkTarget = "mailto:andrew@pitonyakorg"

    oVCursor.HyperLinkURL = "andrew@pitonyak.org"
    ' oVCursor.HyperLinkName = "andrew@pitonyak.org"
    ' oVCursor.UnvisitedCharStyleName = "Internet Link"
    ' oVCursor.VisitedCharStyleName = "Visited Internet Link"
End Sub
```

7.20. Sorting Text

A text cursor can be used to sort data in a Write document.

```
Sub SortTextInWrite
    Dim oText      'Text object for the current object
    Dim oVCursor   'Current view cursor
    Dim oCursor     'Text cursor
    Dim oSort
    Dim i%
    Dim s$

    REM Assume that we want to sort the selected text!
    REM Unfortunately, the view cursor can NOT create a sort descriptor.
    REM Create a new text cursor, that can create a sort descriptor.
    oVCursor = ThisComponent.getCurrentController().getViewCursor()
    oText = oVCursor.getText()
    oCursor = oText.createTextCursorByRange(oVCursor)
    oSort = oCursor.createSortDescriptor()

    ' On Error Resume Next
    ' For i = LBound(oSort) To UBound(oSort)
    '     s = s & "(" & oSort(i).Name & ", "
    '     s = s & oSort(i).Value
    '     s = s & ")" & CHR$(10)
    ' Next
    ' MsgBox s, 0, "Sort Properties"
    oCursor.sort(oSort)
End Sub
```

End Sub

Use the SortDescriptor2, the SortDescriptor has been deprecated. See:

<http://api.openoffice.org/docs/common/ref/com/sun/star/text/TextSortDescriptor2.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/table/TableSortDescriptor2.html>

The default values for the sort descriptor seems to be: (IsSortInTable, False), (Delimiter, 32), (IsSortColumns, True), (MaxSortFieldsCount, 3), (SortFields,). To set the sort order and Locale, you need to modify the SortFields.

7.21. Outline numbering

I have been asked a few times how to change which styles are used for outline numbering. The first step in setting the outline numbering is to have a paragraph style. The macro in Listing 7.61 creates a custom paragraph style with the specified character height. The parent style is set to “Heading”. You will want to change the paragraph style to fit your requirements.

Listing 7.61: *Create a custom paragraph style.*

```
Function CreateParaStyle(oDoc, NewParaStyle, nCharHeight%)
    Dim pFamilies, pStyle, pParaStyles

    pFamilies = oDoc.StyleFamilies
    pParaStyles = pFamilies.GetByName("ParagraphStyles")

    If Not pParaStyles.HasByName(NewParaStyle) Then
        pStyle = oDoc.CreateInstance("com.sun.star.style.ParagraphStyle")
        pStyle.SetParentStyle("Heading")
        pStyle.CharHeight = nCharHeight
        pParaStyles.InsertByName(NewParaStyle, pStyle)
    End If
End Function
```

After you have created your paragraph styles, or at least chosen the paragraph styles to use, you can set the numbering. The macro in Listing 7.62 accepts an array of paragraph style names and sets the outline numbering to use these styles.

The outline numbering information is stored in the chapter numbering rules object. Obtain each rule using `getByIndex()`, which returns an array of properties. Inspect the name of each property to find the property that you want to change. Each property is copied OUT of the array. If the properties were stored as an UNO Service rather than an UNO structure, then a reference would be copied from the array rather than copy (I explain the copy by value rather than copy by reference behavior in my book). After finding and modifying the appropriate properties, the rule is copied back into the rules object.

Listing 7.62: *Set the outline numbering*

```
Sub SetNumbering(sNames())
    Dim i%, j%
```

```

Dim oRules
Dim oRule()
Dim oProp

oRules = ThisComponent.getChapterNumberingRules()
For i = 0 To UBound(sNames())
    If i >= oRules.getCount() Then Exit Sub
    oRule() = oRules.getByIndex(i)
    REM I do not set the following:
    REM Adjust, StartWith, LeftMargin,
    REM SymbolTextDistance, FirstLineOffset
    For j = LBound(oRule()) To Ubound(oRule())
        REM oProp is only a copy of the property.
        REM You must assign the property back into the array.
        oProp = oRule(j)
        Select Case oProp.Name
            Case "HeadingStyleName"
                oProp.Value = sNames(i)
            Case "NumberingType"
                oProp.Value = com.sun.star.style.NumberingType.ARABIC
            Case "ParentNumbering"
                oProp.Value = i + 1
            Case "Prefix"
                oProp.Value = ""
            Case "Suffix"
                oProp.Value = " "
            'Case "CharStyleName"
            '    oProp.Value =
        End Select
        oRule(j) = oProp
    Next
    oRules.replaceByIndex(i, oRule())
Next
End Sub

```

If the array passed to Listing 7.62 contains three entries, then only the first three entries in the outline numbering are modified. Use the code as a starting point. Listing 7.63 demonstrates how to use both Listing 7.61 and Listing 7.62.

Listing 7.63: Set and use the outline numbering

```

Sub SetOutlineNumbering
    CreateParaStyle(ThisComponent, "_New_Heading_1", 16)
    CreateParaStyle(ThisComponent, "_New_Heading_2", 14)
    setNumbering(Array("_New_Heading_1", "_New_Heading_2"))
End Sub

```

7.22. Configure Outline numbering

What if you want to number your headings as "1.2. blah blah".

Listing 7.64: Set numbering method for outline numbering

```
Sub CheckOutLine()  
    Dim i%, j%  
    Dim oRules  
    Dim oRule()  
    Dim oProp  
  
    oRules = ThisComponent.getChapterNumberingRules()  
    For i = 0 To oRules.getCount() - 1  
        oRule() = oRules.getByIndex(i)  
        REM I do not set the following:  
        REM Adjust, StartWith, LeftMargin,  
        REM SymbolTextDistance, FirstLineOffset  
        For j = LBound(oRule()) To Ubound(oRule())  
            REM oProp is only a copy of the property.  
            REM You must assign the property back into the array.  
            oProp = oRule(j)  
            Select Case oProp.Name  
            Case "HeadingStyleName"  
                'oProp.Value = sNames(i)  
            Case "NumberingType"  
                oProp.Value = com.sun.star.style.NumberingType.ARABIC  
            Case "ParentNumbering"  
                oProp.Value = i + 1  
            Case "Prefix"  
                oProp.Value = ""  
            Case "Suffix"  
                oProp.Value = "."  
            'Case "CharStyleName"  
            '    oProp.Value =  
            End Select  
            oRule(j) = oProp  
        Next  
        oRules.replaceByIndex(i, oRule())  
    Next  
End Sub
```

7.23. Numbering paragraphs – not outline numbering

I rarely use outline numbering because it is too limiting. I use two numbering styles. `_Chapters` is for the standard text, and it provides the primary numbering you see in this document. I also have a numbering called `_Appendix`, which numbers the Heading 1 style as Appendix A, Appendix B, etc. The Heading 2 style numbers as A.1, A.2, etc. The advantage

is that I use the standard heading styles for my headings, so the table of contents and outlines build correctly at the correct levels.

Unfortunately, some of the development builds for OOo 3.0 introduced an error with numbering, which broke numbering. My first attempt to fix the problem was to troll through all of the paragraphs, clear the numbering by setting the NumberingStyleName property to an empty string, and then setting it back. Unfortunately, this did not work. I had to use a dispatch to make this work.

```
Sub RefreshHeadingNumStyle()
    REM Author: Andrew Pitonyak
    Dim oParEnum      'Enumerator used to enumerate the paragraphs
    Dim oPar          'The enumerated paragraph
    Dim s$

    Dim oFrame
    Dim oDisp
    oFrame = ThisComponent.CurrentController.Frame
    oDisp = createUnoService("com.sun.star.frame.DispatchHelper")

    'Family number probably changes based on other things!
    Dim args1(1) as new com.sun.star.beans.PropertyValue
    args1(0).Name = "Template"
    args1(0).Value = "_Chapters"
    args1(1).Name = "Family"
    args1(1).Value = 16

    REM Enumerate the paragraphs.
    REM Tables are enumerated along with paragraphs
    oParEnum = ThisComponent.getText().createEnumeration()
    Do While oParEnum.hasMoreElements()
        oPar = oParEnum.nextElement()

        REM This avoids the tables. Add an else statement if you want to
        REM process the tables.
        If oPar.supportsService("com.sun.star.text.Paragraph") Then
            'MsgBox oPar.getString(), 0, "I found a paragraph"
            If Len(oPar.NumberingStyleName) > 0 Then
                If InStr(oPar.ParaStyleName, "Heading ") = 1 Then
                    If RefreshStyle(oPar.NumberingStyleName) Then
                        args1(0).Value = oPar.NumberingStyleName
                        oPar.NumberingStyleName = ""
                        ThisComponent.getCurrentController().select(oPar)
                        oDisp.executeDispatch(oFrame, ".uno:StyleApply", "", 0, args1())
                    EndIf
                End If
            End If
        ElseIf oPar.supportsService("com.sun.star.text.TextTable") Then
```

```

        'Print "I found a TextTable"
    Else
        'Print "What did I find?"
    End If
Loop
End Sub

Function RefreshStyle(s$) As boolean
    If s = "_Chapters" OR s = "_Appendix" Then
        RefreshStyle = True
    Else
        RefreshStyle = False
    End If
End Function

```

7.24. Insert a table of contents (TOC) or other index.

The following code inserts a table of contents TOC into a document. If the TOC already exists, then the update method is called on the existing index. Use the dispose method to remove an existing index from the document.

```

Sub InsertATOC
    REM Author: Andrew Pitonyak
    Dim oCurs                'Used to insert the text content.
    Dim oIndexes              'All of the existing indexes
    Dim oIndex                'TOC if it exists and a new one if not
    Dim i As Integer          'Find an existing TOC
    Dim bIndexFound As Boolean 'Flag to track if the TOC was found
    Dim s$

    REM First, find an existing TOC if it exists. If so,
    REM then this will simply be updated.
    oIndexes = ThisComponent.getDocumentIndexes()
    bIndexFound = False
    For i = 0 To oIndexes.getCount() - 1
        oIndex = oIndexes.getByIndex(i)
        If oIndex.supportsService("com.sun.star.text.ContentIndex") Then
            bIndexFound = True
            Exit For
        End If
    Next

    If Not bIndexFound Then
        Print "I did not find an existing content index"
        REM Perhaps you should create and insert a new one!
        REM Notice that this MUST be created by the document that
        REM will contain the index.
        S = "com.sun.star.text.ContentIndex"
    End If
End Sub

```

```

oIndex = ThisComponent.createInstance(s)

REM On my system, these are the default values
REM How do you want to create the index?
REM CreateFromChapter = False
REM CreateFromLevelParagraphStyles = False
REM CreateFromMarks = True
REM CreateFromOutline = False
oIndex.CreateFromOutline = True

REM You can set all sorts of other things such as the
REM Title or Level

oCurs = ThisComponent.getText().createTextCursor()
oCurs.gotoStart(False)
ThisComponent.getText().insertTextContent(oCurs, oIndex, False)
End If

REM Even the newly inserted index is not updated until right HERE!
oIndex.update()
End Sub

```

7.25. Text sections

A TextSection is a range of complete paragraphs contained a text object. The content may be the content of a link into another document, a link from the same document, or the result of a DDE operation. TextSection instances can be linked from, and to, other texts. The contents of a text section are enumerated along with regular text and can be traversed with a text cursor. The text is traversed even if it is marked as not visible, which you can check with the oSect.IsVisible property.

Listing 7.65: Display the text in the first text section

```

REM Author: Andrew Pitonyak
Sub GetTextFromTextSection
    Dim oSect
    Dim oCurs

    If ThisComponent.getTextSections().getCount() = 0 Then
        Print "There are no text sections"
        Exit Sub
    End If

    REM You can get a text section by name or index.
    'oSect = ThisComponent.getTextSections().getByName("Special")
    oSect = ThisComponent.getTextSections().getByIndex(0)

    REM Create a cursor and move the cursor to the text section.

```

```

oCurs = ThisComponent.getText().createTextCursor()
oCurs.gotoRange(oSect.getAnchor(), False)

REM In this example, I assume that there is text after
REM the text section. If there is not, then this is
REM an infinite loop. You had better verify that
REM gotoNextParagraph does not return False.
Do While NOT IsEmpty(oCurs.TextSection)
    oCurs.gotoNextParagraph(False)
Loop

oCurs.gotoPreviousParagraph(False)
oCurs.gotoRange(oSect.getAnchor(), True)
MsgBox oCurs.getString()
End Sub

```

7.25.1. Insert a text section, setting columns and widths

Insert a text section with two columns. Then, set a ½ space between the columns. I could explain, but I am tired.

Listing 7.66: Display the text in the first text section

```

Sub AddTextSection
    Dim oSect
    Dim sName$
    Dim oVC
    Dim oText
    Dim oCols
    Dim s$
    sName = "ADPSection"
    If ThisComponent.getTextSections().hasByName(sName) Then
        Print "Text section " & sName & " Already exists"
        oSect = ThisComponent.getTextSections().getByName(sName)
    Else
        REM Create a text section at the cursor
        oVC = ThisComponent.getCurrentController().getViewCursor()
        oText = oVC.getText()
        REM Insert a new paragraph
        oText.insertControlCharacter(oVC, _
            com.sun.star.text.ControlCharacter.LINE_BREAK, False)
        REM Insert a new paragraph and select it
        oText.insertControlCharacter(oVC, _
            com.sun.star.text.ControlCharacter.LINE_BREAK, True)
        s = "com.sun.star.text.TextSection"
        oSect = ThisComponent.createInstance(s)
        oSect.setName(sName)
        REM Now, create the columns...
        s = "com.sun.star.text.TextColumns"
    End If
End Sub

```



```

oCols = ThisComponent.CreateInstance(s)
oCols.setColumnCount(2)
oSect.TextColumns = oCols
oText.InsertTextContent(oVC, oSect, True)
oText.InsertString(oVC, "This is new text. " & _
    "I suppose that I could count and repeat myself as " & _
    "an example of how text can go on and on and on. " & _
    "I suppose that I could count and repeat myself as " & _
    "an example of how text can go on and on and on. " & _
    "I suppose that I could count and repeat myself as " & _
    "an example of how text can go on and on and on. " & _
    "I suppose that I could count and repeat myself as " & _
    "an example of how text can go on and on and on. " & _
    "I suppose that I could count and repeat myself as " & _
    "an example of how text can go on and on and on. " & _
    "I suppose that I could count and repeat myself as " & _
    "an example of how text can go on and on and on. " & _
    "And finally I will stop.", True)
Print "Created the text section"
End If
oCols = oSect.TextColumns
Dim OC()
OC() = oCols.getColumns()
REM Set the right margin on the first column to 1/4 inch.
OC(0).RightMargin = 635
REM Set the left margin on the second column to 1/4 inch.
OC(1).LeftMargin = 635
oCols.setColumns(OC())
oSect.TextColumns = oCols
End Sub

```

7.26. Footnotes and Endnotes

There are two types of footnotes, automatic and user specified label. Automatic footnotes automatically set the footnote number sequentially; it is not just a field. If the label is specified, then the footnote is not included in the automatic numbering.

Listing 7.67: It is easy to enumerate the footnotes.

```

oNotes = ThisComponent.getFootnotes()
For i = 0 To oNotes.getCount() - 1
    oNote = oNotes.getByIndex(i)
Next

```

Use getString() and setString() to get and set the footnote text.

Use getLabel() and setLabel() to return the user specified label. If this is an empty string, then this is an automatic footnote. Convert between the two footnote types by setting the label to a zero length or non-zero length string.

Use `getEndNotes()` on the document to obtain end notes rather than footnotes.

Do not ignore the settings (`getFootnoteSettings`). Here, you can specify how footnotes are numbered, restart numbering for each chapter, for example.

7.27. Redlines

I have seen and posed questions to the developers on this topic a few times over the last five years. I have NEVER seen an answer. I assume, therefore, that it is not possible to use the API to accept or reject individual redlines.

While recording changes, they are saved as “redlines”. Although you can easily enumerate the redlines, I have not found a way to do anything useful with them. Specifically, I am not able to obtain the text, or move a cursor to the redline, even though I can obtain the “start” and “end” of the redline.

Listing 7.68: Enumerate redlines.

```
oEnum = ThisComponent.redlines.createEnumeration()  
Do While oEnum.hasMoreElements()  
    oRedLine = oEnum.nextElement()  
    'oRedLine.RedlineType  
    'oredline.redlineauthor  
    'oredline.redlinecomment  
loop
```

I am, however, able to do some work by enumerating the text content (see Listing 7.53). The following simple macro ignores things such as tables and frames, but it provides some insight into how to deal with redlines.

```
Sub EnumerateRedlineContent  
    REM Author: Andrew Pitonyak  
    Dim oParEnum      'Enumerator used to enumerate the paragraphs  
    Dim oPar          'The enumerated paragraph  
    Dim oSectionEnum  'Enumerator used to enumerate the text sections  
    Dim oSection      'The enumerated text section  
    Dim oCurs         'Track the redlines.  
    Dim oText  
  
    oParEnum = ThisComponent.getText().createEnumeration()  
    Do While oParEnum.hasMoreElements()  
        oPar = oParEnum.nextElement()  
        If oPar.supportsService("com.sun.star.text.Paragraph") Then  
            oSectionEnum = oPar.createEnumeration()  
            Do While oSectionEnum.hasMoreElements()  
                oSection = oSectionEnum.nextElement()  
                If oSection.TextPortionType = "Redline" Then  
                    If oSection.IsStart Then  
                        'At a start, so create the text cursor.  
                    End If  
                End If  
            End Do  
        End If  
    End Do
```

```

        oText = oSection.getText()
        oCurs = oText.createTextCursorByRange(oSection.getStart())
    Else
        'Move cursor to the end of the redline.
        oCurs.gotoRange(oSection.getEnd(), True)
        Print oSection.RedlineType & " " & oCurs.getString()
    End If
End If
Loop
End If
Loop
End Sub

```

A quick test, that did not check things such as formatting changes, seemed to indicate that I can delete a section by setting the string to an empty string.

```

If oSection.RedlineType = "Delete" Then
    'Delete the range!
    oCurs.setString("")
End If

```

I have not, however, found any other method to remove a redline using a macro. I have also not found a method to accept an individual redline.

7.28. Formulas

A formula in a Write document is really a Math document embedded in the document. The best way to insert a formula (ignoring the hack in Listing 10.12) is shown in Listing 7.69.

Listing 7.69: Insert formula at view cursor.

```

Sub InsertEquationAtViewCursor(oDoc, sFormula$)
    Dim oVC
    Dim oObj

    oVC = oDoc.CurrentController.getViewCursor()
    oVC.gotoRange(oVC.getEnd(), False)
    oObj = oDoc.CreateInstance("com.sun.star.text.TextEmbeddedObject")
    oObj.CLSID = "078B7ABA-54FC-457F-8551-6147e776a997"
    oObj.AnchorType = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
    oVC.Text.insertTextContent(oVC, oObj, False)
    oObj.EmbeddedObject.formula = sFormula
End Sub

```

To demonstrate, the following macro inserts multiple simple formulas into the document at the current cursor location.

Listing 7.70: Insert multiple formulas.

```

Sub InsertSimpleEquations(oDoc, iNumToInsert As Integer)
    Dim sPropName As String

```

```

Dim aValue As Any
Dim iCount As Integer
Dim iSeed As Integer
Dim sFormula As String
Dim iBuildString As Integer
Dim iA As Integer
Dim iB As Integer
Dim iC As Integer

iSeed = 0
For iCount = 1 to iNumToInsert

    sFormula = "{"
    iA = Int(20*Rnd(iSeed)+1)
    iB = Int(20*Rnd(iSeed)+1)
    iC = Int(20*Rnd(iSeed)+1)
    sFormula = sFormula + Str(iA) + "x +" + Str(iB) + " =" + iC + "}"
    InsertEquationAtViewCursor(oDoc, sFormula)
    InsertNewLines(oDoc, 1)
Next iCount
End Sub

Sub InsertNewLines(oDoc, nNumLines As Integer)
    Dim oText
    Dim oViewCursor
    Dim oCursor
    Dim i As Integer
    Dim oVC

    REM Get the view cursor and move the cursor to the end
    REM of the selected range in case a range is selected.
    REM This is likely to fail if cells in a table are selected.
    oVC = oDoc.CurrentController.getViewCursor()
    oVC.gotoRange(oVC.getEnd(), False)
    For i = 0 To nNumLines
        oText = oVC.getText()
        oText.insertControlCharacter(oVC, _
            com.sun.star.text.ControlCharacter.LINE_BREAK, False)
    Next
End Sub

```

A CLSID (Class Identifier) is a globally unique identifier that identifies a COM class object. Formulas are embedded as objects. You can enumerate embedded objects and check the CLSID for formula objects.

Listing 7.71: Enumerate embedded objects.

```

Sub LookAtEmbeddedObjects
    Dim oDoc

```

```

Dim oEmbed
Dim oObj
Dim i As Integer
Dim s As String

oDoc = ThisComponent
oEmbed = oDoc.getEmbeddedObjects()
s = ""
For i = 0 To oEmbed.getCount()-1
    oObj = oEmbed.getByIndex(i)
    If oObj.supportsService("com.sun.star.text.TextEmbeddedObject") Then
        If oObj.CLSID = "078B7ABA-54FC-457F-8551-6147e776a997" Then
            s = s & oObj.getEmbeddedObject().Formula & CHR$(10)
        End If
    End If
Next
MsgBox s
End Sub

```

I was asked to write a macro that compared two documents. Each line contained a formula as either text, or as a formula object. Enumerating the paragraphs is easy. Perform a compare one line at a time.

Listing 7.72: Enumerate paragraphs comparing formulas.

```

Function compareDocs(oDoc1, oDoc2) As Boolean
    Dim oParEnum1, oParEnum2
    Dim oPar1, oPar2
    Dim f1$, f2$
    Dim i%

    compareDocs = True
    If IsNull(oDoc1) OR IsEmpty(oDoc1) OR IsNull(oDoc2) OR IsEmpty(oDoc2) Then
        Exit Function
    End If

    oParEnum1 = oDoc1.getText().createEnumeration()
    oParEnum2 = oDoc2.getText().createEnumeration()
    i = 1
    Do While oParEnum1.hasMoreElements() AND oParEnum2.hasMoreElements()
        oPar1 = oParEnum1.nextElement()
        oPar2 = oParEnum2.nextElement()
        If oPar1.supportsService("com.sun.star.text.Paragraph") AND _
            oPar2.supportsService("com.sun.star.text.Paragraph") Then
            f1 = GetParOrFormula(oPar1)
            f2 = GetParOrFormula(oPar2)
            If (f1 <> f2) Then
                MsgBox "Formulas do not match at line " & i & CHR$(10) & _
                    "Doc 1: " & f1 & CHR$(10) & _

```

```

        "Doc 2: " & f2, 48, "Warning"
    compareDocs = False
    'Exit Function
End If
End If
i = i + 1
Loop
End Function

```

Call `getString()` on a paragraph object to obtain the paragraph as a string. Unfortunately, formulas are not returned as text. Paragraphs are embedded as Frames. You can then enumerate formula properties (Thanks to Bernard Marcelly).

Listing 7.73: Return paragraph string, or formula string.

```

Function GetParOrFormula(oPar) As String
    Dim oEnum      'Enumeration of text sections.
    Dim oSect
    Dim oTcEnum
    Dim oTc
    Dim s$
    Dim sFormulas$ : sFormulas = "com.sun.star.formula.FormulaProperties"

    REM Assume no formula, so set the string to the entire paragraph.
    GetParOrFormula = oPar.getString()
    oEnum = oPar.createEnumeration()
    Do While oEnum.hasMoreElements()
        oSect = oEnum.nextElement()
        If oSect.supportsService("com.sun.star.text.TextPortion") Then
            If oSect.TextPortionType = "Frame" Then
                s = ""
                oTcEnum = oSect.createContentEnumeration(sFormulas)
                REM This funds multiple formulas if they are present in the paragraph
                Do While oTcEnum.hasMoreElements()
                    oTc = oTcEnum.nextElement()
                    s = s & oTc.getEmbeddedObject().Formula
                Loop
                GetParOrFormula = s
            End If
        End If
    Loop
End Function

```

7.29. Cross references

I decided to write this section because I cannot find much information regarding references in a document. It is possible to reference the following items:

- Number range variable such as a reference to table or drawing caption. Number range variables are stored in text master fields.
- Manually set reference. Use `getReferenceMarks` to obtain a list of manually set references.
- Bookmark. Use `getBookMarks` to obtain a list of bookmarks.
- Heading. The only method I know to find headings is to enumerate the text content.

When you look at master fields, you find names such as:

```
com.sun.star.text.FieldMaster.SetExpression.Illustration
com.sun.star.text.FieldMaster.SetExpression.Table
com.sun.star.text.FieldMaster.SetExpression.Text
com.sun.star.text.FieldMaster.SetExpression.Drawing
com.sun.star.text.FieldMaster.User
```

See section 5.18.3 Master Fields for a macro to enumerate the macro fields. For each master field, the `DependentTextFields` property provides an array of the dependent fields. The dependent fields are the instances that can be referenced, as opposed to the references.

I enumerate reference marks and bookmarks as follows:

```
MsgBox Join(ThisComponent.getReferenceMarks().getElementNames(), CHR(10))
MsgBox Join(ThisComponent.getBookMarks().getElementNames(), CHR(10))
```

What can you find by enumerating the contained text fields (`getTextFields`)?

- You can find set expressions for number ranges. Number range variables are associated to a master text field. The child text fields contain equations such as “Figure + 1”.
- Page numbers.
- DocInfo fields such as Revision and Subject.
- References to all of the reference types listed above (manual, bookmarks, etc.).

7.29.1. Enumerate text fields

Fields that reference headings have names such as `__RefHeading__29453202`. I am certain that I missed many field types, but here is a macro to enumerate them:

```
REM Author: Andrew Pitonyak
Sub EnumerateTextFields(Optional oUseDoc)
    Dim oDoc      'Document to use.
    Dim oEnum      'Enumeration of the text fields.
    Dim oField     'Enumerated text field.
    Dim s$         'Generic string variable.
    Dim n%         'Count the number of text fields.
    Dim nUnknown% 'Count the unexpected field types.
```

```

Dim bDisplayExpressions As Boolean
bDisplayExpressions = True

If IsMissing(oUseDoc) Then
    oDoc = ThisComponent
Else
    oDoc = oUseDoc
End If

oEnum = oDoc.getTextFields().createEnumeration()
If IsNull(oEnum) Then
    Print "getTextFields().createEnumeration() returns NULL"
    Exit Sub
End If

Do While oEnum.hasMoreElements()
    oField = oEnum.nextElement()
    If oField.supportsService("com.sun.star.text.TextField.Input") Then
        REM If you update Content, use oDoc.TextFields.refresh() afterwards.
        n = n + 1
        s = s & "Input (" & oField.getPropertyValue("Hint") & ", " & _
            oField.getPropertyValue("Content") & ")" & CHR$(10)
    ElseIf oField.supportsService("com.sun.star.text.TextField.User") Then
        REM You can update the text master field, but be certain to use
        REM oDoc.TextFields.refresh() afterwards.
        n = n + 1
        s = s & "User (" & oField.TextFieldMaster.Name & ", " & _
            oField.TextFieldMaster.Value & ", " & _
            oField.TextFieldMaster.InstanceName & ")" & CHR$(10)
    ElseIf oField.supportsService("com.sun.star.text.TextField.PageNumber") Then
        s = s & "Found page number" & CHR$(10)
    ElseIf oField.supportsService("com.sun.star.text.TextField.GetReference")
Then
        n = n + 1
        s = s & "Reference (" & oField.SourceName & ", " &
oField.getPresentation(False) & ")" & CHR$(10)
    ElseIf oField.supportsService("com.sun.star.text.TextField.SetExpression")
Then
        REM This includes things such as Tables, Listing, and Figures
        REM The values will be similar to "SetExpression (Table, Table + 1)"
        If bDisplayExpressions Then
            n = n + 1
            s = s & "SetExpression (" & oField.VariableName & ", " & oField.Content
& ")" & CHR$(10)
        Else
            s = s & "What is this?"
        End If
    End If

```



```

        ElseIf oField.supportsService("com.sun.star.text.TextField.Chapter") OR _
            oField.supportsService("com.sun.star.text.TextField.DocInfo.Title")
OR _
oField.supportsService("com.sun.star.text.TextField.DocInfo.ChangeDateTime") OR
-
            oField.supportsService("com.sun.star.text.TextField.Bibliography") OR
-
            oField.supportsService("com.sun.star.text.TextField.Annotation") Then
    n = n + 1
    s = s & oField.getPresentation(True) & " : " &
oField.getPresentation(False) & CHR$(10)
    Else
        nUnknown = nUnknown + 1
        n = n + 1
        If nUnknown = 1 Then inspect(oField)
        s = s & "Unknown : " & oField.getPresentation(True) & " : " &
oField.getPresentation(False) & CHR$(10)
    End If
    If n > 50 Then
        MsgBox s, 0, "Text Fields"
        s = ""
        n = 0
    End If
Loop
If n > 0 Then
    MsgBox s, 0, "Text Fields"
    s = ""
    n = 0
End If
If nUnknown > 0 Then Print "Found " & nUnknown & " unexpected field types"
End Sub

```

Based on the above, it is easy to remove references from a document. Enumerate fields and set references to the reference text.

7.29.2. Enumerate text content

All references and reference definitions are available by enumerating the text content. To use the following macro, select a portion of your document and then run the macro. The output is very lengthy.

```

Sub EnumerateSelTextSectionsFindFields(Optional oUseDoc)
    Dim oDoc          'Document to use.
    Dim oEnum          'Enumeration of text sections.
    Dim oSect
    Dim oParEnum
    Dim oPar
    Dim oField        'Enumerated text field.

```

```

Dim s$           'Generic string variable.
Dim n%           'Count the number of text fields.
Dim oCursors()
Dim sPrompt$
Dim i%
Dim bMarkStart As Boolean
Dim bRefStart As Boolean
Dim sRefName
Dim oSel
Dim oSels

If IsMissing(oUseDoc) Then
    oDoc = ThisComponent
Else
    oDoc = oUseDoc
End If

oSels = oDoc.getCurrentSelection()
If IsNull(oSels) Then Exit Sub
If oSels.getCount() < 1 Then Exit Sub

For i = 0 To oSels.getCount() - 1
    oSel = oSels.getByIndex(i)
    If HasUNOInterfaces(oSel, "com.sun.star.container.XEnumerationAccess") Then
        oParEnum = oSel.createEnumeration()
        bMarkStart = False
        bRefStart = False
        Do While oParEnum.hasMoreElements()
            oPar = oParEnum.nextElement()
            If oPar.supportsService("com.sun.star.text.Paragraph") Then
                oEnum = oPar.createEnumeration()
                Do While oEnum.hasMoreElements()
                    oSect = oEnum.nextElement()
                    '??Inspect oSect
                    If NOT IsNULL(oSect.ReferenceMark) Then
                        n = n + 1
                        bRefStart = NOT bRefStart
                        if bRefStart Then
                            s = s & "<RefMark name='" & oSect.ReferenceMark.getName() & "'>"
                        Else
                            s = s & "</RefMark>" & CHR$(10)
                        End If
                    ElseIf NOT IsNULL(oSect.bookmark) Then
                        n = n + 1
                        bMarkStart = NOT bMarkStart
                        if bMarkStart Then
                            s = s & "<Bookmark name='" & oSect.bookmark.LinkDisplayName &
"
```

```

Else
    s = s & "</Bookmark>" & CHR$(10)
End If
ElseIf oSect.supportsService("com.sun.star.text.TextField") Then
    If
oSect.TextField.supportsService("com.sun.star.text.TextField.SetExpression")
Then
        s = s & "<SetExpression name='" & oSect.TextField.VariableName &
        "'>" & oSect.getString() & "</SetExpression>" & CHR$(10)
        'Inspect(oSect.TextField)
    Else
        s = s & "<TextField"
        If NOT IsNull(oSect.TextField) Then
            If
oSect.TextField.supportsService("com.sun.star.text.TextField.GetReference") Then
                s = s & " refTo=" & oSect.TextField.SourceName
            End If
        End If
        s = s & ">" & oSect.getString() & "</TextField>" & CHR$(10)
        'inspect oSect.TextField
    End If
Else
    If NOT bRefStart AND NOT bMarkStart Then
        s = s & "<TextContent>" & oSect.getString() & "</TextContent>" &
CHR$(10)

    Else
        s = s & oSect.getString()
    End If
    n = n + 1
End If
If n > 40 Then
    MsgBox s
    s = "" : n = 0
End If
Loop
ElseIf opar.supportsService("com.sun.star.text.TextTable") Then
    REM How to handle text tables?
    'Inspect(oPar)
    s = "<Table>...</Table>" & CHR$(10)
Else
    REM What did we find here?
    Inspect(oPar)
End If
Loop
Else
    REM Something is selected that can not be enumerated.
End If
MsgBox s, 0, "Reference Marks"

```

Next
End Sub

This is an example of the output:

```
1. <Bookmark name='__RefHeading__29453202'>Heading 1 here</Bookmark>
2. <TextContent>Table</TextContent>
3. <SetExpression name='Table'>1</SetExpression>
4. <TextContent>Manual reference to </TextContent>
5. <RefMark name='fooref'>foo</RefMark>
6. <TextContent> named fooref.</TextContent>
7. <TextContent>I am a reference to</TextContent>
8. <TextField refTo=fooref>foo<TextField>
9. <TextContent> on page </TextContent>
10.<TextField refTo=fooref>1<TextField>
11.<TextContent>See </TextContent>
12.<TextField refTo=Table>Table 1<TextField>
13.<TextContent>See </TextContent>
14.<TextField refTo=__RefHeading__29453202>Heading 1 here<TextField>
15.<TextContent>See </TextContent>
16.<TextField refTo=here ref>here<TextField>
17.<TextContent>Another Heading</TextContent>
```

Line 1 contains a text heading that is referenced at line 14. A text field is used to reference the heading and the heading has been turned into a bookmark. The bookmarks object does not include this bookmark in the list of names. Line 17 is also a heading, which demonstrates that a bookmark is not created for the heading until it is referenced.

Lines 2 and 3 represent the start of a table caption. A table caption is a number range with the equation “Table + 1”, which is why it is represented as a set expression. The caption is referenced on line 12

I set a manual reference on line 5 named “fooref”. Foo is referenced on line 10

Bookmarks and reference marks use a begin / end type construct. When first encountered, the item starts, and when next encountered, it ends. The macro uses a flag to track this.

Although a reference to a heading is internally implemented by creating a bookmark, the bookmark does not exist with the bookmark object. A style may have two names, and only one is visible. In this case, however, I see no way find the created bookmarks, even when you know the name. Cross referenced headings are named __RefHeading__<number>. You also have the ability to reference numbered paragraphs, which use the same methodology, and are named __RefNumPara__<number>.

7.29.3. Removing references

I do know how to do this, in fact, you should consider it easy..

7.29.4. Adding references

I have no idea how to reference a heading, for example...

8. Text tables

Text tables are difficult to understand and I see many questions regarding text tables. As such, I have started a new section specifically on text tables. As time permits, I will continue to expand this section and to move all existing text table examples into this section.

8.1. Finding text tables

The easiest way to find text tables is to obtain the TextTables object and then obtain them directly. The TextTable object returns all of the text tables in the document.

Listing 8.1: Get the text tables from the TextTables object.

```
Sub GetTextTablesDirectly
    Dim s As String
    Dim i As Long
    Dim oTables
    Dim oTable

    oTables = ThisComponent.getTextTables()
    Print "This document contains " & oTables.getCount() & " tables"

    REM I could also check getCount() = 0.
    If NOT oTables.hasElements() Then Exit Sub

    For i = 0 To oTables.getCount() - 1
        oTable = oTables.getByIndex(i)
        s = s & oTable.getName() & CHR$(10)
    Next
    MsgBox s, 0, "Using Enumeration"

    REM Even faster
    s = Join(oTables.getElementNames(), CHR$(10))
    MsgBox s, 0, "Using getElementNames()"

    REM Get the name of the first table.
    s = oTables.getByIndex(0).getName()

    REM You can test to see if a table with a specific name exists
    REM and then get the table based on its name.
    If oTables.hasByName(s) Then
        oTable = oTables.getByName(s)
    End If
End Sub
```

8.1.1. Where is the text table

Use the `getAnchor()` method of a text table to determine where a text table is located. An anchor is a text range and a text cursor is a text range. A text cursor is able to goto a text range. Unfortunately, the anchor returned by a text table acts differently than many other anchors so a cursor is not able to move to a text table anchor.

Every visible document has a controller and the controller can select things. Selecting a table causes the view cursor to move into the first cell in the text table. You can then use the view cursor to move left one character, which places the text cursor directly before the text table.

Listing 8.2: Move the view cursor before a text table.

```
REM Move the cursor to the first row and column
ThisComponent.getCurrentController().select(oTable)
oCurs = ThisComponent.getCurrentController().getViewCursor()
oCurs.goLeft(1, False)
```

The final location of the view cursor depends on what is before the text table. If there is a paragraph immediately before the text table, then the view cursor is immediately before the text table. Unfortunately, this is not always true. For example, when a text table is the first thing in a document, or the first thing in a table cell.

A standard trick to determine if the view cursor is in a specific type of object is to inspect the view cursor properties. As already stated, the anchor returned by a text table does not work as most text range objects, so although this works for the view cursor, it fails for a text table anchor.

Listing 8.3: Is the view cursor in a text table?

```
Dim oVC
oVC = ThisComponent.getCurrentController().getViewCursor()
If NOT IsEmpty(oVC.Cell) Then Print "In a Cell"
If NOT IsEmpty(oVC.TextField) Then Print "In a Field"
If NOT IsEmpty(oVC.TextFrame) Then Print "In a Frame"
If NOT IsEmpty(oVC.TextSection) Then Print "In a Section"
If NOT IsEmpty(oVC.TextTable) Then Print "In a Table"
```

A table anchor is able to return the text object in which it is anchored. Create a text cursor from the text object and then inspect the cursor to determine if the text object containing the text table anchor is contained in another object.

Listing 8.4: Is the view cursor in a text table contained in another text table?

```
Dim oTable
Dim oVC
Dim oText
Dim oCurs

oVC = ThisComponent.getCurrentController().getViewCursor()
If IsEmpty(oVC.TextTable) Then
    Print "The view cursor is not in a text table"
    Exit Sub
End If

oTable = oVC.TextTable
oText = oTable.getAnchor().getText()
oCurs = oText.createTextCursor()
If NOT IsEmpty(oCurs.Cell) Then Print "Table is in a Cell"
If NOT IsEmpty(oCurs.TextField) Then Print "Table is in a Field"
If NOT IsEmpty(oCurs.TextFrame) Then Print "Table is in a Frame"
If NOT IsEmpty(oCurs.TextSection) Then Print "Table is in a Section"
If NOT IsEmpty(oCurs.TextTable) Then Print "Table is in a Table"
```

The examples are all extreme cases that illustrate the issues that must be addressed. You know your documents and how complicated they will be, so usually, this level of detail is not required.

8.1.2. Enumerating text tables.

A text document contains one primary text object that contains most of the text content in a document. Examples of text content include paragraphs, text tables, text fields, and graphics. Every text object provides a method to enumerate the text content.

Listing 8.5: Enumerate the paragraphs in a text object.

```
Sub EnumerateParagraphs
    Dim oParEnum
    Dim oPar
    Dim i As Long

    oParEnum = ThisComponent.getText().createEnumeration()
    Do While oParEnum.hasMoreElements()
        i = i + 1
        oPar = oParEnum.nextElement()
    Loop
    Print "There are " & i & " paragraphs"
End Sub
```

Paragraphs and text tables are enumerated at the highest level. In other words, text tables act very much like paragraphs. It makes sense, therefore, that a single paragraph or line can not contain more than one text table.

Listing 8.6: Enumerate the paragraphs and text tables in a text object.

```
Sub EnumerateTextContent
    Dim oParEnum
    Dim oPar
    Dim nPar As Long
    Dim nTable As Long

    oParEnum = ThisComponent.getText().createEnumeration()
    Do While oParEnum.hasMoreElements()
        oPar = oParEnum.nextElement()
        If oPar.supportsService("com.sun.star.text.Paragraph") then
            nPar = nPar + 1
        ElseIf oPar.supportsService("com.sun.star.text.TextTable") Then
            nTable = nTable + 1
        End if
    Loop
    MsgBox "There are " & nPar & " paragraphs" & CHR$(10) & _
        "There are " & nTable & " tables"
End Sub
```

There are many ways to display more than one text table on the same line. A text table can be inserted into a frame, which can be anchored as a character. A text table can be inserted into a text section with multiple columns, and a text table can be inserted into a cell in another text table. The macro in *Listing 8.6* will not find a text table that is in a non-standard location such as embedded in a header or footer, inside another text table, in a frame, or in a section.

8.2. Enumerating cells in a text table.

The regular text in a document is contained in the document's text object, which is available using the method `getText()`. Each cell in a text table also contains a text object that can be enumerated. The method used to enumerate the cells depends on the text table. A simple table contains no merged or split cells (see *Table 8.1*).

Table 8.1. Simple table with the cell names labeled.

A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4

Use `getRows()` to obtain the rows from the text table. The returned rows object is useful to determine how many rows are present, retrieving, inserting, and deleting rows (see *Table 8.2*).

Table 8.2. Main access methods supported by a table rows object.

Method	Description
<code>getByIndex</code>	Retrieve a specific row.
<code>getCount</code>	Number of rows in the table.
<code>hasElements</code>	Determine if there are any rows in the table.
<code>insertByIndex</code>	Add rows to the table.
<code>removeByIndex</code>	Remove rows from the table.

Although it is easy to enumerate individual rows, a row is not useful for obtaining the cells that it contains. An individual row object is primarily used as follows:

- Set the row height using the Height attribute.
- Set `IsAutoHeight` to true so that the rows height is automatically adjusted.
- Set `IsSplitAllowed` to false so that a row can not be split at a page boundary.
- Modify the `TableColumnSeparators` to change column widths.

Use `getColumns()` to obtain the columns from the text table. The columns object supports the same methods as the rows object (see *Table 8.2*). It is not possible, however, to obtain a specific column from a column object; the method `getByIndex()` exists, but it returns null. Although I expected to set a column width by obtaining a specific column and setting the width, you need to modify the table column separators available from the row object.

8.2.1. Simple text tables

It is easy to enumerate the cells in a simple text table. The macro in *Listing 8.7* enumerates the cells in the text table containing the view cursor. Each cell is obtained using the method `getCellByPosition()`, which is only available for a simple text table. If the text table contains merged or split cells, another method must be used.

Listing 8.7: Enumerate the cells in a simple text table.

```

Dim s As String
Dim oTable
Dim oVC
Dim oCell
Dim nCol As Long
Dim nRow As Long

oVC = ThisComponent.getCurrentController().getViewCursor()
If IsEmpty(oVC.TextTable) Then
    Print "The view cursor is not in a text table"
    Exit Sub
End If

oTable = oVC.TextTable
For nRow = 0 To oTable.getRows().getCount() - 1
    For nCol = 0 To oTable.getColumns().getCount() - 1
        oCell = oTable.getCellByPosition(nCol, nRow)
        s = s & oCell.CellName & ":" & oCell.getString() & CHR$(10)
    Next
Next
MsgBox s

```

8.2.2. Formatting a simple text table

I use a similar method to format text tables as specified by the OOoAuthors web site (see *Table 8.2*). I place the text cursor into a text table and then I run the macro in *Listing 8.8*. First, the macro disables vertical lines between rows; and enables all the rest. Next, all of the cells are enumerated using the method demonstrated in *Listing 8.7*. Each cell background color is set based on its position in the table. Finally, the text object is obtained from each table cell and the paragraphs are enumerated and the paragraph style is set.

Listing 8.8: Format a text table as specified by OOoAuthors.

```

Sub Main
    FormatTable()
End Sub

Sub FormatTable(Optional oUseTable)
    Dim oTable
    Dim oCell
    Dim nRow As Long
    Dim nCol As Long

    If IsMissing(oUseTable) Then
        oTable = ThisComponent.CurrentController.getViewCursor().TextTable
    Else
        oTable = oUseTable
    End If
    If IsNull(oTable) OR IsEmpty(oTable) Then
        Print "FormatTable: No table specified"
        Exit Sub
    End If

```

```

End If

Dim v
Dim x
v = oTable.TableBorder
x = v.TopLine      : x.OuterLineWidth = 2 : v.TopLine      = x
x = v.LeftLine     : x.OuterLineWidth = 2 : v.LeftLine     = x
x = v.RightLine    : x.OuterLineWidth = 2 : v.RightLine    = x
x = v.TopLine      : x.OuterLineWidth = 2 : v.TopLine      = x
x = v.VerticalLine  : x.OuterLineWidth = 2 : v.VerticalLine  = x
x = v.HorizontalLine : x.OuterLineWidth = 0 : v.HorizontalLine = x
x = v.BottomLine   : x.OuterLineWidth = 2 : v.BottomLine   = x
oTable.TableBorder = v

For nRow = 0 To oTable.getRows().getCount() - 1
  For nCol = 0 To oTable.getColumns().getCount() - 1
    oCell = oTable.getCellByPosition(nCol, nRow)
    If nRow = 0 Then
      oCell.BackColor = 128
      SetParStyle(oCell.getText(), "OOoTableHeader")
    Else
      SetParStyle(oCell.getText(), "OOoTableText")
      If nRow MOD 2 = 1 Then
        oCell.BackColor = -1
      Else
        REM color is (230, 230, 230)
        oCell.BackColor = 15132390
      End If
    End If
  Next
Next
End Sub

Sub SetParStyle(oText, sParStyle As String)
  Dim oEnum
  Dim oPar
  oEnum = oText.createEnumeration()
  Do While oEnum.hasMoreElements()
    oPar = oEnum.nextElement()
    If oPar.supportsService("com.sun.star.text.Paragraph") Then
      'oPar.ParaConditionalStyleName = sParStyle
      oPar.ParaStyleName = sParStyle
    End If
  Loop
End Sub

```

The macro in *Listing 8.8* assumes that the table is simple. If a cell contains text tables, frames, or sections, they are simply ignored. While writing your own macros, you must decide how flexible the macro will be. I made assumptions based on my usage. I knew that I would only use this macro on simple text tables so the macro itself is relatively simple and short.

If you have a text table auto-format style, it is even easier to format the table. Text table objects support the method `autoFormat(name)`, which accepts the format name as an argument. One might argue that the macro in

8.2.3. What is a complex text table

A complex text table is a text table that is not simple. More accurately, a complex text table contains cells that have been split, or merged. To demonstrate a complex text table, start with *Table 8.1* and perform the following tasks to obtain *Table 8.6*.

- 1) Right click in cell A2 and choose Cell > Split > Horizontal. Cell A2 becomes two cells. From a cell naming perspective, a new third row is inserted that contains one column. At this point, the API indicates that there are five rows and four columns (see *Table 8.3*).

Table 8.3. Split cell A2 horizontally.

A1	B1	C1	D1
A2	B2	C2	D2
(A3)			
A3=>A4	B3=>B4	C3=>C4	D3=>D4
A4=>A5	B4=>B5	C4=>C5	D4=>D5

- 2) Right click in cell B2 and choose Cell > Split > Vertical. Cell B2 becomes two distinct cells, B2 and C2; a new column appears to have been inserted at cell B2. The API still indicates that there are five rows and four columns (see *Table 8.4*).

Table 8.4. Split cell B2 vertically..

A1	B1	C1	D1
A2	B2	(C2)	C2=>D2
A3			D2=>E2
A4	B4	C4	D4
A5	B5	C5	D5

- 3) Select cells A4 and B4, right click and choose Cell > Merge.

Table 8.5. Complex table after merging cells in the same row.

A1	B1	C1	D1
A2	B2	C2	D2
A3			E2
A4, (B4) => A4		C4=>B4	D4=>C4
A5	B5	C5	D5

- 4) Select cells B4 and C5, right click and choose Cell > Merge. Cell C5 just goes away.

Table 8.6. Split cell B2 vertically..

A1	B1		C1	D1
A2	B2	C2	D2	E2
A3				
A4			B4, (C5)=>B4	C4
A5	B5			D5

8.2.4. Enumerating cells in any text table

You can use `getCellByPosition(col, row)` or `getCellByName("A2")` to obtain a cell. Both methods work on simple and complex text tables. Use `getCellNames()` to return an array of cell names contained in the table.

Table 8.6 is a complex table identifying cell names. The table is duplicated in Table 8.7 showing both column, row, and name to return an individual cell.

Table 8.7. Split cell B2 vertically.

(0, 0) = A1	(1, 0) = B1		(2, 0) = C1	(3, 0) = D1
(0, 1) = A2	(1, 1) = B2	(2, 1) = C2	(3, 1) = D2	(4, 1) = E2
(0, 2) = A3				
(0, 3) = A4			(1, 3) = B4	(2, 3) = C4
(0, 4) = A5	(1, 4) = B5			(4, 0) = D5

I created a table with 5 columns and 5 rows (see Table 8.8). Each cell indicates a column, row, and name. A cell with a red background (and the text Error) indicates a cell that you can not get based on the column and row. Each grayed cell (and the text Hidden) indicates a cell that you can get based on the column and row, but that is not visible, and the name is not returned by `getCellNames()`.

Table 8.8. Identify cells available by name and position.

(0, 0) = A1	(1, 0) = B1	(2, 0) = C1	(3, 0) = D1	(4, 0) = E1 Error
(0, 1) = A2	(1, 1) = B2	(2, 1) = C2	(3, 1) = D2	(4, 1) = E2
(0, 2) = A3	(1, 2) = B3 Hide	(2, 2) = C3 Hide	(3, 2) = D3 Hide	(4, 2) = E3 Hide
(0, 3) = A4	(1, 3) = B4	(2, 3) = C4 Hide	(3, 3) = D4 Error	(4, 3) = E4 Error
(0, 4) = A5	(1, 4) = B5	(2, 4) = C5 Hide	(3, 4) = D5	(4, 4) = E5 Error

Listing 8.9: Enumerate the cells in any text table.

```
Dim sNames()  
Dim i As Long
```

```

Dim s As String
Dim oTable
Dim oVC
Dim oCell

oVC = ThisComponent.getCurrentController().getViewCursor()
If IsEmpty(oVC.TextTable) Then
    Print "The view cursor is not in a text table"
    Exit Sub
End If

oTable = oVC.TextTable
sNames() = oTable.getCellNames()
For i = LBound(sNames()) To UBound(sNames())
    oCell = oTable.getCellByName(sNames(i))
    s = s & sNames(i) & " = " & oCell.getString() & CHR$(10)
Next
MsgBox s

```

Although the order that the cells are returned is not clearly defined, this method works well for all text table types. You can easily display a list of cell names as follows:

```
MsgBox Join(oTable.getCellNames(), CHR$(10))
```

8.3. Getting data from a simple text table

If you need to obtain the data in a table, use the methods `getDataArray()` and `getData()`, which return all of the data in the table in an array. Getting the data is faster than enumerating the cells and then obtaining the data. Each get method has a corresponding set method, which can set all of the data at one time.

The data is returned as an array of arrays, not as a two dimensional array. The following code displays all of the numeric data in the first row of a table.

```

Dim oData() : oData() = oTable.getDataArray()
MsgBox Join(oData(0), CHR$(10))

```

Assuming that the text table contains two rows and three columns, the following code will set the data in the table.

```
Otable.setData(Array(Array(0, 1, 2), Array(3, 4, 5)))
```

Use `getData()` and `setData()` to set numerical data. All entries are assumed to be double precision numbers. Text data is converted to be zero.

Tip	A cell is considered to contain numeric data if, and only if, the cell is formatted as a number.

Use `getDataArray()` and `setDataArray()` to set data that contains strings. In version 2.01, string data is not returned or set, but an issue has been filed.

8.4. Table cursors and cell ranges

Text tables support the method `createCursorByCellName()`. Table cursors can be moved and positioned similarly to their text cursor counter-parts. You can also use a table cursor to split and merge cells. The view cursor acts like a table cursor when it is placed inside of a text table, and can therefore be used to copy a text table or a range of cells to a new text table (see my book *OpenOffice.org Macros Explained* pages 309 – 311).

8.5. Cell ranges

Most of the functionality in a text table is also supported by a cell range. In other words, if you can do it with a table object, you might be able to do it with a cell range object. For example, you can use the get and set data functions on a cell range. You can also sort a cell range, and obtain individual cells from a cell range.

Use the methods `getCellRangeByPosition()` and `getCellRangeByName()` to obtain a cell range from a text table. A table cursor can also select a range of cells, but a table cursor is not a cell range. A table cursor supports the method `getRangeName()`, which can then be used to obtain the range.

8.5.1. Using a cell range to clear cells

In a Calc document, the cell range object supports a method to clear a cell; a cell range from a text table does not. You can fake it as follows:

Listing 8.10: Clearing data from a text table.

```
Sub ClearCells()
    Dim oTable
    Dim oRange
    Dim oData()
    Dim oRow()
    Dim i%, j%

    REM Get the FIRST text table
    oTable = ThisComponent.getTextTables().getByIndex(0)
    oRange = oTable.getCellRangeByName("B1:D4")
    oData() = oRange.getDataArray()
    For i = LBound(oData()) To UBound(oData())
        oRow() = oData(i)
        For j = LBound(oRow()) To UBound(oRow())
            oRow(j) = ""
        Next
    Next
    oRange.setDataArray(oData())
End Sub
```

I made no attempt to make the code efficient.

8.6. Chart data

Don't worry about this stuff... I should probably not bother mentioning this unless I intend to do something with charts.??

ARRAY getColumnDescriptions (void)

ARRAY getRowDescriptions (void)

VOID setColumnDescriptions (ARRAY)

VOID setRowDescriptions (ARRAY)

8.7. Column Widths

The column separator specifies where the column ends as percentage of the table width. A column end position of 5000 specifies 50% of the table width. The macro in Listing 8.11 sets the first column to end at 50% of the current table width and the second column at 70% of the total table width.

A table provides this property only if all rows have the same structure. In other words, you can not set the column width for an entire table on a complex table. Also, if a particular separator has the `IsVisible` flag set to false, then it is not visible. Hidden separators cannot be moved and they cannot be overtaken by visible separators.

Listing 8.11: Set column width for the first two columns.

```
Sub SetTwoColsWidths
    Dim oTblColSeps 'The array of table column separators.
    Dim oTable      'The first text table in the document.

    'Print
    oTable = ThisComponent.getTextTables().getByIndex(0)
    oTblColSeps = oTable.TableColumnSeparators

    Rem Change the positions of the two separators.
    oTblColSeps(0).Position = 5000
    oTblColSeps(1).Position = 7000

    REM You must assign the array back
    oTable.TableColumnSeparators = oTblColSeps
End Sub
```

8.8. Setting the optimal column width

In a Calc document, you set the `OptimalWidth` column property to `True`. There is no simple solution for a text table using the API.

The GUI provides a method that can set the width of a column based on the location of the text cursor, or the portion of the text table that is selected. This method is available by using a dispatch. My book explains how to select areas in a text table, so I will not repeat that discussion here. The macro in Listing 8.12 selects an entire text table and then sets the column width for the entire text table.

Listing 8.12: Set an entire table for optimal column width.

```
Sub SetTableOptimumWidth
    Dim oDispHelper 'Dispatch helper
    Dim oFrame      'Current window frame.
    Dim oTable      'First table in the document.
```



```

Dim oVCursor      'The view cursor
Dim s$

oTable = ThisComponent.getTextTables().getByIndex(0)
ThisComponent.getCurrentController().select(oTable)
oVCursor = ThisComponent.getCurrentController().getViewCursor()
oVCursor.gotoEnd(True)
oVCursor.gotoEnd(True)
oFrame = ThisComponent.CurrentController.Frame
oDispHelper = createUnoService("com.sun.star.frame.DispatchHelper")
s$ = ".uno:SetOptimalColumnWidth"
oDispHelper.executeDispatch(oFrame, s, "", 0, Array())
End Sub

```

8.9. How wide is a text table?

On the surface, it seems very easy to determine the width of a text table; access the Width property. This is, unfortunately, not sufficient. To determine the actual width of a text table, you must inspect the properties shown in Table 8.9.

Table 8.9: Text table properties related to the text table width.

Property	Description
LeftMargin	Left margin of the table.
RightMargin	Right margin of the table.
HoriOrient	Contains the horizontal orientation from the com.sun.star.text.HoriOrientation constant group.
RelativeWidth	Determines the width of the table relative to its environment.
IsWidthRelative	Determines if the value of the relative width is valid.
Width	Sometimes this contains the absolute table width.

In my book (page 307), I mention the HoriOrient as controlling the meaning of the other properties. If the HoriOrient property contains the default value FULL, then the properties are essentially useless, including the Width property. In this case, you must determine the width of the column that contains the text table. I could elaborate, but I have been answering macro questions for hours and I have other work to do.

8.10. The cursor in a text table

You can check to see if the cursor is inside of a text table using the code in Listing 7.2, this code also demonstrates how to determine if more than a single cell is selected. The macro in Listing 8.13 prints information about text cursor in a text table.

Listing 8.13: Inspect the text table that contains the cursor.

```

Sub InspectCurrentTable
Dim oTable      'The table that contains the cursor.
Dim oCell       'The cell that contains the cursor.
Dim oVCurs      'The current view cursor.
Dim s$          'Contains explanatory text.

```

```

Dim nCol%      'The column that contains the cursor.
Dim nRow%      'The row that contains the cursor.

oVCurs = ThisComponent.getCurrentController().getViewCursor()
If IsEmpty(oVCurs.TextTable) Then
    s = s & "The cursor is NOT in a text table"
Else
    oTable = oVCurs.TextTable
    oCell  = oVCurs.Cell

    REM Assume less than 26 columns
    nCol = Asc(oCell.Cellname) - 65
    nRow = CInt(Right(oCell.Cellname, Len(oCell.Cellname) - 1)) - 1
    s = s & "The cursor is in text table " & _
        oTable.getName() & CHR$(10) & _
        "The current cell is " & oCell.Cellname & CHR$(10) & _
        "The cell is at (" & nCol & ", " & nRow & ")" & _
        CHR$(10) "The table has " & _
        oTable.getColumns().getCount() & " columns and " & _
        oTable.getRows().getCount() & " Rows" & CHR$(10)
    REM *** Insert more code here!
End If
MsgBox s
End Sub

```

To select an entire row, first move the cursor so the first cell in the row:

Listing 8.14: *Move the cursor to the first cell in a table.*

```

oCell1 = oTable.getCellByPosition(0, nRow)
ThisComponent.getCurrentController().select(oCell1)

```

Next, select the entire cell. If the cell contains text, `oVCurs.gotoEnd(True)` moves the cursor to the end of the text in the current cell. If the cursor is already at the end of the cell, such as when the cell does not contain any text, then the cursor will move to the end of the table. The `gotoEndOfLine()` method always moves the cursor to the end of the current line, which is fine if the cell contains only one line of text. After selecting the entire cell, you can move the view cursor to the right to select the rest of the cells. The code in Listing 8.15 is written to assume that it is inserted into the code in Listing 8.13.

Listing 8.15: *Select the entire row of a simple text table.*

```

Dim oCell1
Dim oText
oCell1 = oTable.getCellByPosition(0, nRow)
ThisComponent.getCurrentController().select(oCell1)
oText = oCell1.getText()
Dim oStart : oStart = oText.getStart()
If oText.compareRegionStarts(oStart(), oText.getEnd()) <> 0 Then
    oVCurs.gotoEnd(True)
End If
oVCurs.goRight(oTable.getColumns().getCount()-1, True)

```

8.10.1. Move the cursor after a text table

First, assume that the view cursor is in a text table. Using a trick that I mention in my book, I can easily move the view cursor after the current text table.

Listing 8.16: *Move the view cursor after the current text table.*

```
Sub CursorAfterCurrentTable()
    Dim oCursor

    REM Get the view cursor
    oCursor = ThisComponent.getCurrentController().getViewCursor()

    REM Verify that the cursor is in a text table.
    If IsNull(oCursor.TextTable) OR IsEmpty(oCursor.TextTable) Then
        Print "No text table is selected"
        Exit Sub
    End If

    REM Now, move to the last cell in the table (as explained in my book).
    REM This works well UNLESS the current cell is empty.
    'oCursor.gotoEnd(False)
    'oCursor.gotoEnd(False)
    'oCursor.goDown(1, False)
    REM This solution was suggested by JohnV on the OooForum.
    REM This may have problems if two tables are next to each
    REM other or if one table is contained in another. You can
    REM work around this easily if required.
    While Not IsEmpty(oCursor.TextTable)
        oCursor.goDown(1, False)
    Wend
End Sub
```

Moving the cursor after a specific table is just as easy. The following example assumes that at least one table exists:

Listing 8.17: *Move the view cursor after a specified table.*

```
Sub CursorAfterFirstTable()
    Dim oTable

    REM Get the FIRST text table
    oTable = ThisComponent.getTextTables().getByIndex(0)

    REM Move the cursor to the first cell in the table
    ThisComponent.getCurrentController().Select(oTable)

    REM Move AFTER the current table
    CursorAfterCurrentTable()
End Sub
```

8.11. Creating a table

The following macro demonstrates handling text tables and the contained cells.

Listing 8.18: *Create a text table and insert cells.*

```
'Author: Hermann Kienlein
'Author: Christian Junker
Sub easyUse( )
    Dim odoc, otext, ocursor, mytable, tablecursor

    odoc = thisComponent
    otext = odoc.getText()
    mytable = CreateTable(odoc)

    'create normal TextCursor
    ocursor = otext.CreateTextCursor()
    ocursor.gotoStart(false)

    'now that we defined the range = position of the table, let's insert it
    otext.insertTextContent(ocursor, myTable, false )
    tablecursor = myTable.createCursorByCellName("A1")

    InsertNextItem("first cell", tablecursor, mytable) 'insert a new item:
    InsertNextItem("second cell", tablecursor, mytable) 'and another one:
End Sub

Sub InsertNextItem(what, oCursor, oTable)
    Dim oCell As Object
    'name of the cell range that is selected by this cursor
    sName = oCursor.getRangeName()
    ' The cell name will be something like D3
    oCelle = oTable.getCellByName(sName)
    oCelle.String = what
    oCursor.goRight(1,FALSE)
End Sub

Function CreateTable(document) As Object
    oTextTable = document.createInstance("com.sun.star.text.TextTable")
    oTextTable.initialize(5, 8)
    oTextTable.HoriOrient = 0 'com.sun.star.text.HoriOrientation::NONE
    oTextTable.LeftMargin = 2000
    oTextTable.RightMargin = 1500
    CreateTable = oTextTable
End Function

Sub deleteTables()
    'sometimes deleting tables in the GUI seems kind of silly,
    'this procedure will delete all tables
    Dim enum, textobject

    enum = thisComponent.Text.createEnumeration
    While enum.hasMoreElements()
```

```

txtcontent = enum.nextElement()
If txtcontent.supportsService("com.sun.star.text.TextTable") Then
    thisComponent.Text.removeTextContent(txtcontent)
End If
Wend
End Sub

```

8.12. A table with no borders

Lalaimia Samia <samia.lalaimia@infotel.com> asked me how to insert a table into a document that contained no borders. You can not manipulate borders of a table until after it has been inserted. The next problem is that each of the properties is a structure so it is not possible to directly modify the structure because you are only modifying a copy. The answer is to copy the structure to a temporary variable, modify the temporary variable, and then copy the temporary variable back. This is tedious but simple.

Listing 8.19: *Insert a text table with no borders.*

```

Sub InsertATableWithNoBorders
    Dim oTable    'Newly created table to insert
    Dim oEnd

    REM Let the document create the text table.
    oTable = ThisComponent.createInstance( "com.sun.star.text.TextTable" )
    oTable.initialize(4, 1) 'Four rows, one column

    REM Now insert the text table at the end of the document.
    Oend = ThisComponent.Text.getEnd()
    ThisComponent.Text.insertTextContent(oEnd, oTable, False)

    Dim x 'represents each BorderLine
    Dim v 'represents the TableBorder Object as a whole
    v = oTable.TableBorder
    x = v.TopLine           : x.OuterLineWidth = 0 : v.TopLine = x
    x = v.LeftLine          : x.OuterLineWidth = 0 : v.LeftLine = x
    x = v.RightLine         : x.OuterLineWidth = 0 : v.RightLine = x
    x = v.TopLine           : x.OuterLineWidth = 0 : v.TopLine = x
    x = v.VerticalLine       : x.OuterLineWidth = 0 : v.VerticalLine = x
    x = v.HorizontalLine    : x.OuterLineWidth = 0 : v.HorizontalLine = x
    x = v.BottomLine        : x.OuterLineWidth = 0 : v.BottomLine = x

    oTable.TableBorder = v
    Dim a()
    a() = Array(Array("Files"), Array("One"), Array("Two"), Array("Three"))
    oTable.setDataArray(a())
End Sub

```

9. MacroFormatterADP - Colorize code and XML

This section contains the macros I use to format macros in text documents so that they look like macros viewed in the IDE. I packaged this as an extension using the extension compiler from Bernard Marcelly. If you desire to see the complete package that I use to create the extension, just ask.

9.1. Strings Module

I categorize each character as white space, a special character, or a word separator. I use an array, indexed by ASCII value from 0 to 256, to indicate that a character is in a specific category.

Listing 9.1: Module header for formatting macro string utilities.

```
Option Explicit

Private bCheckWhite(0 To 256) As Boolean
Private bCheckSpecial(0 To 256) As Boolean
private bWordSep(0 To 256) As Boolean
```

Every value is set to False, and then the values for the special characters are explicitly set to True.

Listing 9.2: Initialize special character arrays.

```
'*****
'** Initialize the variables that contain the special characters.
'*****

Sub FMT_InitSpecialCharArrays()
    Dim i As Long

    For i = LBound(bCheckWhite()) To UBound(bCheckWhite())
        bCheckWhite(i) = False
        bCheckSpecial(i) = False
        bWordSep(i) = False
    Next

    bCheckWhite(9) = True
    bCheckWhite(10) = True
    bCheckWhite(13) = True
    bCheckWhite(32) = True
    bCheckWhite(160) = True

    bCheckSpecial(Asc("+")) = True
    bCheckSpecial(Asc("-")) = True
    bCheckSpecial(Asc("&")) = True
    bCheckSpecial(Asc("*")) = True
    bCheckSpecial(Asc("/")) = True
    bCheckSpecial(Asc(":")) = True
    bCheckSpecial(Asc(";")) = True
    bCheckSpecial(Asc("=")) = True
```

```

bCheckSpecial(Asc("<")) = True
bCheckSpecial(Asc(">")) = True
bCheckSpecial(Asc("(")) = True
bCheckSpecial(Asc(")")) = True
bCheckSpecial(Asc("{")) = True
bCheckSpecial(Asc("}")) = True
bCheckSpecial(Asc("[")) = True
bCheckSpecial(Asc("]")) = True
bCheckSpecial(Asc(", ")) = True
bCheckSpecial(Asc("#")) = True
bCheckSpecial(Asc("@")) = True
bCheckSpecial(Asc("!")) = True
bCheckSpecial(Asc("^")) = True
bCheckSpecial(Asc("%")) = True
bCheckSpecial(Asc("_")) = True

For i = LBound(bCheckWhite()) To UBound(bCheckWhite())
    bWordSep(i) = bCheckWhite(i) OR bCheckSpecial(i)
Next
bWordSep(Asc(".")) = True
End Sub

```

Special functions are used to check for special status. Most of the functions expect an integer argument that corresponds to the ASCII value of the character to test.

Table 9.1: Functions used to identify special characters.

Function	Description
IsWhiteSpace(Integer)	True if the character is a space or tab.
FMT_IsSpecialChar(Integer)	True if the character is special such as a *,;, &.
FMT_IsWordSep(Integer)	True if special character or white space.
FMT_IsDigit(Integer)	True if a numeric digit [0 – 9].
FMT_IsHexDigit(Integer)	True if a numeric digit [0 – 9], [A-F], or [a-f].
FMT_IsOctDigit(Integer)	True if a numeric digit [0 – 7].
FMT_StrIsDigit(String)	True if the first character of the string is a digit.

The functions in *Table 9.1* are implemented in *Listing 9.3*. Most of the functions use an array lookup for speed. Error handling is used in case the ASCII value (UNICODE number) is too large. A numeric digit is checked directly against the ASCII values for a '0' and '9'. The point of these functions was to make them fast.

Listing 9.3: Functions to check for special characters.

```

'*****
'** An array lookup is faster.
'** This assumes that I can get an ASCII Character, which I can

```

```

'** not in Oo 2.0. I can in version 2.01.
'*****
Function FMT_IsWhiteSpace(iChar As Integer) As Boolean
    On Error Resume Next
    FMT_IsWhiteSpace() = False
    FMT_IsWhiteSpace() = bCheckWhite(iChar)
    ' Select Case iChar
    ' Case 9, 10, 13, 32, 160
    '     iIsWhiteSpace = True
    ' Case Else
    '     iIsWhiteSpace = False
    ' End Select
End Function

'*****
'** Return true if the character is a special character.
'*****
Function FMT_IsSpecialChar(iChar As Integer) As Boolean
    On Error Resume Next
    FMT_IsSpecialChar() = False
    FMT_IsSpecialChar() = bCheckSpecial(iChar)
End Function

'*****
'** Return true if the character is a word separator.
'*****
Function FMT_IsWordSep(iChar As Integer) As Boolean
    On Error Resume Next
    FMT_IsWordSep() = False
    FMT_IsWordSep() = bWordSep(iChar)
End Function

'*****
'** Does this character reference the digit 0, 1, 2, ..., or 9?
'*****
Function FMT_IsDigit(iChar As Integer) As Boolean
    FMT_IsDigit() = (48 <= iChar AND iChar <= 57)
End Function

'*****
'** Does this character reference an octal digit 0 through 7.
'*****
Function FMT_IsOctDigit(iChar As Integer) As Boolean
    FMT_IsDigit() = (48 <= iChar AND iChar <= 55)
End Function

'*****

```



```

'*** Does this character reference a hex digit
'*****
Function FMT_IsHexDigit(iChar As Integer) As Boolean
    FMT_IsHexDigit() = FMT_IsDigit(iChar) OR _
        (65 <= iChar AND iChar <= 70) OR _
        (97 <= iChar AND iChar <= 102)
End Function

'*****
'*** Does this character reference the digit 0, 1, 2, ..., or 9?
'*****
Function FMT_StrIsDigit(s$) As Boolean
    FMT_StrIsDigit = FMT_IsDigit(ASC(s))
End Function

'*****
'*** This code is permissive, and assumes valid XML.
'*****
Function FMT_XMLElementEnd(s$, iStart%, iLen%) As Integer
    Dim i%
    Dim sChar$
    For i = iStart To iLen
        sChar$ = Mid(s, i, 1)
        If FMT_IsWhiteSpace(Asc(sChar)) OR sChar = "=" OR _
            sChar = "/" OR sChar = ">" Then
            FMT_XMLElementEnd = i - 1
            Exit Function
        End If
    Next
    FMT_XMLElementEnd = iLen
End Function

```

9.1.1. Special characters and numbers in strings

Special functions are used to quickly find “text of interest”.

Table 9.2: Functions used to find the next relevant special character.

Function	Description
FMT_FindNextNonSpace	Used to quickly skip white space starting from iPos.
FMT_FindEndQuote	Search for the next quote character.
FMT_FindEndQuoteEscape	Search for the next quote character, ignoring any character preceded by a backslash (\) character.
FMT_FindEndQuoteDouble	Search for the next quote character, ignoring any character followed by a quote character. For example, the text """" finds all quote characters.
FMT_FindNumberEnd	Identify a number, including 0xFF

Listing 9.4: Functions to find the next relevant special character.

```
*****
** Increment iPos until it points at the first non-white space
** character; or past the end of the line.
*****
Sub FMT_FindNextNonSpace(sLine$, iPos%, iLen%)
    If iPos <= iLen Then
        REM Position the cursor AFTER the white space.
        Do While FMT_IsWhiteSpace(Asc(Mid(sLine, iPos, 1)))
            iPos = iPos + 1
            If iPos > iLen Then Exit Do
        Loop
    End If
End Sub

*****
** Increment iPos until it points past the closing quote.
** It is not possible to have a quote character in the string.
*****
Sub FMT_FindEndQuote(s$, iPos%, iLen%)
    Dim sQuote$ : sQuote = Mid(s, iPos, 1)

    iPos = iPos + 1
    If iPos <= iLen Then
        Do While Mid(s, iPos, 1) <> sQuote
            iPos = iPos + 1
            If iPos > iLen Then Exit Do
        Loop
        Rem iPos might point two past the string...
        iPos = iPos + 1
    End If
End Sub

*****
** Increment iPos until it points past the closing quote.
** Preceding a quote with \ escapes it.
*****
Sub FMT_FindEndQuoteEscape(s$, iPos%, iLen%)
    Dim sQuote$ : sQuote = Mid(s, iPos, 1)
    Dim sCur$

    iPos = iPos + 1
    If iPos <= iLen Then
        Do
            sCur = Mid(s, iPos, 1)
```

```

        iPos = iPos + 1
        If sCur = "\" Then iPos = iPos + 1
        If iPos > iLen Then Exit Do
    Loop Until sCur = sQuote
End If
End Sub

'*****
'** Increment iPos until it points past the closing quote.
'** Preceding a quote with a quote escapes it.
'*****
Sub FMT_FindEndQuoteDouble(s$, iPos%, iLen%)
    Dim sQuote$ : sQuote = Mid(s, iPos, 1)

    iPos = iPos + 1
    Do While iPos <= iLen
        If Mid(s, iPos, 1) <> sQuote OR iPos = iLen Then
            iPos = iPos + 1
        Else
            REM iPos references a quote character AND we are not
            REM at the end of the string.
            iPos = iPos + 1
            If Mid(s, iPos, 1) <> sQuote Then Exit Do
            REM there were two double quote characters, ignore them.
            iPos = iPos + 1
        End If
    Loop
    REM Never point more than one past the end.
    If iPos > iLen Then iPos = iLen + 1
End Sub

REM This routine is called if, and only if, oCurs is on a number,
REM or a period and a number.
'*****
'** Determine if the cursor is on a number. If it is, then set
'** iEnd to one character past the end of the number and return
'** True. If not, then simply return False.
'** A valid number is as follows:
'** -Number may start with a digit or a decimal point.
'** -Number may have a single decimal point.
'** -e or f are accepted for exponentiation.
'** -Exponents may start with + or -.
'** -Exponents may contain a decimal point.
'*****
Function FMT_FindNumberEnd(sLine$, iPos%, iLen%, iEnd%) As Boolean
    Dim sChar$
    Dim bDecimal As Boolean

```

```

iEnd = iPos
bDecimal = False
sChar = ""
REM Skip leading digits.
Do While FMT_IsDigit(ASC(Mid(sLine, iEnd, 1)))
    iEnd = iEnd + 1
    If iEnd > iLen Then Exit do
Loop

REM Check for hex digits such as 0xFF.
REM No use for Basic, only for other languages.
REM The following must be true:
REM -- found a single character
REM -- more than one character to process
REM -- the single found character must be 0
If iEnd - iPos = 1 AND iEnd < iLen AND Mid(sLine, iPos, 1) = "0" Then
    If Mid(sLine, iEnd, 1) = "x" OR Mid(sLine, iEnd, 1) = "X" Then
        If FMT_IsHexDigit(ASC(Mid(sLine, iEnd + 2, 1))) Then
            FMT_FindNumberEnd() = True
            iEnd = iEnd + 2
            If iEnd <= iLen Then
                Do While FMT_IsHexDigit(ASC(Mid(sLine, iEnd, 1)))
                    iEnd = iEnd + 1
                    If iEnd > iLen Then Exit do
                Loop
            End If
            Exit Function
        End If
    End If
End If

REM Now check for a decimal
If iEnd <= iLen Then
    If Mid(sLine, iEnd, 1) = "." Then
        iEnd = iEnd + 1
        bDecimal = True
        REM Skip trailing digits.
        Do While iEnd <= iLen
            If NOT FMT_IsDigit(ASC(Mid(sLine, iEnd, 1))) Then Exit Do
            iEnd = iEnd + 1
        Loop
    End If
End If

REM If there was just a ".", then iEnd = iPos + 1
If (bDecimal AND iEnd = iPos + 1) OR (iEnd = iPos) Then

```

```

    FMT_FindNumberEnd() = False
    Exit Function
End If

REM This is a number, now look for scientific notation.
FMT_FindNumberEnd() = True
If iEnd <= iLen Then
    sChar = Mid(sLine, iEnd, 1)
    If sChar <> "f" AND sChar <> "e" Then Exit Function
    iEnd = iEnd + 1
End If

REM This is scientific notation, so check for + or -.
If iEnd <= iLen Then sChar = Mid(sLine, iEnd, 1)
If sChar = "+" OR sChar = "-" Then iEnd = iEnd + 1

REM Skip leading digits.
Do While iEnd <= iLen
    If NOT FMT_IsDigit(ASC(Mid(sLine, iEnd, 1))) Then Exit Do
    iEnd = iEnd + 1
Loop

REM Now check for a decimal
If iEnd <= iLen Then
    If Mid(sLine, iEnd, 1) = "." Then
        iEnd = iEnd + 1
        REM Skip trailing digits.
        REM They really should be zeros if they exist.
        Do While iEnd <= iLen
            If NOT FMT_IsDigit(ASC(Mid(sLine, iEnd, 1))) Then Exit Do
            iEnd = iEnd + 1
        Loop
    End If
End If
End Function

```

I use the following macro to test my special functions. I used the same code to time the functions for optimization purposes.

Listing 9.5: Test code for special functions.

```

Sub TestSpecialChars
    Dim ss$
    Dim n%
    Dim s$ : s = "how are you?"
    Dim iPos% : iPos = 4
    Dim iLen% : iLen = Len(s)
    Dim x
    Dim i As Long

```

```

Dim nIts      As Long
Dim nMinIts As Long : nMinIts = 1000
Dim nNow      As Long
Dim nNum      As Long : nNum    = 2000
Dim nFirst    As Long : nFirst  = GetSystemTicks()
Dim nLast     As Long : nLast   = nFirst + nNum
Dim iEnd%
Dim b As Boolean

ss = "1.3f+3.2x"
iPos = 1
b = FMT_FindNumberEnd(ss$, iPos%, Len(ss), iEnd%)
Print ss & " ==> " & b & " end = " & iEnd
'Exit Sub

FMT_InitSpecialCharArrays()

Do
    For i = 1 To nMinIts
        FMT_FindNextNonSpace(s, iPos%, iLen%)
    Next
    nIts = nIts + 1
    nNow = GetSystemTicks()
Loop Until nNow >= nLast
MsgBox "Finished with " & CStr(nIts * nMinIts) & _
    " Iterations" & CHR$(10) & _
    CStr(CDbl(nIts * nMinIts) * 1000 / CDbl(nNow - nFirst)) & _
    " its/second"

End Sub

```

9.1.2. Arrays of strings

The macro has lists of strings that I search. I sort each array (using a bubble sort) so that I can use a binary search, which is much faster than a linear search.

Listing 9.6: Array functions.

```

'*****
'*** Sort the sItems() array in ascending order.
'*** The algorithm is simple and inefficient.
'*** The worst case runtime is O(n^2). If you bother to do the
'*** math, you will get (n^2-4n+1)/2, not that it matters.
'*****

Sub FMT_SortStringArrayAscending(sItems())
    Dim i As Integer      'Outer index variable
    Dim j As Integer      'Inner index variable
    Dim s As String        'Temporary to swap two values.
    Dim bChanged As Boolean 'Becomes True when something changes
    For i = LBound(sItems()) To UBound(sItems()) - 1

```

```

        bChanged = False
    For j = UBound(sItems()) To i+1 Step -1
        If sItems(j) < sItems(j-1) Then
            s = sItems(j) : sItems(j) = sItems(j-1) : sItems(j-1) = s
            bChanged = True
        End If
    Next
    If Not bChanged Then Exit For
Next
End Sub

'*****
'** Determine if an array contains a specific string.
'** Although a binary search is faster for large arrays, I
'** expect small arrays here, so a linear search might be faster.
'*****
Function FMT_ArrayHasString(s$, sItems()) As Boolean
    Dim i As Integer
    Dim iUB As Integer
    Dim iLB As Integer
    FMT_ArrayHasString() = False

    iUB = UBound(sItems())
    iLB = LBound(sItems())
    Do
        i = (iUB + iLB) \ 2
        If sItems(i) = s Then
            FMT_ArrayHasString() = True
            iLB = i + 1
            'Exit Do
        ElseIf sItems(i) > s Then
            iUB = i - 1
        Else
            iLB = i + 1
        End If
    Loop While iUB >= iLB
End Function

'*****
'** Insert a string in an array in formatted order.
'** n% is the number of items in the array.
'** The array is assumed to have room for one more item.
'** Return True if s$ is inserted into the array, and False if
'** the item is not inserted into the array. If it is not inserted,
'** this means that it was already there (so it avoids duplicates).
'*****
Function FMT_InsertSortedArray(n%, s$, sItems()) As Boolean

```

```

Dim i As Integer
Dim j As Integer
i = FMT_IndexInArray(n%, s$, sItems())
If i >= n Then
    FMT_InsertSortedArray = True
    sItems(n) = s
    Exit Function
End If
If sItems(i) = s Then
    FMT_InsertSortedArray = False
    Exit Function
End If
For j = n To i+1 Step -1
    sItems(j) = sItems(j-1)
Next
sItems(i) = s
FMT_InsertSortedArray = True
End Function

'*****
'** Find a string in an array. n% is the number of items in the array.
'** Return the index where the string should be.
'*****
Function FMT_IndexInArray(n%, s$, sItems()) As Integer
    Dim i As Integer
    Dim iUB As Integer
    Dim iLB As Integer
    If n = 0 Then
        FMT_IndexInArray() = n
        Exit Function
    End If

    iUB = n - 1
    iLB = LBound(sItems())
    Do
        i = (iUB + iLB) \ 2
        If sItems(i) = s Then
            FMT_IndexInArray() = i
            Exit Function
        'iLB = iUB + 1
        ElseIf sItems(i) > s Then
            iUB = i - 1
        Else
            iLB = i + 1
        End If
    Loop While iUB >= iLB
    FMT_IndexInArray() = iLB

```


End Function

9.2. Utilities Module

9.2.1. Where does the code start?

When I place a macro into a document, each line is in a separate paragraph. Specific paragraph styles identify a paragraph as a “line of code”. After placing a cursor in text containing a macro, I need to find the first and last line (paragraph) of the macro.

The first macro accepts a text cursor and a list of style names used to identify a paragraph as code. The cursor is moved backwards through the paragraphs until a paragraph style is found that is not code. If required, the cursor is then moved forward one paragraph to move it back into a code paragraph. The last paragraph containing code is found in a similar manner.

Listing 9.7: Find the area around the cursor containing code.

```
'*****
'** Move the cursor to the first paragraph using a code specific
'** style.
'*****
Sub FMT_CursorToFirstCodeParagraph(oCurs, sStyles())

    oCurs.gotoStartOfParagraph(False)
'    This check should be done before calling this routine.
'    If NOT FMT_ArrayHasString(oCurs.ParaStyleName, sStyles()) Then
'        Print "The text cursor must be in a code segment"
'        Exit sub
'    End If

    REM Find the first paragraph that is computer code.
    Do While FMT_ArrayHasString(oCurs.ParaStyleName, sStyles())
        If NOT oCurs.gotoPreviousParagraph(False) Then Exit Do
    Loop

    If NOT FMT_ArrayHasString(oCurs.ParaStyleName, sStyles()) Then
        oCurs.gotoNextParagraph(False)
    End If
    oCurs.gotoStartOfParagraph(False)
End Sub

'*****
'** Move the cursor to the last paragraph using a code specific
'** style.
'*****
Sub FMT_CursorToLastCodeParagraph(oCurs, sStyles())

    REM Find the last paragraph that is part of the computer code.
```

```

Do While FMT_ArrayHasString(oCurs.ParaStyleName, sStyles())
    If NOT oCurs.gotoNextParagraph(False) Then Exit Do
Loop

If NOT FMT_ArrayHasString(oCurs.ParaStyleName, sStyles()) Then
    oCurs.gotoPreviousParagraph(False)
End If
oCurs.gotoEndOfParagraph(False)
End Sub

'*****
'** Given a cursor, select a range around the cursor that
'** uses the required paragraph styles.
'*****
Sub FMT_FindCodeAroundCursor(oCurs, sStyles())
    Dim oEndCurs

    REM Find the last paragraph that is part of the computer code.
    oEndCurs = oCurs.getText().createTextCursorByRange(oCurs)
    FMT_CursorToLastCodeParagraph(oEndCurs, sStyles())
    FMT_CursorToFirstCodeParagraph(oCurs, sStyles())

    REM Select the entire thing and then format it.
    oCurs.gotoRange(oEndCurs, True)
End Sub

```

The last macro in Listing 9.7 is used to select the entire macro listing. The first and last paragraphs are found, and then a cursor selects the entire range. It is assumed that the cursor starts in a code paragraph.

9.2.2. Stacks

XML is recursive in nature, so I use a stack to track certain things related to the XML. I use the stack routines in Listing 9.8.

Listing 9.8: *Treat an array like a stack.*

```

'*****
'** Treat the array iStack() as a stack. It is assumed that the
'** stack is already dimensioned. The value x is added, and n is
'** incremented.
'*****
Sub FMT_PushStack(iStack%(), n%, x%)
    If n = UBound(iStack()) Then
        ReDim Preserve iStack(0 To n + 100) As Integer
    End If
    iStack(n) = x
    n = n + 1
End Sub

```

```

'*****
'** Treat the array iStack() as a stack. Pull the last value from
'** the stack.
'*****

Function FMT_PopStack(iStack%(), n%) As Integer
    If n > LBound(iStack()) Then
        n = n - 1
    End If
    FMT_PopStack = iStack(n)
End Function

Function FMT_PopStackUntil(iStack%(), n%, x%) As Integer
    Do While n > LBound(iStack())
        n = n - 1
        If iStack(n) = x Then Exit Do
    Loop
    FMT_PopStackUntil = iStack(n)
End Function

'*****
'** Treat the array iStack() as a stack. Peek at the last value on
'** the stack.
'*****

Function FMT_PeekStack(iStack%(), n%) As Integer
    If n > LBound(iStack()) Then
        FMT_PeekStack = iStack(n-1)
    Else
        FMT_PeekStack = -1
    End If
End Function

```

9.2.3. Set a character style

Processing is typically done relative to the start of the line. The macro in Listing 9.8 moves the cursor forward nSkip characters, then highlights the next nChars. The highlighted characters are set to use the style at index nStyle from the Styles array.

Listing 9.9: *Format a range of characters with a style.*

```

'*****
'** Jump forward nSkip chars, then set nchars to the specified
'** character style.
'*****

Sub FMT_SetStyleForChars(oCurs, nSkip%, nChars%, nStyle, Styles())
    oCurs.goRight(nSkip, False)
    oCurs.goRight(nChars, True)
    oCurs.CharStyleName = Styles(nStyle)

```

```

    oCurs.gotoStartOfParagraph(False)
End Sub

```

The primary working model is to move one paragraph at a time through the text. If less than an entire paragraph is selected, then this must be accounted for. The first paragraph is selected as shown below. The variable oSel contains the entire selection. The variable oCurs starts at the selection start and then jumps to the end of the paragraph. If too much was selected, then the entire selection is used.

```

'*****
'** Create a text cursor that starts where the selection starts
'** and then selects either the entire selection, or the rest of
'** the current paragraph, which ever is smaller.
'*****
Function FMT_CreateStartCursor(oText, oSel)
    Dim oCurs
    oCurs = oText.createTextCursorByRange(oSel.getStart())
    oCurs.gotoEndOfParagraph(True)
    If oText.compareRegionEnds(oCurs, oSel) = -1 Then
        oCurs.gotoRange(oSel, False)
    End If
    FMT_CreateStartCursor = oCurs
End Function

```

All following paragraphs are formatted by jumping to the end of the next paragraph. If that is too much, then the area from the paragraph start to the selection end is used.

```

Function FMT_CursorToParEnd(oText, oSel, oCurs) As Boolean
    ' This should never happen, but, if it does, fail out.
    If NOT oCurs.gotoEndOfParagraph(True) Then
        FMT_CursorToParEnd = False
        Exit Function
    End If
    ' Oops, we went to far, try to backup.
    If oText.compareRegionEnds(oCurs, oSel) = -1 Then
        oCurs.gotoStartOfParagraph(False)
        oCurs.gotoRange(oSel.getEnd(), True)
    End If
    FMT_CursorToParEnd = True
End Function

```

9.2.4. Create a property

Many routines accept an array of property values. The CreateProperty routine is a convenience method.

Listing 9.10: *Create a PropertyValue.*

```

'*****
'** Create and return a PropertyValue structure.

```

```

! *****
Function CreateProperty( Optional cName As String, Optional uValue ) As
com.sun.star.beans.PropertyValue
    Dim oPropertyValue As New com.sun.star.beans.PropertyValue
    If Not IsMissing( cName ) Then
        oPropertyValue.Name = cName
    EndIf
    If Not IsMissing( uValue ) Then
        oPropertyValue.Value = uValue
    EndIf
    CreateProperty() = oPropertyValue
End Function

```

9.2.5. Find a text document

The colorize macros assume that a text document is current. This macro verifies that ThisComponent exists and is a text document. ThisComponent may contain a "bad" value. To see this, (1) focus the IDE, (2) focus a text document, (3) close the text document so that the IDE has focus. Now, run your macro. If ThisComponent is not valid, then search until a valid text document is found.

```

Function FMT_FindTextDoc()
    ' At least one component appears to NOT support the SupportsService
    ' method - because the call fails sometimes.
    On Error Resume Next
    Dim oDocs As Object
    Dim oDoc As Object

    FMT_FindTextDoc() = oDoc
    Dim x As Boolean
    x = False
    x = ThisComponent.SupportsService("com.sun.star.text.TextDocument")
    If x Then
        FMT_FindTextDoc() = ThisComponent
        Exit Function
    End If

    oDocs = StarDesktop.getComponents().createEnumeration()
    Do While oDocs.hasMoreElements()
        oDoc = oDocs.nextElement()
        x = oDoc.SupportsService("com.sun.star.text.TextDocument")
        If x Then
            FMT_FindTextDoc() = oDoc
            Exit Function
        End If
    Loop
End Function

```

9.3. Styles Module

Code to be colorized is assumed to be organized as paragraphs to colorize using specific paragraph styles. The colorizer has a few different styles. Listings with no margins and small characters (to save space) are expected to be formatted using the paragraph styles as shown below:

```
_code_first_line    ' Space above this style and no space below.
_code              ' No space above or below this style.
_code
_code_last_line    ' Space below this style and no space above.
```

I format code in this way when I am short on space or the code listing is long. If I have a single line that I want to format, then I use a style with no indent, small font, and space above and below the listing.

```
_code_one_line      ' Space above and below.
```

The paragraph styles that start with `_OOoComputerCode` indent the paragraph so that it stands out and uses a slightly larger font than the `_code` styles.

```
_OOoComputerCode      'No space above and little space below.
_OOoComputerCode
_OOoComputerCodeLastLine 'No space above, but has space below.
```

The style meant for use in a text table has smaller indents.

```
_OOoComputerCodeInTable
```

The text in the paragraphs is incoded using character styles. The character styles do not set the font size so that it uses what ever font size is set in the containing paragraph style.

9.3.1. Create character styles

The primary code parses the text looking for comments, identifiers, keywords, and literals. Character styles are used to format and color code each portion of the code. The character styles are created in the document if they do not exist. Note that the parent style is not set if it does not already exist in the document.

Listing 9.11: Creating a character style.

```
'*****
** Create a character style if it does not exist.
**
'*****
Sub CreateCharacterStyle(sStyleName$, oProps())
    Dim i%
    Dim oFamilies
    Dim oStyle
    Dim oStyles

    oFamilies = ThisComponent.StyleFamilies
    oStyles = oFamilies.GetByName("CharacterStyles")
```

```

If oStyles.HasByName(sStyleName) Then
    'PrintColor(oStyles.GetByName(sStyleName).CharColor)
Exit Sub
End If
oStyle = ThisComponent.CreateInstance("com.sun.star.style.CharacterStyle")
For i=LBound(oProps) To UBound(oProps)
    If oProps(i).Name = "ParentStyle" Then

        If oStyles.HasByName(oProps(i).Value) Then
            oStyle.ParentStyle = oProps(i).Value
        Else
            Print "Parent character style (" & oProps(i).Value & _
                ") does not exist, ignoring parent."
        End If
        oStyle.ParentStyle = oProps(i).Value
    Else
        oStyle.SetPropertyValue(oProps(i).Name, oProps(i).Value)
    End If
Next
oStyles.InsertByName(sStyleName, oStyle)
End Sub

```

The code to create the specific character styles is shown here:

Listing 9.12: Force the required character styles to exist

```

'*****
'** Create base character styles
'*****
Sub CreateBaseCharStyles
    Dim oProps()

    REM Base style for all.
    REM computer code that is not color coded and used in regular text
    REM uses this style.
    oProps() = Array(CreateProperty("CharFontName", "Courier"), _
        CreateProperty("CharColor", RGB(0, 0, 0)), _
        CreateProperty("CharNoHyphenation", True) )

    CreateCharacterStyle("OOoComputerCode", oProps())

    REM Base style for normal listings.
    oProps() = Array(CreateProperty("ParentStyle", "OOoComputerCode"))

    CreateCharacterStyle("_OOoComputerBase", oProps())
End Sub

'*****
'** Create character styles for StarBasic using the same colors

```

```

'** as the Oo IDE.
'*****

Sub CreateStarBasicCharStyles
    Dim oProps()

    REM If you do not want something to have a language, which prevents
    REM a spell check, set CharLocale to noLocale.
    Dim noLocale As New com.sun.star.lang.Locale
    noLocale.Country = ""
    noLocale.Language = "zxx"

    CreateBaseCharStyles()

    oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
        CreateProperty("CharColor", RGB(76, 76, 76)))
    CreateCharacterStyle("_OOoComputerComment", oProps())

    oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
        CreateProperty("CharColor", RGB(255, 0, 0)))
    CreateCharacterStyle("_OOoComputerLiteral", oProps())

    oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
        CreateProperty("CharLocale", noLocale), _
        CreateProperty("CharColor", RGB(0, 0, 128)))
    CreateCharacterStyle("_OOoComputerKeyWord", oProps())

    oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
        CreateProperty("CharColor", RGB(0, 128, 0)))
    CreateCharacterStyle("_OOoComputerIdent", oProps())
End Sub

'*****
'** Create character styles for Java using the same colors as Eclipse
'*****

Sub CreateJavaCharStyles
    Dim oProps()

    REM If you do not want something to have a language, which prevents
    REM a spell check, set CharLocale to noLocale.
    Dim noLocale As New com.sun.star.lang.Locale
    noLocale.Country = ""
    noLocale.Language = "zxx"

    CreateBaseCharStyles()

    oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
        CreateProperty("CharColor", RGB(153, 204, 255)))

```



```

CreateCharacterStyle("_JavaComment", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharColor", RGB(0, 0, 255)))
CreateCharacterStyle("_JavaLiteral", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharLocale", noLocale), _
    CreateProperty("CharColor", RGB(153, 40, 76)))
CreateCharacterStyle("_JavaKeyword", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharColor", RGB(0, 0, 0)))
CreateCharacterStyle("_JavaIdent", oProps())
End Sub

' *****
' ** Create character styles for StarBasic using the same colors
' ** as the OOo IDE.
' *****
Sub CreateXMLCharStyles
    Dim oProps()

    REM If you do not want something to have a language, which prevents
    REM a spell check, set CharLocale to noLocale.
    Dim noLocale As New com.sun.star.lang.Locale
    noLocale.Country = ""
    noLocale.Language = "zxx"

    CreateBaseCharStyles()

    oProps() = Array( CreateProperty("ParentStyle", "_OOoComputerBase") )
    CreateCharacterStyle("_XMLContent", oProps())

    oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
        CreateProperty("CharColor", RGB(76, 76, 76)))
    CreateCharacterStyle("_XMLComment", oProps())

    oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
        CreateProperty("CharColor", RGB(255, 0, 0)))
    CreateCharacterStyle("_XMLLiteral", oProps())

    oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
        CreateProperty("CharLocale", noLocale), _
        CreateProperty("CharColor", RGB(0, 0, 128)))
    CreateCharacterStyle("_XMLAttribute", oProps())

```

```

'Entity is dark golden rod. I could use golden rod instead (218, 165, 32).
oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharLocale", noLocale), _
    CreateProperty("CharColor", RGB(184, 134, 11)))
CreateCharacterStyle("_XMLEntity", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharLocale", noLocale), _
    CreateProperty("CharColor", RGB(0, 128, 0)))
CreateCharacterStyle("_XMLKeyword", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharLocale", noLocale), _
    CreateProperty("CharColor", RGB(138, 43, 226)))
CreateCharacterStyle("_XMLBracket", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharLocale", noLocale), _
    CreateProperty("CharWeight", com.sun.star.awt.FontWeight.BOLD))
CreateCharacterStyle("_XMLElement", oProps())
End Sub

```

The original styles all set a parent style, even on the top level styles. They also used to set a font size.

9.3.2. Create paragraph styles

Creating a paragraph style is very similar to creating a character style.

Listing 9.13: Creating a paragraph style

```

'*****
'** Create a paragraph style if it does not exist.
'*****
Sub CreateParStyle(sStyleName$, oProps())
    Dim i%, j%
    Dim oFamilies
    Dim oStyle
    Dim oStyles
    Dim tabStops%

    oFamilies = ThisComponent.StyleFamilies
    oStyles = oFamilies.getByName("ParagraphStyles")
    If oStyles.HasByName(sStyleName) Then
        Exit Sub
    End If
    oStyle = ThisComponent.createInstance("com.sun.star.style.ParagraphStyle")
    For i=LBound(oProps) To UBound(oProps)
        If oProps(i).Name = "ParentStyle" Then

```

```

If oStyles.HasByName(oProps(i).Value) Then
    oStyle.ParentStyle = oProps(i).Value
Else
    Print "Parent paragraph style (" & oProps(i).Value & _
        ") does not exist, ignoring parent"
End If
ElseIf oProps(i).Name = "ParaTabStops" Then
    tabStops = oProps(i).Value
    Dim tab(0 To 19) As New com.sun.star.style.TabStop
    For j = LBound(tab) To UBound(tab)
        tab(j).Alignment = com.sun.star.style.TabAlign.LEFT
        tab(j).DecimalChar = ASC(".")
        tab(j).FillChar = 32
        tab(j).Position = (j+1) * tabStops
    Next
    oStyle.ParaTabStops = tab
ElseIf oProps(i).Name = "FollowStyle" Then
    If oStyles.HasByName(oProps(i).Value) OR oProps(i).Value = sStyleName Then
        oStyle.setPropertyValue(oProps(i).Name, oProps(i).Value)
    Else
        Print "Next paragraph style (" & oProps(i).Value & _
            ") does not exist, ignoring for style " & sStyleName
    End If
Else
    oStyle.setPropertyValue(oProps(i).Name, oProps(i).Value)
End If
Next
oStyles.insertByName(sStyleName, oStyle)

End Sub

```

Sometimes the code failed creating a style. The failures went away when I used typed values such as CLng(0) rather than just 0. If your code fails, therefore, consider using a specific type as specified in the IDL rather than letting Basic convert the type for you.

Listing 9.14: Creating specific paragraph styles

```

Sub CreateParStyles
    Dim oProps()
    Dim tabStopLoc%
    Dim sNextStyle$
    Dim sFontName$
    Dim fParSmallCharHeight As double
    Dim fParNormalCharHeight As double
    Dim oDoc
    Dim oConfigAccess
    Dim i%

    oDoc = FMT_FindTextDoc()

```

```

oConfigAccess = FMT_ConfigAccessStyles(False)
sNextStyle = getNextStyleName(oConfigAccess)
If NOT DocHasParStyle(oDoc, sNextStyle) Then
    i = MsgBox("Current document does not have paragraph style '" & _
        sNextStyle & "' would you like to configure the formatter?", _
        35
If i <> 6 Then
    Exit Sub
End If
RunCfgFmtDlg()
sNextStyle = getNextStyleName(oConfigAccess)
If NOT DocHasParStyle(oDoc, sNextStyle) Then
    MsgBox("Current document does not have paragraph style '" & _
        sNextStyle & "' Exiting now."
    Exit Sub
End If
End If

fParSmallCharHeight = getSmallCharHeight(oConfigAccess)
fParNormalCharHeight = getNormalCharHeight(oConfigAccess)

sFontName = getFontName(oConfigAccess)
If NOT DocHasFontName(oDoc, sFontName) Then
    i = MsgBox("Document does not have font name '" & _
        sName & "' would you like to configure the formatter?", _
        35
If i = 7 Then
    Exit Sub
End If
If i = 6 Then
    RunCfgFmtDlg()
    sFontName = getFontName(oConfigAccess)
End If
RunCfgFmtDlg()
sFontName = getFontName(oConfigAccess)
If NOT DocHasFontName(oDoc, sFontName) Then
    i = MsgBox("Document still does not have font name '" & _
        sName & "' would you like to continue?", _
        35
If i <> 6 Then
    Exit Sub
End If
End If
End If

REM Tab stops are set in the paragraph style
' 1/4 of an inch

```

```

'tabStopLoc% = 2540 / 4
tabStopLoc% = getTabWidth(oConfigAccess)

REM Main paragraph stle for "small" text.
REM There is no space above or below the stlye.
REM The first line of code uses _code_first_line
REM The last line of code uses _code_last_line.
oProps() = Array(CreateProperty("ParaTopMargin", CLng(0)), _
    CreateProperty("ParaBottomMargin", CLng(0)), _
    CreateProperty("ParaLeftMargin", CLng(0)), _
    CreateProperty("ParaRightMargin", CLng(0)), _
    CreateProperty("ParaFirstLineIndent", CLng(0)), _
    CreateProperty("FollowStyle", "_code"), _
    CreateProperty("CharFontName", sFontName), _
    CreateProperty("CharFontStyleName", "Bold"), _
    CreateProperty("ParaTabStops", tabStopLoc), _
    CreateProperty("ParaLineNumberCount", False), _
    CreateProperty("WritingMode", com.sun.star.text.WritingMode.LR_TB), _
    CreateProperty("CharAutoKerning", False), _
    CreateProperty("CharWeight", 150.0), _
    CreateProperty("CharHeight", fParSmallCharHeight) )
CreateParStyle("_code", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_code"), _
    CreateProperty("ParaTopMargin", CLng(2540 * 0.05)), _
    CreateProperty("ParaBottomMargin", CLng(0)), _
    CreateProperty("ParaLeftMargin", CLng(0)), _
    CreateProperty("ParaRightMargin", CLng(0)), _
    CreateProperty("ParaFirstLineIndent", CLng(0)), _
    CreateProperty("CharFontName", sFontName), _
    CreateProperty("CharFontStyleName", "Bold"), _
    CreateProperty("ParaTabStops", tabStopLoc), _
    CreateProperty("ParaLineNumberCount", False), _
    CreateProperty("WritingMode", com.sun.star.text.WritingMode.LR_TB), _
    CreateProperty("CharAutoKerning", False), _
    CreateProperty("CharWeight", 150.0), _
    CreateProperty("CharHeight", fParSmallCharHeight), _
    CreateProperty("FollowStyle", "_code") )
CreateParStyle("_code_first_line", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_code"), _
    CreateProperty("ParaTopMargin", CLng(0)), _
    CreateProperty("ParaBottomMargin", CLng(2540 * 0.05)), _
    CreateProperty("ParaLeftMargin", CLng(0)), _
    CreateProperty("ParaRightMargin", CLng(0)), _
    CreateProperty("ParaFirstLineIndent", CLng(0)), _
    CreateProperty("CharFontName", sFontName), _

```

```

CreateProperty("CharFontStyleName", "Bold"), _
CreateProperty("ParaTabStops", tabStopLoc), _
CreateProperty("ParaLineNumberCount", False), _
CreateProperty("WritingMode", com.sun.star.text.WritingMode.LR_TB), _
CreateProperty("CharAutoKerning", False), _
CreateProperty("CharWeight", 150.0), _
CreateProperty("CharHeight", fParSmallCharHeight), _
CreateProperty("FollowStyle", sNextStyle) )
CreateParStyle("_code_last_line", oProps())

oProps() = Array(CreateProperty("ParaTopMargin", CLng(0)), _
CreateProperty("ParaBottomMargin", CLng(2540 * 0.03)), _
CreateProperty("ParaLeftMargin", CLng(2540 * 0.20)), _
CreateProperty("ParaRightMargin", CLng(0)), _
CreateProperty("ParaFirstLineIndent", CLng(0)), _
CreateProperty("CharFontName", sFontName), _
CreateProperty("ParaTabStops", tabStopLoc), _
CreateProperty("ParaLineNumberCount", False), _
CreateProperty("WritingMode", com.sun.star.text.WritingMode.LR_TB), _
CreateProperty("CharAutoKerning", False), _
CreateProperty("CharHeight", fParNormalCharHeight) )
CreateParStyle("_OOoComputerCode", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerCode"), _
CreateProperty("ParaTopMargin", CLng(0)), _
CreateProperty("ParaBottomMargin", CLng(2540 * 0.10)), _
CreateProperty("ParaLeftMargin", CLng(2540 * 0.20)), _
CreateProperty("ParaRightMargin", CLng(0)), _
CreateProperty("ParaFirstLineIndent", CLng(0)), _
CreateProperty("CharFontName", sFontName), _
CreateProperty("ParaTabStops", tabStopLoc), _
CreateProperty("ParaLineNumberCount", False), _
CreateProperty("WritingMode", com.sun.star.text.WritingMode.LR_TB), _
CreateProperty("CharAutoKerning", False), _
CreateProperty("CharHeight", fParNormalCharHeight), _
CreateProperty("FollowStyle", sNextStyle) )
CreateParStyle("_OOoComputerCodeLastLine", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerCode"), _
CreateProperty("ParaTopMargin", CLng(0)), _
CreateProperty("ParaBottomMargin", CLng(2540 * 0.03)), _
CreateProperty("ParaLeftMargin", CLng(2540 * 0.10)), _
CreateProperty("ParaRightMargin", CLng(2540 * 0.10)), _
CreateProperty("ParaFirstLineIndent", CLng(0)), _
CreateProperty("CharFontName", sFontName), _
CreateProperty("ParaTabStops", tabStopLoc), _
CreateProperty("ParaLineNumberCount", False), _

```

```

CreateProperty("WritingMode", com.sun.star.text.WritingMode.LR_TB), _
CreateProperty("CharAutoKerning", False), _
CreateProperty("CharHeight", fParNormalCharHeight), _
CreateProperty("FollowStyle", "_OOoComputerCodeInTable" )
CreateParStyle("_OOoComputerCodeInTable", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerCode"), _
CreateProperty("ParaTopMargin", CLng(0)), _
CreateProperty("ParaBottomMargin", CLng(2540 * 0.08)), _
CreateProperty("ParaLeftMargin", CLng(2540 * 0.10)), _
CreateProperty("ParaRightMargin", CLng(2540 * 0.10)), _
CreateProperty("ParaFirstLineIndent", CLng(0)), _
CreateProperty("CharFontName", sFontName), _
CreateProperty("ParaTabStops", tabStopLoc), _
CreateProperty("ParaLineNumberCount", False), _
CreateProperty("WritingMode", com.sun.star.text.WritingMode.LR_TB), _
CreateProperty("CharAutoKerning", False), _
CreateProperty("CharHeight", fParNormalCharHeight), _
CreateProperty("FollowStyle", sNextStyle) )
CreateParStyle("_OOoComputerCodeLastLineInTable", oProps())
End Sub

```

9.4. Basic Module

This section contains the code to format Basic code. Learn how the Basic module works, and you will understand the others.

Character styles are used to format the macro. The character style names are obtained from a function. The order is important, because the styles are extracted based on their location in the array. The constants identify the order; for example, comments are formatted with the style at index 0, which is `_OooComputerComment`.

Listing 9.15: Styles for Basic

```

REM Character Style Name Index
Public const FMT_CSNI_Comment = 0
Public const FMT_CSNI_Literal = 1
Public const FMT_CSNI_KeyWord = 2
Public const FMT_CSNI_Ident = 3
Public const FMT_CSNI_Special = 4
'*****
'** Get the character styles meant for highlighting StarBasic code.
'** Special characters are formatted as a keyword.
'*****

Function FMT_GetBasicCharacterStyles()
CreateStarBasicCharStyles()
FMT_GetBasicCharacterStyles() = Array( "_OOoComputerComment", _
    "_OOoComputerLiteral", "_OOoComputerKeyWord", _
    "_OOoComputerIdent", "_OOoComputerKeyWord")

```

End Function

Keywords and paragraph style names are stored in sorted arrays. The arrays are built using subroutines. It is easy, therefore, to modify the code to recognize new tokens.

Listing 9.16: Initialize keywords and paragraph style arrays.

```
*****
'*** The following words are tokens recognized by the Basic IDE.
'*** This list is in alphabetical order. I got this list from
'*** the file: basic/source/comp/tokens.cxx.
'*** Multi-word tokens such as "end enum" are redundant because
'*** the code recognizes single words. Both words are in the list
'*** already, so in the worst case, this will simply slow down
'*** the search because there are extra words.
*****
Sub FMT_InitTokensBasic(sTokens())
    sTokens() = Array("access", _
        "alias", "and", "any", "append", "as", _
        "base", "binary", "boolean", "byref", "byval", _
        "call", "case", "cdecl", "classmodule", "close", _
        "compare", "compatible", "const", "currency", _
        "date", "declare", "defbool", "defcur", "defdate", _
        "defdbl", "deferr", "defint", "deflng", "defobj", _
        "defsng", "defstr", "defvar", "dim", "do", "double", _
        "each", "else", "elseif", "end", _
        "end enum", "end function", "end if", "end property", _
        "end select", "end sub", "end type", _
        "endif", "enum", "eqv", "erase", "error", _
        "exit", "explicit", _
        "for", "function", _
        "get", "global", "gosub", "goto", _
        "if", "imp", "implements", "in", "input", _
        "integer", "is", _
        "let", "lib", "line", "line input", "local", _
        "lock", "long", _
        "loop", "lprint", "lset", _
        "mod", "msgbox", _
        "name", "new", "next", "not", _
        "object", "on", "open", "option", _
        "optional", "or", "output", _
        "paramarray", "preserve", "print", _
        "private", "property", "public", _
        "random", "read", "redim", "rem", "resume", _
        "return", "rset", _
        "select", "set", "shared", "single", "static", _
        "step", "stop", "string", "sub", "system", _
        "text", "then", "to", "type", "typeof", _
        "until", _
```



```

        "variant", _
        "wend", "while", "with", "write", _
        "xor")
    FMT_SortStringArrayAscending(sTokens())
End Sub

'*****
'** Code listings are formatted using specific paragraph styles.
'** The relevant paragraph styles are listed here.
'*****
Sub FMT_InitParStyles(sStyles())
    sStyles() = Array( "_OOoComputerCode", _
        "_OOoComputerCodeInTable", _
        "_OOoComputerCodeLastLine", _
        "_code", _
        "_code_first_line", _
        "_code_last_line", _
        "_code_one_line")
    FMT_SortStringArrayAscending(sStyles())
End Sub

```

9.4.1. Use the macros

The macro that I use the most, adds color highlighting to the macro containing the cursor. The macro is very simple.

1. Initialize arrays.
2. Verify that the view cursor is in a macro; by checking the paragraph style.
3. Find the first and last paragraph (see *Listing 9.7*).
4. Use the main worker macro (see *Listing 9.20*).

The hard work is in the main worker macro.

Listing 9.17: *Format the current macro.*

```

'*****
'** Color code the Basic code surrounding the view cursor.
'*****
Sub FMT_ColorCodeCurrentBasic()
    Dim oVCurs
    Dim oCurs
    Dim sStyles()      ' Paragraph styles for code listings.
    Dim sTokens()      ' Keyword tokens.
    Dim sCharStyles() : sCharStyles() = FMT_GetBasicCharacterStyles()

    FMT_InitSpecialCharArrays()

```

```

FMT_InitParStyles(sStyles())
FMT_InitTokensBasic(sTokens())

REM Get the view cursor as the starting location
oVCurs = ThisComponent.getCurrentController().getViewCursor()
oCurs = oVCurs.getText().createTextCursorByRange(oVCurs)
If NOT FMT_ArrayHasString(oCurs.ParaStyleName, sStyles()) Then
    MsgBox "No code found" & CHR$(10) & _
        "You must use a 'CODE' paragraph style" & CHR$(10) & _
        "AndrewMacro.odt uses these styles" & CHR$(10) & _
        "-----" & CHR$(10) & _
        Join(sStyles(), CHR$(10))
    Exit sub
End If
FMT_FindCodeAroundCursor(oCurs, sStyles())
FMT_ColorCodeOneRangeStringBasic(oCurs, sTokens(), sCharStyles())
End Sub

```

This next macro finds and format all macros in the current document.

Listing 9.18: Highlight all macros in the document.

```

REM Highlight all code in this document.
Sub HighlightDoc()
    Dim oCurs
    Dim oStartCurs
    Dim oEndCurs
    Dim bFoundCompStyle As Boolean
    Dim sStyles()
    Dim sTokens()
    Dim sCharStyles() : sCharStyles() = FMT_GetBasicCharacterStyles()

    FMT_InitSpecialCharArrays()
    FMT_InitParStyles(sStyles())
    FMT_InitTokensBasic(sTokens())
    bFoundCompStyle = False

    oCurs = ThisComponent.getText().createTextCursor()
    oStartCurs = ThisComponent.getText().createTextCursor()
    oEndCurs = ThisComponent.getText().createTextCursor()
    oEndCurs.gotoStart(False)
    Do
        If FMT_ArrayHasString(oEndCurs.ParaStyleName, sStyles()) Then
            If NOT bFoundCompStyle Then
                bFoundCompStyle = True
                oCurs.gotoRange(oEndCurs, False)
                oCurs.gotoEndOfParagraph(True)
            Else
                oCurs.gotoNextParagraph(True)
            End If
        End If
    Loop
End Sub

```

```

        oCurs.gotoEndOfParagraph(True)
    End If
Else
    If bFoundCompStyle Then
        bFoundCompStyle = False
        FMT_ColorCodeOneRangeStringBasic(oCurs, sTokens(), sCharStyles())
    End If
End If
Loop While oEndCurs.gotoNextParagraph(False)
If bFoundCompStyle Then
    bFoundCompStyle = False
    FMT_ColorCodeOneRangeStringBasic(oCurs, sTokens(), sCharStyles())
End If
End Sub

```

Color code selected text only. As of version 2.1.0, selected text is handled properly – prior to this, an entire paragraph needed to be formatted.

Listing 9.19: Highlight selected macros

```

REM Format just the selected text
Sub HighlightSelBasic()
    Dim oSels
    Dim oSel
    Dim i%
    Dim sTokens()
    Dim sCharStyles() : sCharStyles() = FMT_GetBasicCharacterStyles()

    FMT_InitSpecialCharArrays()
    FMT_InitTokensBasic(sTokens())
    oSels = ThisComponent.getCurrentController().getSelection()
    For i = 0 To oSels.getCount() - 1
        oSel = oSels.getByIndex(i)
        FMT_ColorCodeOneRangeStringBasic(oSel, sTokens(), sCharStyles())
    Next
End Sub

```

9.4.2. The worker macro

Most of the work is done in one macro, which color codes the text selected by a text cursor. All of the selected text is assumed to be macro code so paragraph styles are not checked.

Listing 9.20: Highlight text selected by a text cursor

```

' *****
' ** Color code the Basic code in the oSel range.
' ** Use the keywords in the sTokens() array.
' *****
Sub FMT_ColorCodeOneRangeStringBasic(oSel, sTokens(), sCharStyles())
    Dim oCurs 'Iterate paragraphs in the selected region.

```

```

Dim oTCurs      'Iterate the characters in a paragraph.
Dim oText       'Text object containing the selection.
Dim iPos%
Dim iLen%
Dim i%          'Temporary integer variable.
Dim sChar$      'Current character
Dim sLine$      'Current line (in lower case).

REM Position oTCurs at the start of the selection.
oText = oSel.GetText()
oTCurs = oText.CreateTextCursorByRange(oSel.getStart())
oTCurs.goRight(0, False)

REM oCurs contains the first paragraph, but not more than oSel.
oCurs = FMT_CreateStartCursor(oText, oSel)

REM Verify that oCurs ends before (or at the same place)
REM as oSel.
Do While oText.compareRegionEnds(oCurs, oSel) >= 0
    REM Now, process a single line of text!
    REM oCurs has selected the entire paragraph.
    REM oTCurs is at the start of the paragraph.
    sLine = LCase(oCurs.getString())
    iLen = Len(sLine)
    iPos = 1
    Do While iPos <= iLen
        REM Skip leading white space.
        FMT_FindNextNonSpace(sLine, iPos%, iLen%)
        If iPos > iLen Then Exit Do

        sChar = Mid(sLine, iPos, 1)
        If sChar = "'" Then
            Rem Found a comment, mark the rest of the line.
            REM Move the character cursor from the paragraph start
            REM to the single quote character.
            REM Select the rest of the paragraph.
            oTCurs.goRight(iPos-1, False)
            oTCurs.gotoEndOfParagraph(True)
            oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Comment)
            iPos = iLen + 1
        ElseIf sChar = """" Then
            REM Move to the first double quote
            oTCurs.goRight(iPos-1, False)

            REM Remember the location of the first double quote
            REM and then find then end of the quoted text.
            i = iPos

```

```

FMT_FindEndQuoteDouble(sLine$, iPos%, iLen%)

REM Move the cursor to the closing double quote.
REM Set the character style for the string.
REM Move the cursor back to the start of the paragraph.
oTCurs.goRight(iPos - i, True)
oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Literal)
oTCurs.gotoRange(oCurs.start, False)
ElseIf FMT_FindNumberEnd(sLine, iPos, iLen, i) Then
    REM Move to the number start.
    oTCurs.goRight(iPos-1, False)
    oTCurs.goRight(i - iPos, True)
    oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Literal)
    oTCurs.gotoRange(oCurs.start, False)
    iPos = i
ElseIf sChar = "." OR FMT_IsSpecialChar(ASC(sChar)) Then
    i = iPos
    oTCurs.goRight(iPos - 1, False)

Do
    iPos = iPos + 1
    If iPos > iLen Then Exit Do
Loop Until NOT FMT_IsSpecialChar(ASC(Mid(sLine, iPos, 1)))
oTCurs.goRight(iPos - i, True)
oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Special)
oTCurs.gotoRange(oCurs.start, False)
ElseIf sChar = "_" AND iPos = iLen Then
    REM An Identifier can start with an "_" (I think).
    REM It is likely that trailing spaces will be in a text
    REM document, but we will ignore these for now!
    oTCurs.goRight(iPos-1, False)
    oTCurs.goRight(1, True)
    oTCurs.CharStyleName = sCharStyles(FMT_CSNI_KeyWord)
    oTCurs.gotoRange(oCurs.start, False)
    iPos = iPos + 1
Else
    REM No special characters, so this is a variable
    REM or logic statement. Move to the first character.
    i = iPos
    oTCurs.goRight(iPos-1, False)
Do
    iPos = iPos + 1
    If iPos > iLen Then Exit Do
Loop Until FMT_IsWordSep(Asc(Mid(sLine, iPos, 1)))

oTCurs.goRight(iPos - i, True)
sChar = LCase(oTCurs.getString())

```

```

REM This could be a problem for a variable named
REM "rem.doit.var". The Basic IDE misses this as well
REM so I am not concerned.
If sChar = "rem" Then
    oTCurs.gotoEndOfParagraph(True)
    oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Comment)
    iPos = iLen + 1
ElseIf FMT_ArrayHasString(sChar, sTokens()) Then
    oTCurs.CharStyleName = sCharStyles(FMT_CSNI_KeyWord)
Else
    oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Ident)
End If

    oTCurs.gotoRange(oCurs.start, False)
End If
Loop
If Not oCurs.gotoNextParagraph(False) Then Exit Do
oTCurs.gotoRange(oCurs, False)
If NOT FMT_CursorToParEnd(oText, oSel, oCurs) Then Exit Do
Loop
End Sub

```

9.5. Java Module

I created macros specifically to format Java. Java has a color scheme different from Basic. Although I have formatted numerous code listings in Basic, I have not formatted a significant amount of Java code. The order of the character styles is the same in Java as in Basic.

Listing 9.21: Highlight Java code

```

Sub FMT_ColorCodeCurrentJava()
    Dim oVCurs
    Dim oCurs
    Dim sStyles()      ' Paragraph styles for code listings.
    Dim sTokens()      ' Keyword tokens.

    FMT_InitSpecialCharArrays()
    FMT_InitParStyles(sStyles())
    FMT_InitTokensJava(sTokens())

    REM Get the view cursor as the starting location
    oVCurs = ThisComponent.getCurrentController().getViewCursor()
    oCurs = oVCurs.getText().createTextCursorByRange(oVCurs)
    If NOT FMT_ArrayHasString(oCurs.ParaStyleName, sStyles()) Then
        MsgBox "No code found" & CHR$(10) & _
            "You must use a 'CODE' paragraph style" & CHR$(10) & _
            "AndrewMacro.odt uses these styles" & CHR$(10) & _

```

```

        "-----" & CHR$(10) & _
        Join(sStyles(), CHR$(10))
    Exit sub
End If
REM Select the set of paragraphs that define the code range.
FMT_FindCodeAroundCursor(oCurs, sStyles())

REM Now format it!
FMT_ColorCodeOneRangeStringJava(oCurs, sTokens(),
FMT_GetJavaCharacterStyles())
End Sub

REM Format just the selected text
Sub HighlightSelJava()
    Dim oSels
    Dim oSel
    Dim i%
    Dim sTokens() ' Keyword tokens.
    Dim sCharStyles() : sCharStyles() = FMT_GetJavaCharacterStyles()

    FMT_InitSpecialCharArrays()
    FMT_InitTokensJava(sTokens())

    oSels = ThisComponent.getCurrentController().getSelection()
    For i = 0 To oSels.getCount() - 1
        oSel = oSels.getByIndex(i)
        FMT_ColorCodeOneRangeStringJava(oSel, sTokens(), sCharStyles())
    Next
End Sub

' *****
' ** Very simple parsing of Java code.
' *****
Sub FMT_ColorCodeOneRangeStringJava(oSel, sTokens(), sCharStyles())
    Dim oCurs 'Iterate paragraphs in the selected region.
    Dim oTCurs 'Iterate the characters in a paragraph.
    Dim oText 'Text object containing the selection.
    Dim iPos%
    Dim iLen%
    Dim i% 'Temporary integer variable.
    Dim sChar$ 'Current character
    Dim sLine$ 'Current line (in lower case).
    Dim bComment As Boolean
    Dim bIsAsterick As Boolean

    REM We are not currently processing a comment.
    bComment = False

```

```

REM Position oTCurs at the start of the selection.
oText = oSel.getText()
oTCurs = oText.createTextCursorByRange(oSel.getStart())
oTCurs.goRight(0, False)

REM oCurs contains the first paragraph.
oCurs = FMT_CreateStartCursor(oText, oSel)

Do While oText.compareRegionEnds(oCurs, oSel) >= 0
    REM Now, process a single line of text!
    REM oCurs has selected the entire paragraph.
    REM oTCurs is at the start of the paragraph.
    sLine = LCase(oCurs.getString())
    iLen = Len(sLine)
    iPos = 1
    Do While iPos <= iLen
        REM Skip leading white space.
        FMT_FindNextNonSpace(sLine, iPos%, iLen%)
        If iPos > iLen Then Exit Do
        sChar = Mid(sLine, iPos, 1)

        REM Is the cursor in a multi-line comment?
        If bComment Then
            i = iPos
            Do while iPos <= iLen
                REM Skip NON '*' characters
                Do
                    If Mid(sLine, iPos, 1) <> "*" Then Exit do
                    iPos = iPos + 1
                Loop Until iPos > iLen

                REM Check for "*"
                bIsAsterick = False
                If iPos <= iLen Then
                    Do While Mid(sLine, iPos, 1) = "*"
                        bIsAsterick = True
                        iPos = iPos + 1
                    If iPos > iLen Then Exit Do
                Loop
            End If

            REM Check for trailing "/"
            If iPos <= iLen Then
                iPos = iPos + 1
                If Mid(sLine, iPos-1, 1) = "/" Then
                    REM Found the end of the comment
                    bComment = False
                End If
            End If
        End If
    Loop
End Do

```



```

        Exit Do
    End If
End if
Loop
oTCurs.goRight(i-1, False)
oTCurs.goRight(iPos - i, True)
oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Comment)
oTCurs.gotoStartOfParagraph(False)
ElseIf sChar = "/" AND iPos < iLen Then
    Rem Might be a comment.
    If Mid(sLine, iPos+1, 1) = "*" Then
        REM This starts a multi-line comment.
        REM The fastest way to find the end comment is with the
        REM built in searching capability. Unfortunately, I can
        REM not then manually set iPos so I will not do this.
        bComment = True
        oTCurs.goRight(iPos-1, False)
        oTCurs.goRight(2, True)
        oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Comment)
        iPos = iPos + 2
        oTCurs.gotoStartOfParagraph(False)
    ElseIf Mid(sLine, iPos+1, 1) = "/" Then
        REM This starts a single line comment.
        oTCurs.goRight(iPos-1, False)
        oTCurs.gotoEndOfParagraph(True)
        oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Comment)
        iPos = iLen + 1
    Else
        REM This is not a comment.
        oTCurs.goRight(iPos-1, False)
        oTCurs.gotoEndOfParagraph(True)
        'oTCurs.CharStyleName = sCharStyles(FMT_CSNI_KeyWord)
        oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Ident)
        iPos = iPos + 1
    End If
' ElseIf sChar = '"' Then
'     Rem Found a comment, mark the rest of the line.
'     REM Move the character cursor from the paragraph start
'     REM to the single quote character.
'     REM Select the rest of the paragraph.
'     oTCurs.goRight(iPos-1, False)
'     oTCurs.gotoEndOfParagraph(True)
'     oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Comment)
'     iPos = iLen + 1
ElseIf sChar = "\"" OR sChar = "'" Then
    REM Move to the first double quote
    oTCurs.goRight(iPos-1, False)

```

```

REM Remember the location of the first double quote
REM and then find then end of the quoted text.
i = iPos
FMT_FindEndQuoteEscape(sLine$, iPos%, iLen%)

REM Move the cursor to the closing double quote.
REM Set the character style for the string.
REM Move the cursor back to the start of the paragraph.
oTCurs.goRight(iPos - i, True)
oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Literal)
oTCurs.gotoRange(oCurs.start, False)
ElseIf FMT_FindNumberEnd(sLine, iPos, iLen, i) Then
    REM Move to the number start.
    oTCurs.goRight(iPos-1, False)
    oTCurs.goRight(i - iPos, True)
    oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Literal)
    oTCurs.gotoRange(oCurs.start, False)
    iPos = i
ElseIf sChar = "." OR FMT_IsSpecialChar(ASC(sChar)) Then
    i = iPos
    oTCurs.goRight(iPos - 1, False)

Do
    iPos = iPos + 1
    If iPos > iLen Then Exit Do
Loop Until NOT FMT_IsSpecialChar(ASC(Mid(sLine, iPos, 1)))
oTCurs.goRight(iPos - i, True)
oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Special)
oTCurs.gotoRange(oCurs.start, False)
Else
    REM No special characters, so this is a variable
    REM or logic statement. Move to the first character.
    i = iPos
    oTCurs.goRight(iPos-1, False)
Do
    iPos = iPos + 1
    If iPos > iLen Then
        Exit Do
    End If
Loop Until FMT_IsWordSep(Asc(Mid(sLine, iPos, 1)))

oTCurs.goRight(iPos - i, True)
sChar = LCase(oTCurs.getString())

If FMT_ArrayHasString(sChar, sTokens()) Then
    oTCurs.CharStyleName = sCharStyles(FMT_CSNI_KeyWord)

```

```

Else
    oTCurs.CharStyleName = sCharStyles(FMT_CSNI_Ident)
End If

    oTCurs.gotoRange(oCurs.start, False)
End If
Loop
If Not oCurs.gotoNextParagraph(False) Then Exit Do
oTCurs.gotoRange(oCurs, False)
If NOT FMT_CursorToParEnd(oText, oSel, oCurs) Then Exit Do
Loop
End Sub

'*****
'** Tokens recognized by java.
'*****
Sub FMT_InitTokensJava(sTokens())
    sTokens() = Array("abstract", "assert", _
        "boolean", "break", "byte", _
        "case", "catch", "char", "class", "const", "continue", _
        "default", "do", "double", _
        "else", "enum", "extends", _
        "final", "finally", "float", "for", _
        "goto", _
        "if", "implements", "import", "instanceof", "int", "interface", _
        "long", _
        "native", "new", _
        "package", "private", "protected", "public", _
        "return", _
        "short", "static", "strictfp", "super", "switch", "synchronized", _
        "this", "throw", "throws", "transient", "try", _
        "void", "volatile", _
        "while")
    FMT_SortStringArrayAscending(sTokens())
End Sub

'*****
'** Get the character styles meant for highlighting Java code.
'** Java formats character styles as an identifier.
'*****
Function FMT_GetJavaCharacterStyles()
    CreateJavaCharStyles()
    FMT_GetJavaCharacterStyles() = Array( "_JavaComment", _
        "_JavaLiteral", "_JavaKeyWord", _
        "_JavaIdent", "_JavaIdent")
End Function

```

Below is an example of formatted Java code.

Listing 9.22: Example highlighted Java code

```
static public String byteToHex(byte b)
{
    // Returns hex String representation of byte b
    char hexDigit[] = {
        '0', '1', '2', '3', '4', '5', '6', '7',
        '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'
    };
    char[] array = { hexDigit[(b >> 4) & 0x0f], hexDigit[b & 0x0f] };
    return new String(array);
}

static public String charToHex(char c)
{
    // Returns hex String representation of char c
    byte hi = (byte) (c >>> 8);
    byte lo = (byte) (c & 0xff);
    return byteToHex(hi) + byteToHex(lo);
}

static public String intToHex(int n)
{
    String s = "";
    for (int i=0; i<8; ++i)
    {
        s = byteToHex((byte) (n & 0xff)) + s;
        n = n >>> 8;
        if (n == 0)
        {
            return s;
        }
    }
    return s;
}
```

9.6. Cpp Module

C++ is sufficiently close to Java, that I used the Java parser to parse the C++ code. This code is very new and is likely to change as problems are found. Be certain to tell me of any issues that you find.

Listing 9.23: C++ character styles.

```
Sub FMT_ColorCodeCurrentCpp()
    Dim oVCurs
    Dim oCurs
    Dim sStyles()      ' Paragraph styles for code listings.
    Dim sTokens()      ' Keyword tokens.
```

```

    Dim sCharStyles() : sCharStyles() = FMT_GetCppCharacterStyles()

    FMT_InitSpecialCharArrays()
    FMT_InitParStyles(sStyles())
    FMT_InitTokensCpp(sTokens())

    REM Get the view cursor as the starting location
    oVCurs = ThisComponent.getCurrentController().getViewCursor()
    oCurs = oVCurs.getText().createTextCursorByRange(oVCurs)
    If NOT FMT_ArrayHasString(oCurs.ParaStyleName, sStyles()) Then
        MsgBox "No code found" & CHR$(10) & _
            "You must use a 'CODE' paragraph style" & CHR$(10) & _
            "AndrewMacro.odt uses these styles" & CHR$(10) & _
            "-----" & CHR$(10) & _
            Join(sStyles(), CHR$(10))
        Exit sub
    End If

    REM Select the set of paragraphs that define the code range.
    FMT_FindCodeAroundCursor(oCurs, sStyles())

    REM Now format it!
    FMT_ColorCodeOneRangeStringCpp(oCurs, sTokens(), sCharStyles())
End Sub

REM Format just the selected text
Sub HighlightSelCpp()
    Dim oSels
    Dim oSel
    Dim i%
    Dim sTokens() ' Keyword tokens.
    Dim sCharStyles() : sCharStyles() = FMT_GetCppCharacterStyles()

    FMT_InitSpecialCharArrays()
    FMT_InitTokensCpp(sTokens())

    oSels = ThisComponent.getCurrentController().getSelection()
    For i = 0 To oSels.getCount() - 1
        oSel = oSels.getByIndex(i)
        FMT_ColorCodeOneRangeStringCpp(oSel, sTokens(), sCharStyles())
    Next
End Sub

' *****
' ** The Java parser is used for the C++ code
' *****
Sub FMT_ColorCodeOneRangeStringCpp(oSel, sTokens(), sCharStyles())
    FMT_ColorCodeOneRangeStringJava(oSel, sTokens(), sCharStyles())

```

```

End Sub

'*****
'*** Tokens recognized by c++
'*****

Sub FMT_InitTokensCpp(sTokens())
    sTokens() = Array("asm", "auto", "bool", "break", _
        "case", "catch", "char", "class", "concept", "const", "constexpr",
        "const_cast", "continue", _
        "default", "define", "defined", "delete", _
        "do", "double", "dynamic_cast", _
        "else", "endif", "enum", "explicit", "export", "extern", _
        "false", "float", "for", "friend", "goto", _
        "if", "ifndef", "include", "inline", "int", "long", _
        "mutable", "namespace", "new", "operator", _
        "private", "protected", "public", _
        "register", "reinterpret_cast", "requires", "return", _
        "short", "signed", "sizeof", "static", "static_cast", _
        "struct", "switch", _
        "template", "this", "thread_local", "throw", "true", "try", "typedef", _
        "typeid", "typename", _
        "union", "unsigned", "using", _
        "virtual", "void", "volatile", _
        "wchar_t", "while")
    FMT_SortStringArrayAscending(sTokens())
End Sub

'*****
'*** Get the character styles meant for highlighting Cpp code.
'*** Format special characters with the Base color.
'*****

Function FMT_GetCppCharacterStyles()
    CreateStarBasicCharStyles()
    FMT_GetCppCharacterStyles() = Array( "_OOoComputerComment", _
        "_OOoComputerLiteral", "_OOoComputerKeyWord", _
        "_OOoComputerIdent", "_OOoComputerBase")
End Function

```

Here is some sample C++ code.

```

#include "NestedException.h"
#include <iostream>
#include <iomanip>
#include <boost/filesystem/operations.hpp>
#include <boost/filesystem/convenience.hpp>
#include <boost/filesystem/path.hpp>
namespace fs = boost::filesystem;

//! Main program entry point.

```

```

int main(int argc, char** argv)
{
    std::string platform( BOOST_PLATFORM );
    std::cout << "Initial API is " << platform << '\n';

    # if defined( BOOST_POSIX_API )
        platform = "POSIX";
    # elif defined( BOOST_WINDOWS_API )
        platform = "Windows";
    # else
        platform = ( platform == "Win32" || platform == "Win64"
                    || platform == "Cygwin" )
                    ? "Windows"
                    : "POSIX";
    # endif
    std::cout << "Final API is " << platform << '\n';

    for (int i=1; i<argc; ++i)
    {
        fs::path filePath(argv[i]);
        std::cout << "Path = " << filePath.string() << std::endl;
    }
    return 0;
}

```

9.7. XML Module

More character attributes are used to format XML than the other languages. The XML code has a distinct parser, that uses states; the general format for the code is the same.

Listing 9.24: Format XML.

```

Private const FMT_XML_Attribute = 0
Private const FMT_XML_Comment   = 1
Private const FMT_XML_Content   = 2
Private const FMT_XML_Element   = 3
Private const FMT_XML_Entity     = 4
Private const FMT_XML_KeyWord    = 5
Private const FMT_XML_Literal    = 6
Private const FMT_XML_Equals     = 7
Private const FMT_XML_Brackets   = 7

REM State values
Private const XML_ST_PorcInst    = 1  'Processing instruction ?
Private const XML_ST_Prolog      = 2  'Non-Comment prolog element !
Private const XML_ST_FndElem     = 3  'Find an element
Private const XML_ST_FndAttr     = 4  'Find an attribute
Private const XML_ST_FndAttrEq   = 5  'Find = after attribute
Private const XML_ST_FndAttrVal  = 6  'Find attribute value

```

```

Private const XML_ST_FndQuote    = 7
Private const XML_ST_LT         = 8  'Found a <
Private const XML_ST_CloseElem  =11  'Close an element /
Private const XML_ST_InElem     =12  'In an element

Sub Main
    FMT_ColorCodeCurrentXML()
End Sub

Sub FMT_ColorCodeCurrentXML()
    Dim oVCurs
    Dim oCurs
    Dim sStyles()      ' Paragraph styles for code listings.
    Dim sTokens()      ' Keyword tokens.
    Dim sCharStyles() : sCharStyles() = FMT_GetXMLCharacterStyles()

    FMT_InitSpecialCharArrays()
    FMT_InitParStyles(sStyles())
    FMT_InitTokensXML(sTokens())

    REM Get the view cursor as the starting location
    oVCurs = ThisComponent.getCurrentController().getViewCursor()
    oCurs  = oVCurs.getText().createTextCursorByRange(oVCurs)
    If NOT FMT_ArrayHasString(oCurs.ParaStyleName, sStyles()) Then
        MsgBox "No XML found" & CHR$(10) & _
            "You must use a 'CODE' paragraph style" & CHR$(10) & _
            "AndrewMacro.odt uses these styles" & CHR$(10) & _
            "-----" & CHR$(10) & _
            Join(sStyles(), CHR$(10))
        Exit sub
    End If
    REM Select the set of paragraphs that define the code range.
    FMT_FindCodeAroundCursor(oCurs, sStyles())

    REM Now format it!
    FMT_ColorCodeOneRangeStringXML(oCurs, sTokens(), sCharStyles())
End Sub

REM Format just the selected text
Sub HighlightSelXML()
    Dim oSels
    Dim oSel
    Dim i%
    Dim sTokens()
    Dim sCharStyles() : sCharStyles() = FMT_GetXMLCharacterStyles()

```



```

FMT_InitSpecialCharArrays()
FMT_InitParStyles(sStyles())
FMT_InitTokensXML(sTokens())

oSels = ThisComponent.getCurrentController().getSelection()
For i = 0 To oSels.getCount() - 1
    oSel = oSels.getByIndex(i)
    FMT_ColorCodeOneRangeStringBasic(oSel, sTokens(), sCharStyles())
Next
End Sub

' *****
' ** Very simple parsing of XML code.
' *****
Sub FMT_ColorCodeOneRangeStringXML(oSel, sTokens(), sCharStyles())
    Dim oCurs      'Iterate paragraphs in the selected region.
    Dim oTCurs     'Iterate the characters in a paragraph.
    Dim oText      'Text object containing the selection.
    Dim iPos%      'Position in the current line.
    Dim iLen%      'Length of the current line.
    Dim i%         'Temporary integer variable.
    Dim sChar$     'Current character
    Dim sLine$     'Current line (in lower case).
    Dim iComment%  'Track nested comments (could use the stack)
    Dim sQuote$    'Which quote character was found?
    Dim iTmp1%
    Dim iTmp2%
    Dim iTmp3%

    Dim iStack (0 To 200) As Integer
    Dim iStackSize%

    iComment = 0      ' Level in a comment
    iStackSize = 0

    REM Position oTCurs at the start of the selection.
    oText = oSel.getText()
    oTCurs = oText.createTextCursorByRange(oSel.getStart())
    oTCurs.goRight(0, False)

    REM oCurs contains the first paragraph.
    oCurs = FMT_CreateStartCursor(oText, oSel)

    Do While oText.compareRegionEnds(oCurs, oSel) >= 0
        REM Now, process a single line of text!
        REM oCurs has selected the entire paragraph.
        REM oTCurs is at the start of the paragraph.

```

```

sLine = LCase(oCurs.getString())
iLen = Len(sLine)
iPos = 1
Do While iPos <= iLen
    REM Skip leading white space.
    FMT_FindNextNonSpace(sLine, iPos%, iLen%)
    If iPos > iLen Then Exit Do

    ' Does this line start a comment?
    If InStr(iPos, sLine, "<!--") = iPos Then
        iComment = iComment + 1
        FMT_SetStyleForChars(oTCurs, iPos, 4, FMT_XML_Comment, sCharStyles())
        iPos = iPos + 4
    End If

    If iPos > iLen Then Exit Do
    ' Remove next line
    sChar = Mid(sLine, iPos, 1)
    'DumpStack(iStack%(), iStackSize%, sLine$, iPos%)
    ' Handle comments!
    If iComment > 0 Then
        i = iPos
        Do While iPos <= iLen AND iComment > 0
            iTmp1 = InStr(iPos, sLine, "<!--")
            iTmp2 = InStr(iPos, sLine, "-->")
            If iTmp1 > 0 AND iTmp1 < iTmp2 Then
                ' There is a start before the end.
                iPos = iTmp1 + 4
                iComment = iComment + 1
            ElseIf iTmp2 > 0 Then
                ' There is an end before a start
                iPos = iTmp2 + 4
                iComment = iComment - 1
            Else
                ' No start
                iPos = iLen + 1
            End If
        Loop
        FMT_SetStyleForChars(oTCurs, i, iPos-i, FMT_XML_Comment, sCharStyles())
    ElseIf InStr(iPos, sLine, "?>") = iPos Then
        FMT_SetStyleForChars(oTCurs, iPos-1, 2, FMT_XML_Brackets, sCharStyles())
        iPos = iPos + 2
        FMT_PopStackUntil(iStack(), iStackSize, XML_ST_PorcInst)
    ElseIf InStr(iPos, sLine, "</") = iPos Then
        FMT_SetStyleForChars(oTCurs, iPos-1, 2, FMT_XML_Brackets, sCharStyles())
        iPos = iPos + 2
        FMT_PopStackUntil(iStack(), iStackSize, XML_ST_InElem)
    End If

```

```

    FMT_PushStack(iStack(), iStackSize, XML_ST_CloseElem)
    FMT_PushStack(iStack(), iStackSize, XML_ST_FndElem)
ElseIf sChar = "<" Then
    REM This can be so many different things.
    FMT_SetStyleForChars(oTCurs, iPos-1, 1, FMT_XML_Brackets, sCharStyles())
    FMT_PushStack(iStack(), iStackSize, XML_ST_LT)
    iPos = iPos + 1
    If iPos > iLen Then
        ' Starts an element, and the < ends a line.
        FMT_PushStack(iStack(), iStackSize, XML_ST_FndElem)
        oTCurs.gotoEndOfParagraph(False)
        Exit Do
    End If

    sChar = Mid(sLine, iPos, 1)
    If sChar = "?" Then
        ' Processing Instruction
        FMT_SetStyleForChars(oTCurs, iPos-1, 1, FMT_XML_Brackets,
sCharStyles())
        iPos = iPos + 1
        iStack%(iStackSize - 1) = XML_ST_PorcInst
        FMT_PushStack(iStack(), iStackSize, XML_ST_FndElem)
    ElseIf sChar = "!" Then
        ' This is not a comment, because comments are
        ' detected elsewhere
        FMT_SetStyleForChars(oTCurs, iPos-1, 1, FMT_XML_Brackets,
sCharStyles())
        iStack%(iStackSize - 1) = XML_ST_Prolog
        FMT_PushStack(iStack(), iStackSize, XML_ST_FndElem)
        iPos = iPos + 1
    Else
        FMT_PushStack(iStack(), iStackSize, XML_ST_FndElem)
    End If
ElseIf sChar = ">" Then
    FMT_SetStyleForChars(oTCurs, iPos-1, 1, FMT_XML_Brackets, sCharStyles())
    iPos = iPos + 1
    Do While iStackSize > 0
        iTmpl = FMT_PeekStack(iStack(), iStackSize)
        If iTmpl = XML_ST_PorcInst OR _
            iTmpl = XML_ST_Prolog OR _
            iTmpl = XML_ST_CloseElem Then
            FMT_PopStack(iStack(), iStackSize)
            Exit Do
        ElseIf iTmpl = XML_ST_LT Then
            ' Went from in an element such as '<x a="blh" ' to '<x>I am here'
            iStack(iStackSize - 1) = XML_ST_InElem
            Exit Do
        Else

```

```

        FMT_PopStack(iStack%(), iStackSize)
    End If
Loop
ElseIf sChar = "/" Then
    ' Not in a comment, and not at an opening element.
    FMT_SetStyleForChars(oTCurs, iPos-1, 1, FMT_XML_Brackets, sCharStyles())
    Do While iStackSize > 0
        iTmpl = FMT_PopStack(iStack(), iStackSize)
        If iTmpl = XML_ST_LT OR iTmpl = XML_ST_InElem Then
            Exit Do
        End If
    Loop
    FMT_PushStack(iStack%(), iStackSize, XML_ST_CloseElem)
    iPos = iPos + 1
Else
    iTmpl = FMT_PeekStack(iStack%(), iStackSize)
    If iTmpl = XML_ST_FndElem or iTmpl = XML_ST_FndAttr Then
        If Mid(sLine, iPos, 1) = "]" OR Mid(sLine, iPos, 1) = "[" Then
            FMT_SetStyleForChars(oTCurs, iPos-1, 1, FMT_XML_Brackets,
sCharStyles())
            iPos = iPos + 1
        Else
            iTmp2 = FMT_XML_ElementEnd(sLine, iPos, iLen)
            If iTmp2 >= iPos Then
                iTmp3 = iTmp2 - iPos + 1
                If FMT_ArrayHasString(Mid(sLine, iPos, iTmp3), sTokens()) Then
                    FMT_SetStyleForChars(oTCurs, iPos-1, iTmp3, FMT_XML_KeyWord,
sCharStyles())
                Else
                    FMT_SetStyleForChars(oTCurs, iPos-1, iTmp3, FMT_XML_Element,
sCharStyles())
                End If
                iPos = iTmp2 + 1
            Else
                ' This is an error
                iPos = iPos + 1
            End If
            iStack(iStackSize - 1) = iTmpl + 1
        End If
    ElseIf iTmpl = XML_ST_FndAttrEq Then
        ' Looking for the attributes "=" sign.
        If Mid(sLine, iPos, 1) = "=" Then
            FMT_SetStyleForChars(oTCurs, iPos-1, 1, FMT_XML_Equals,
sCharStyles())
            iStack(iStackSize - 1) = XML_ST_FndAttrVal
            iPos = iPos + 1
        ElseIf Mid(sLine, iPos, 1) = "\"" OR Mid(sLine, iPos, 1) = "'" Then
            'This could be an error,

```

```

        'but I am not tracking well enough to know.
        iStack(iStackSize - 1) = XML_ST_FndAttrVal
    Else
        iStack(iStackSize - 1) = XML_ST_FndAttr
    End if
    ElseIf iTmp1 = XML_ST_FndAttrVal Then
        ' Looking for the attributes.
        sQuote = Mid(sLine, iPos, 1)
        If sQuote = "'" OR sQuote = "\"" Then
            FMT_SetStyleForChars(oTCurs, iPos-1, 1, FMT_XML_Literal,
sCharStyles())
            iStack(iStackSize - 1) = XML_ST_FndQuote
            iPos = iPos + 1
        Else
            iTmp2 = FMT_XMLElmentEnd(sLine, iPos, iLen)
            If iTmp2 >= iPos Then
                iTmp3 = iTmp2 - iPos + 1
                FMT_SetStyleForChars(oTCurs, iPos-1, iTmp3, FMT_XML_Literal,
sCharStyles())
                iPos = iTmp2 + 1
            Else
                ' This is an error
                iPos = iPos + 1
            End If
            iStack(iStackSize - 1) = XML_ST_FndAttr
        End if
    ElseIf iTmp1 = XML_ST_FndQuote Then
        iTmp3 = iPos
        Do While iPos <= iLen
            sChar = Mid(sLine, iPos, 1)
            iPos = iPos + 1
            If sChar = sQuote OR sChar = "&" Then
                Exit Do
            End If
        Loop
        FMT_SetStyleForChars(oTCurs, iTmp3 - 1, iPos-iTmp3, FMT_XML_Literal,
sCharStyles())
        If sChar = sQuote Then
            iStack(iStackSize - 1) = XML_ST_FndAttr
        ElseIf sChar = "&" Then
            iPos = iPos - 1
            iTmp2 = InStr(iPos, sLine, ";")
            If iTmp2 < 0 Then iTmp2 = iLen - 1
            If iTmp2 > iPos Then
                FMT_SetStyleForChars(oTCurs, iPos - 1, iTmp2-iPos+1,
FMT_XML_Entity, sCharStyles())
                iPos = iTmp2 + 1
            Else

```

```

        iPos = iPos + 1
    End If
End If
ElseIf iTmp1 = XML_ST_InElem Then
    ' Inside an XML statement, but past the white space
    ' Only two things can get us out; an entity, or <.
    iTmp2 = InStr(iPos, sLine, "<")
    iTmp3 = InStr(iPos, sLine, "&")

    If iTmp3 < iTmp2 AND iTmp3 > 0 Then
        iTmp2 = iTmp3
    ElseIf iTmp2 < 1 Then
        iTmp2 = iTmp3
    End If
    If iTmp2 > 0 Then
        FMT_SetStyleForChars(oTCurs, iPos-1, iTmp2-iPos, FMT_XML_Content,
sCharStyles())
        iPos = iTmp2
    Else
        'I should never get here
        iPos = iPos + 1
    End If
Else
    'What are we looking for?
    'Increment to avoid an infinite loop on error.
    iPos = iPos + 1
End If
End If
oTCurs.gotoRange(oCurs.start, False)
Loop
If Not oCurs.gotoNextParagraph(False) Then Exit Do
oTCurs.gotoRange(oCurs, False)
If NOT FMT_CursorToParEnd(oText, oSel, oCurs) Then Exit Do
Loop
End Sub

'*****
'** I use this for debugging
'*****
Sub DumpStack(iStack%(), iStackSize%, sLine$, iPos%)
    Dim s$
    Dim i%
    s = "Pos " & iPos
    s = s & " (" & Mid(sLine, iPos, 1) & ") "
    s = s & " for " & sLine & CHR$(10)
    For i = LBound(iStack()) To iStackSize - 1
        s = s & iStack(i) & " " & StateToString(iStack(i)) & CHR$(10)
    Next

```

```

    MsgBox s
End Sub

'*****
'*** I use this for debugging
'*****

Function StateToString(iState%) As String
    Select Case iState
        Case XML_ST_PorcInst
            StateToString = "XML_ST_PorcInst"
        Case XML_ST_Prolog
            StateToString = "XML_ST_Prolog"
        Case XML_ST_FndElem
            StateToString = "XML_ST_FndElem"
        Case XML_ST_FndAttr
            StateToString = "XML_ST_FndAttr"
        Case XML_ST_FndAttrEq
            StateToString = "XML_ST_FndAttrEq"
        Case XML_ST_FndAttrVal
            StateToString = "XML_ST_FndAttrVal"
        Case XML_ST_FndQuote
            StateToString = "XML_ST_FndQuote"
        Case XML_ST_LT
            StateToString = "XML_ST_LT"
        Case XML_ST_CloseElem
            StateToString = "XML_ST_CloseElem"
        Case XML_ST_InElem
            StateToString = "XML_ST_InElem"
        Case Else
            StateToString = "Invalid"
    End Select
End Function

'*****
'*** Tokens recognized by XML
'*****

Sub FMT_InitTokensXML(sTokens())
    sTokens() = Array( "#fixed", "#implied", _
        "#pcdata", "#required", _
        "any", "attlist", "cdata", "charset", _
        "default", "doctype", "element", _
        "empty", "encoding", "entities", "entity", _
        "euc-jp", "euc-kr", "href", _
        "id", "idref", "idrefs", "ignore", "include", _
        "iso-10646-ucs-2", "iso-10646-ucs-4", "iso-2022-jp", _
        "iso-8859-1", "iso-8859-2", _

```

```

"media", "ndata", "nmtoken", "nmtokens", "notation", _
"preserve", "public", _
"shift_jis", "standalone", "system", "title", "type", _
"utf-16", "utf-8", "utf-8", "version", _
"xml", "xml-styleSheet", "xml:lang", "xml:space", _
"xsd:anyuri", "xsd:base64binary", "xsd:boolean", "xsd:byte", _
"xsd:date", "xsd:datetime", "xsd:decimal", "xsd:double", _
"xsd:float", _
"xsd:gday", "xsd:gmonth", "xsd:gmonthday", "xsd:gyear", _
"xsd:gyearmonth", "xsd:hexbinary", _
"xsd:int", "xsd:integer", "xsd:language", "xsd:long", _
"xsd:name", "xsd:ncname", "xsd:negativeinteger", "xsd:nmtoken", _
"xsd:nonnegativeinteger", "xsd:nonpositiveinteger", _
"xsd:normalizedstring", "xsd:positiveinteger", _
"xsd:short", "xsd:string", "xsd:time", "xsd:token", _
"xsd:unsignedbyte", "xsd:unsignedint", _
"xsd:unsignedlong", "xsd:unsignedshort" )
FMT_SortStringArrayAscending(sTokens())
End Sub

'*****
'*** Get the character styles meant for highlighting XML code.
'*** Format special characters with the Base color.
'*****
Function FMT_GetXMLCharacterStyles()
    CreateXMLCharStyles()
    FMT_GetXMLCharacterStyles() = Array( "_XMLAttribute", _
        "_XMLComment", "_XMLContent", "_XMLElement", _
        "_XMLEntity", "_XMLKeyWord", "_XMLLiteral", "_XMLBracket" )
End Function

```

Consider the following XML sample.

```

<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >
    <owl:Ontology rdf:about="">
        <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >Represent Countries.</rdfs:comment>
    </owl:Ontology>
    <owl:Class rdf:ID="Country">
        <rdfs:subClassOf>
            <owl:Class rdf:ID="Political_Group"/>
        </rdfs:subClassOf>
    </owl:Class>

```



```
<owl:Class rdf:about="#Political_Group">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >A country is roughly considered a political group.
    Political groups allow grouping such as NATO.</rdfs:comment>
</owl:Class>
```

10. Forms

Warning

I provide very little information on forms in this document. Please download [AndrewBase.odt](#) from the database page on my web site.

Frank Schönheit [fs@openoffice.org] has the following to say:

To update an awt-control, make changes to the **model**, and the control which belongs to the model is automatically updated to be in sync with the model. Changing the control directly may lead to some inconsistency. For example, for a list box, do not use the XListBox interface provided by the control for selecting an item (XListBox::selectItem), but rather using the com.sun.star.awt.UnoControlListBoxModel::SelectedItems property of the model.

10.1. Introduction

A form or form document is a set of forms and/or controls such as a button or combo-box. Forms may be used to access data sources and other complicated things.

See Also:

<http://api.openoffice.org/docs/common/ref/com/sun/star/form/XFormsSupplier.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/form/module-ix.html>

To obtain the form, you must first obtain the draw page. The method to do this varies depending upon the document type. As a test, I inserted a button into a spreadsheet. I named the button “TestButton” and the form “TestForm”. I then created the following macro that changes the button label when it is clicked.

Listing 10.1: Change a button's label in a Calc document

```
Sub TestButtonClick
    Dim vButton, vForm
    Dim oForms

    oForms = ThisComponent.CurrentController.ActiveSheet.DrawPage.Forms
    vForm=oForms.getByName("TestForm")
    vButton = vForm.getByName("TestButton")
    vButton.Label = "Wow"
    Print vButton.getServiceName()
End Sub
```

If this had been a write document, you would obtain the form as follows:

```
oForm=THISCOMPONENT.DrawPage.Forms.getByName("FormName")
```

10.2. Dialogs

Most dialog examples start as follows:

```
Dim oDlg As Object
Sub StartDialog
    oDlg = CreateUnoDialog(DialogLibraries.Standard.Dialog1 )
    oDlg.execute()
End Sub
```

Notice that there is a variable called `oDlg` that has scope outside of the method that creates it. This is required because the dialog is manipulated by other subroutines that are called as event handlers. If these subroutines will access the dialog then they must be able to access a variable that references it.

Dialogs, like macro libraries, have their own hierarchy. In this example, I entered the macro code shown above. I then chose Tools->Macros and then I clicked on the “Organizer” button. You will notice that this is organizing things under the document with a section labeled Standard. When you click on the “New Dialog” button, the default dialog name is “Dialog1”. Because this code is run from a macro embedded in a document, `DialogLibraries` refers to the Document's library hierarchy. If you want to access the application library hierarchy from a Document's macro you must use `GlobalScope.DialogLibraries`.

The standard method of closing a dialog involves setting up an event handler that closes the dialog. I usually add a “close” button that calls a method similar to the following macro.

```
Sub CloseDialog
    oDlg.endExecute()
End Sub
```

Please forgive the great amount of detail present, but I feel that it is required for beginners. While editing my new dialog, I clicked on the “control” button on the tool-bar and I chose a command button. I then inserted the button. I right clicked the button and chose properties. From the General tab, I set the name of the button to “ExitButton” and the label to “Exit”. I then clicked on the “Events” tab and next to “When Initiating” I clicked on “...” to open the dialog that will allow me to assign actions to events. In the event section, I chose “When Initiating”. In the “Macro” section, I selected the “CloseDialog” subroutine and then I clicked on the “Assign” button. Although this handles both a mouse or keyboard activation, it is possible to have a different subroutine for “Key pressed” or “Mouse clicked” but I have no reason to differentiate.

Tip	You must have a variable referencing the dialog that has scope outside the subroutine that creates it so that other subroutines can access it.
Tip	From a document macro, <code>DialogLibraries.Standard.Dialog1</code> references a dialog in the current document; <code>GlobalScope.DialogLibraries.Standard...</code> references a global dialog.

10.2.1. Controls

Controls all share certain common features. A few are as follows:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/UnoControl.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XWindow.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/module-ix.html>

This allows controlling things such as visibility, enabled, size, and other common features. Many controls share common methods such as `setLabel(string)`. Many different event types are supported. In my experience, the most commonly used events are for notification of a state change on the control.

A control may be obtained from a dialog using the `getControl(control_name)` method. It is also possible to iterate through all of the controls if you desire.

10.2.2. Control Label

A label acts as regular text in the dialog box. It is usually used to label a control. It is possible to get and set the label text using `getText()` and `setText(string)`. It is also possible to specify the alignment of this text as left, centered, or right justified. It is common to right justify the text of a label so that it is closer to the control that it labels. See:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XFixedText.html>

10.2.3. Control Button

Usually, a control button is only used to call a subroutine when a button is pressed. It is also possible to call `setLabel(string)` to change the button label. See also:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XButton.html>

10.2.4. Text Box

A text box is used to hold standard text. It is possible to limit the maximum text length and control the maximum number of lines. It is possible to write your own formatting controls if you desire. The most commonly used methods are `getText()` and `setText(string)`. If you want a scroll bar, you should set it from the properties while designing the dialog.

There are special input boxes for dates, time, numerics, pattern, formatted, and currency. If you insert one, be certain to pay careful attention to the properties to see what they can do. You can turn off strict format checking and provide limited input ranges, for example. A pattern field has an input mask and a character mask. The input mask determines which user data can be entered. The character mask determines the state of the masked field when loading the form. The formatted field allows arbitrary formatting as allowed by OOo. If I wanted a field for social security number or percentages, I would use this field.

See:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XTextComponent.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XTimeField.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XDateField.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XCurrencyField.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XNumericField.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XPatternField.html>

10.2.5. List Box

A list box provides a list of values from which you may select a value. You can choose to enable multiple selections. To add items to a list box, I usually use something similar to `addItem(Array("one", "two", "three"), 0)`. It is also possible to remove items from a list box.

For a single selection, you can use `getSelectedItemPos()` to determine which item is selected. A -1 is returned if nothing is selected. If something is selected, 0 means the first item in the list. For multiple selections, use `getSelectedItemsPos()` which returns a sequence of shorts. See: <http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XListBox.html>

10.2.6. Combo Box

A combo box is an input field with an attached list box. This is sometimes called a drop down control. In my example, I set the combo box in two stages. First, I set the list box values and then I set the input box to properties.

```
aControl.addItem(Array("properties", "methods", "services"), 0)
aControl.setText("properties")
```

I then went to events and indicated that when the “Item status changed” the `NewDebugType` subroutine should be called. This will display all of the methods, properties, or services supported by the dialog. I did this to demonstrate a call back event that actually shows useful debug information. See:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XComboBox.html>

10.2.7. Check Box

Although a check box is usually only used to indicate a single state of yes or no, there is a common check box called a tristate check box. In OOo, a check box can have a state of 0, not checked, or 1, checked. You use `getState()` to obtain the current state. If you call `enableTriState(true)`, then a state of 2 is allowed. This state has a check mark in the box but then the box becomes Dim. You can set the state using `setState(int)`. See:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XCheckBox.html>

10.2.8. Option/Radio Button

The purpose of an radio button is usually to select one item from a group of items. Because of this, the typical usage in a dialog is to first insert a group box and then place the option buttons inside of the group box. To determine which one of the radio buttons has been selected, you call the `getState()` method on each of the radio buttons until you find the one that is selected. I could have used radio buttons rather than a drop down list in my example to choose what to display in my debug list box. See:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XRadioButton.html>

10.2.9. Progress Bar

In my example, I use a progress bar to show the progress of filling in the debug list box. This is probably not a good example because it happens so quickly, but it is still an example.

You can set the range of the progress bar using `setRange(min, max)`. This makes it easier to report your progress. Because I am processing a string, I set the min to 0 and the max to the length of the string. I then call `setValue(int)` with the current position in the string to show

how far along I am. See:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XProgressBar.html>

10.3. Obtaining Controls

If you want to enumerate the controls in a document form, the following will work

Listing 10.2: Enumerate the controls in a form.

```
Sub EnumerateControlsInForm
    Dim oForm, oControl, iNumControls%, i%
    'By default this is where the controls are
    oForm = ThisComponent.Drawpage.Forms.getByName("Standard")
    oControl = oForm.getByName("MyPushButton")
    MsgBox "Used get by name to get control named " & oControl.Name
    iNumControls = oForm.Count()
    For i = 0 To iNumControls - 1
        MsgBox "Control " & i & " is named " & oControl.Name
    Next
End Sub
```

To the controls in a dialog, use code as follows:

Listing 10.3: Enumerate the controls in a dialog.

```
x = oDlg.getControls()
For ii=LBound(x) To UBound(x)
    Print x(ii).getImplementationName()
Next
```

Usually, the controls will be retrieved by name rather than by enumeration.

10.3.1. Size and location of a Control by name.

Paolo Mantovani [mantovani.paolo@tin.it] indicated that a form control is placed on a drawing shape (com.sun.star.drawing.XControlShape), so you must obtain the shape underlying the control to manage the size and position. The Tools Library (module "ModuleControls") supplies some facilities in order to work with form controls. Check those functions:

```
GetControlModel()
GetControlShape()
GetControlView()
```

I wrote the following macro, which although it does obtain the control, it never uses it after that.

Listing 10.4: Obtain a control and it's shape by name.

```
Sub GetControlAndShape
    Dim oForm, oControl
```

```

'By default this is where the controls are
oForm = ThisComponent.Drawpage.Forms.getByName("Standard")
oControl = oForm.getByName("CheckBox")
Dim vShape
Dim vPosition, vSize
Dim s$
vShape = GetControlShape(ThisComponent, "CheckBox")
vPosition = vShape.getPosition()
vSize = vShape.getSize()
s = s & "Position = (" & vPosition.X & ", " & vPosition.Y & ")"
s = s & CHR$(10)
s = s & "Height = " & vSize.Height & " Width = " & vSize.Width
MsgBox s
End Sub

```

The following macro changes the size of the control.

Listing 10.5: Modify a control's size

```

Sub ModifyControlSize
    Dim oShape
    Dim oSize

    REM The tools library contains GetControlShape
    GlobalScope.BasicLibraries.LoadLibrary("Tools")

    REM Default main form name.
    oShape = GetControlShape(ThisComponent, "MainPushButton")
    oSize = oShape.getSize()

    REM Now, make the control larger!
    oSize.Height = 1.10 * oSize.Height
    oSize.Width = 1.10 * oSize.Width
    oShape.setSize(oSize)
    Print "The control is now 10% larger"
End Sub

```

10.3.2. Which control called a handler and where is it located?

Consider a text table containing buttons. All of the buttons call the same macro. Which cell contains the button?

1. Add an argument to button handler. The argument contains the “Source” property, which references the button.
2. Control shapes contain contain the “Control” property, which references the control's model. Iterate through the shapes and compare the shape's model to the control's model.

The following macro assumes that all returned shapes are control shapes, you probably want to verify that this is true.

Listing 10.6: Find a control's shape if you do NOT know the name.

```
Sub ButtonCall(x)
    Dim oButton          ' Button that was used to call the handler.
    Dim oModel           ' The model for the button.
    Dim oShape           ' The underlying button shape.
    Dim i As Long        ' Generic index variable.
    Dim bFound As Boolean ' True after find the matching shape.

    REM First, get the button used to call this routine.
    REM Save the button's model.
    oButton = x.Source
    oModel = oButton.getModel()

    REM Iterate through the controls
    i = ThisComponent.getDrawPage().getCount()
    bFound = False
    Do While (i > 0 AND NOT bFound)
        i = i - 1
        oShape = ThisComponent.getDrawPage().getByIndex(i)
        bFound = EqualUNOObjects(oShape.Control, oModel)
    Loop
    If bFound Then
        Print "The button is in cell " & oShape.getAnchor().Cell.CellName
    End If
End Sub
```

10.4. Choosing a File Using the File Dialog

The following example displays the standard choose file dialog.

Listing 10.7: Using the standard choose file dialog.

```
Sub ExampleGetAFileName
    Dim filterNames(1) As String
    filterNames(0) = "*.txt"
    filterNames(1) = "*.sxw"
    Print GetAFileName(filterNames())
End Sub

Function GetAFileName(Filternames()) As String
    Dim oFileDialog as Object
    Dim iAccept as Integer
    Dim sPath as String
    Dim InitPath as String
    Dim RefControlName as String
```



```

Dim oUcb as object
'Dim ListAny(0)

GlobalScope.BasicLibraries.LoadLibrary("Tools")
'Note: The following services must be called in the following order,
' otherwise the FileDialog Service is not removed.
oFileDialog = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
' Defaults to Open, you can also use Save
'oFileDialog.Initialize( _
'     Array(com.sun.star.ui.dialogs.TemplateDescription.FILESAVE_SIMPLE))
oUcb = createUnoService("com.sun.star.ucb.SimpleFileAccess")
'ListAny(0) = _
' com.sun.star.ui.dialogs.TemplateDescription.FILEOPEN_SIMPLE
'oFileDialog.initialize(ListAny())
AddFiltersToDialog(FilterNames(), oFileDialog)

'Set your initial path here!
'InitPath = ConvertToUrl(oRefModel.Text)

If InitPath = "" Then
    InitPath = GetPathSettings("Work")
End If
If oUcb.Exists(InitPath) Then
    oFileDialog.SetDisplayDirectory(InitPath)
End If
iAccept = oFileDialog.Execute()
If iAccept = 1 Then
    sPath = oFileDialog.Files(0)
    GetAFileName = sPath
    'If oUcb.Exists(sPath) Then
    '    oRefModel.Text = ConvertFromUrl(sPath)
    'End If
End If
oFileDialog.Dispose()
End Function

```

10.5. Center a dialog on the screen

Thanks to Berend Cornelius [Berend.Cornelius@sun.com] for providing this macro. The primary trick is to use the current controller to finally retrieve the component window and from there to retrieve the position and size of the window.

Listing 10.8: Center a dialog on the screen.

```

Sub CenterDialogOnScreen
    Dim CurPosSize As New com.sun.star.awt.Rectangle
    Dim oFrame
    oFrame = ThisComponent.getCurrentController().Frame

```

```

FramePosSize = oFrame.getComponentWindow.PosSize
xWindowPeer = oDialog.getPeer()
CurPosSize = oDialog.getPosSize()
WindowHeight = FramePosSize.Height
WindowWidth = FramePosSize.Width
DialogWidth = CurPosSize.Width
DialogHeight = CurPosSize.Height
iXPos = ((WindowWidth/2) - (DialogWidth/2))
iYPos = ((WindowHeight/2) - (DialogHeight/2))
oDialog.setPosSize(iXPos, iYPos, DialogWidth, _
                  DialogHeight, com.sun.star.awt.PosSize.POS)

oDialog.execute()
End Sub

```

10.5.1. DisplayAccess tells you all about the screen

Use the `com.sun.star.awt.DisplayAccess` service, which is undocumented as of April 2009.

Listing 10.9: Print display specific information.

```

Sub DisplayInfo
    Dim nCount%                ' Number of displays.
    Dim nDefaultDisplay%       ' Index of default (zero based).
    Dim bMultiDisplay As Boolean ' More than one display?
    Dim oDisplayAccess As Object ' DisplayAccess Service.
    Dim oDisplayInfo As Object
    Dim aScreenArea As Object
    Dim aWorkArea As Object
    Dim s$

    oDisplayAccess = createUnoService("com.sun.star.awt.DisplayAccess")
    nDefaultDisplay = oDisplayAccess.DefaultDisplay
    bMultiDisplay = oDisplayAccess.MultiDisplay
    nCount = oDisplayAccess.getCount()
    oDisplayInfo = oDisplayAccess.getByIndex(0)
    aScreenArea = oDisplayInfo.ScreenArea
    aWorkArea = oDisplayInfo.WorkArea

    s = "Number of displays: " & nCount & CHR$(10) & _
        "Default display: " & nDefaultDisplay & CHR$(10) & _
        "Multiple displays: " & bMultiDisplay & CHR$(10) & _
        "Screen Area = " & aScreenArea.X & ", " & aScreenArea.Y & ", " & _
        aScreenArea.Width & ", " & aScreenArea.Height & CHR$(10) & _
        "Work Area = " & aWorkArea.X & ", " & aWorkArea.Y & ", " & _
        aWorkArea.Width & ", " & aWorkArea.Height
    MsgBox s
End Sub

```

An astute person will notice that you can obtain this information for each screen.

10.6. Set the event listener for a control

The following little code snippet sets the event in a control to call the Test subroutine in the document standard library in the module named Module1. Thanks to Oliver Brinzing [OliverBrinzing@t-online.de] for the code.

Listing 10.10: Set the event listener for a control.

```
Sub SetEvent
    Dim oDoc as Object
    Dim oView as Object
    Dim oDrawPage as Object
    Dim oForm as Object
    Dim oEvents(0) As New com.sun.star.script.ScriptEventDescriptor

    oDoc = StarDesktop.getCurrentComponent
    oView = oDoc.CurrentController
    oDrawPage = oView.getActiveSheet.DrawPage

    ' get the first form
    oForm = oDrawPage.getForms.getByIndex(0)
    oEvents(0).ListenerType = "XActionListener"
    oEvents(0).EventMethod = "actionPerformed"
    oEvents(0).AddListenerParam = ""
    oEvents(0).ScriptType = "StarBasic"
    oEvents(0).ScriptCode = "document:Standard.Module1.Test"
    oForm.registerScriptEvent(0, oEvents(0))
End Sub

Sub Test(oEvt)
    Print oEvt.Source.Model.Name
End Sub
```

I suppose that a brief description on the registerScriptEvent method is in order. See <http://api.openoffice.org/docs/common/ref/com/sun/star/script/XEventAttacherManager.html> as a good starting point.

```
registerScriptEvent(index, ScriptEventDescriptor)
```

The oEvents variable is an array of event descriptors, see <http://api.openoffice.org/docs/common/ref/com/sun/star/script/ScriptEventDescriptor.html>, which describe the event that will be attached (EventMethod), and what it should call (ScriptCode). The above example could have just as easily used a single variable rather than an array, because the code only uses a single entry from the array. The first argument to the registerScriptEvent method is an index to the object that will “use” the event descriptor. The help pages assume that you know which objects are indexed and state that “If any object is attached under this index, then this event is attached automatically.” What it does not say is that the form acts as a container object for form components. These form components are available by name and by index.

If you do not know the index of the control, you can always use a macro such as the following:

Listing 10.11: Find control ID

```
Function getID(oForm, sname$) As Integer
    Dim i%
    getID() = -1
    If NOT oForm.HasByName(sname) Then
        Exit Function
    End If
    For i = 0 To oForm.GetCount() - 1
        If (oForm.GetByIndex(i).Name = sname) Then
            getID() = i
            Exit Function
        End If
    Next
End Function
```

I have little experience with this sort of thing, but, it appears as though you can set multiple events for a single control. I registered a listener that did not exist, then, I registered one that did exist. The second added to, rather than replaced, the original, so I received an error, and then the correct listener was called. I had to remove the listener that was in error.

10.7. Controlling a dialog I did not create.

Some dispatch commands load a dialog that you need to control. An interesting solution is presented by “ms777” on the oooforum: <http://www.oooforum.org/forum/viewtopic.phtml?t=22845>

1. Create a TopWindowListener.
2. Execute the dispatch.
3. The dispatch opens a new top window.
4. The top window listener is called.
5. Inside the top window listener, the accessibility interface is used to execute the desired user interactions.
6. Inside the top window listener, the dispatch dialog's OK Button is pressed to terminate the dispatch.
7. The TopWindowListener is removed.

There are a couple of caveats.

- Do not attempt to use a breakpoint or perform other actions after creating the top window listener. The listener is trying to handle the input.

- You need to understand the layout of the dialog that you wish to control. This is not an easy task, especially since you can not inspect the objects.
- Every top level window interface must be implemented or you may hang both OOo and your entire computer (this is new with OOo version 3.?). You also require all methods from all parent interfaces, which I had not initially done.

10.7.1. Inserting a formula

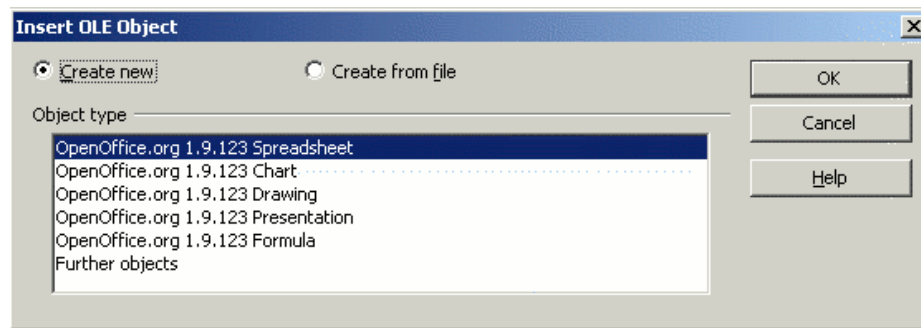


Figure 10.1: Insert an OLE object dialog.

Use the Insert OLE dialog to insert a Formula, or other object (see Figure 10.1). Manipulating the dialog requires that you know the objects that are inserted into the dialog so that you can access them. You also need to know how to interact with each object.

Table 10.1: Accessible content in the Insert OLE dialog.

Child	Name	Type
0	Create new	RADIO_BUTTON
1	Create from file	RADIO_BUTTON
2		PANEL
3	Object type	SEPARATOR
4	OK	PUSH_BUTTON
5	Cancel	PUSH_BUTTON
6	Help	PUSH_BUTTON

The macro in Listing 10.12 demonstrates how to use the Insert OLE dialog in Figure 2.1. The macro inserts an equation, and leaves the cursor in the insert equation mode (so it does not really set the equation).

Listing 10.12: Insert a formula into writer the hard way.

```
Sub InsertFormulaIntoWriter()
    Dim oFrame ' Frame from the current window.
    Dim oToolkit ' Container window's com.sun.star.awt.Toolkit
```

```

Dim oDisp      ' Dispatch helper.
Dim oList      ' XTopWindowListener that handles the interactions.
Dim s$

REM Get the com.sun.star.awt.Toolkit
oFrame  = ThisComponent.getCurrentController().getFrame()
oToolkit = oFrame.getContainerWindow().getToolkit()
s$ = "com.sun.star.awt.XTopWindowListener"
oList  = createUnoListener("TopWFormula_", s$)
oDisp  = createUnoService("com.sun.star.frame.DispatchHelper")

REM Insert an OLE object!
oToolkit.addTopWindowListener(oList)
oDisp.executeDispatch(oFrame, ".uno:InsertObject", "", 0, Array())
oToolkit.removeTopWindowListener(oList)
End Sub

Sub TopWFormula_windowOpened(e As Object)
    Dim oAC
    Dim oACRadioButtonNew
    Dim oACList
    Dim oACButtonOK

    REM Get the accessible window, which is the entire dialog.
    oAC = e.source.AccessibleContext

    REM Get the buttons
    oACRadioButtonNew = oAC.getAccessibleChild(0).AccessibleContext
    DIM oAC2
    oAC2 = oAC.getAccessibleChild(2)
    oACList = oAC2.AccessibleContext.getAccessibleChild(0)
    oACButtonOK = oAC.getAccessibleChild(4).AccessibleContext

    REM Select "Create New"
    oACRadioButtonNew.doAccessibleAction(0)

    REM Access the Fifth item in the list (as in 0, 1, 2, 3, 4...)
    oACList.selectAccessibleChild(4)

    REM The accessible action of a command button is to "use" it.
    oACButtonOK.doAccessibleAction(0)
End Sub

Sub TopWFormula_windowClosing(e As Object)
End Sub

Sub TopWFormula_windowClosed(e As Object)

```

```

End Sub

Sub TopWFormula_windowMinimized(e As Object)
End Sub

Sub TopWFormula_windowNormalized(e As Object)
End Sub

Sub TopWFormula_windowActivated(e As Object)
End Sub

Sub TopWFormula_windowDeactivated(e As Object)
End Sub

Sub TopWFormula_disposing(e As Object)
End Sub

```

10.7.2. Discovering the accessible content (by Andrew)

You can not directly inspect the objects so it is difficult to determine the accessible content so I wrote the macro in Listing 10.13. The macro inspects all of the child content and then prints its name and type. I did not spend much time worrying about error handling, so be warned.

Listing 10.13: Inspect the accessible content.

```

Function InspectAccessibleContent(sLead$, oAC) As String
    REM Author Andrew Pitonyak
    Dim x
    Dim s$
    Dim i%
    Dim sL$
    Dim s1$, s2$
    Dim oACUse
    Dim oACChild

    s1 = "com.sun.star.accessibility.XAccessibleContext"
    s2 = "com.sun.star.accessibility.XAccessible"
    If HasUnoInterfaces(oAC, s1) Then
        oACUse = oAC
        sL = sLead
    ElseIf HasUnoInterfaces(oAC, s2) then
        oACUse = oAC.AccessibleContext
        sL = sLead & ".AC."
    Else
        Exit Function
    End If

    x = Array( "UNKNOWN", "ALERT", "COLUMN_HEADER", "CANVAS", _

```

```

"CHECK_BOX", "CHECK_MENU_ITEM", "COLOR_CHOOSER", "COMBO_BOX", _
"DATE_EDITOR", "DESKTOP_ICON", "DESKTOP_PANE", _
"DIRECTORY_PANE", "DIALOG", "DOCUMENT", _
"EMBEDDED_OBJECT", "END_NOTE", "FILE_CHOOSER", "FILLER", _
"FONT_CHOOSER", "FOOTER", "FOOTNOTE", "FRAME", "GLASS_PANE", _
"GRAPHIC", "GROUP_BOX", "HEADER", "HEADING", "HYPER_LINK", _
"ICON", "INTERNAL_FRAME", "LABEL", "LAYERED_PANE", "LIST", _
"LIST_ITEM", "MENU", "MENU_BAR", "MENU_ITEM", "OPTION_PANE", _
"PAGE_TAB", "PAGE_TAB_LIST", "PANEL", "PARAGRAPH", _
"PASSWORD_TEXT", "POPUP_MENU", "PUSH_BUTTON", "PROGRESS_BAR", _
"RADIO_BUTTON", "RADIO_MENU_ITEM", "ROW_HEADER", "ROOT_PANE", _
"SCROLL_BAR", "SCROLL_PANE", "SHAPE", "SEPARATOR", "SLIDER", _
"SPIN_BOX", "SPLIT_PANE", "STATUS_BAR", "TABLE", "TABLE_CELL", _
"TEXT", "TEXT_FRAME", "TOGGLE_BUTTON", "TOOL_BAR", "TOOL_TIP", _
"TREE", "VIEW_PORT", "WINDOW")

For i=0 to oACUse.AccessibleChildCount-1
    oACChild = oACUse.AccessibleChild(i)
    If HasUnoInterfaces(oACChild, s1) Then
        s = s & sL & i & " " & oACChild.AccessibleName & " " & _
            x(oACChild.AccessibleRole) & CHR$(10)
    ElseIf HasUnoInterfaces(oACChild, s2) Then
        oACChild = oACChild.AccessibleContext
        s = s & sL & "AC." & i & " " & oACChild.AccessibleName & _
            " " & x(oACChild.AccessibleRole) & CHR$(10)
    Else
        Exit Function ' unexpected situation
    End If
    Dim nTemp As Long
    nTemp = com.sun.star.accessibility.AccessibleRole.PANEL
    If oACChild.AccessibleRole = nTemp Then
        s = s & InspectIt(sL & " " & x(oACChild.AccessibleRole) & _
            ".", oACChild)
    End If
Next
InspectAccessibleContent = s
End Function

```

The Options dialog is very complicated, which makes it an excellent item to inspect. The code shown in Listing 10.14 is not complete. The code is modified from Listing 10.12; the required changes should be obvious. The code opens the Options dialog, and leaves it open. Information is displayed for whichever tab happens to be visible.

Listing 10.14: *How to inspect the accessible content.*

```

Option Explicit
Private sss$

Sub ManipulateOptions()

```



```

Dim oFrame      ' Frame from the current window.
Dim oToolkit    ' Container window's com.sun.star.awt.Toolkit
Dim oDisp       ' Dispatch helper.
Dim oList       ' XTopWindowListener that handles the interactions.

REM Get the com.sun.star.awt.Toolkit
oFrame = ThisComponent.getCurrentController().getFrame()
oToolkit = oFrame.getContainerWindow().getToolkit()
oList = createUnoListener("TopWFormula_", _
                          "com.sun.star.awt.XTopWindowListener")
oDisp = createUnoService("com.sun.star.frame.DispatchHelper")

oToolkit.addTopWindowListener(oList)
oDisp.executeDispatch(oFrame, ".uno:OptionsTreeDialog", "", _
                    0, Array())
oToolkit.removeTopWindowListener(oList)
MsgBox sss
End Sub

Sub TopWFormula_windowOpened(e As Object)
    sss = InspectAccessibleContent("", e.source.AccessibleContext)
End Sub

```

Because of the complexity of some dialogs, ms777 wrote some search routines to find controls based on their name, type, or description. While searching, a delay is performed if the content is not found. When the dialog is manipulated, it is possible that the accessible content has not yet been updated to be displayed (so it might not exist).

Listing 10.15: Search for the specified child.

```

Function SearchOneSecForChild(oAC As Object, sName$, sDescription$, lRole As
Long) As Object
    Dim k%
    Dim oRes

    For k=1 To 50
        oRes = SearchForChild(oAC, sName, sDescription, lRole)
        If NOT IsNull(oRes) Then
            SearchOneSecForChild = oRes
            Exit Function
        End If
        Wait(20)
    Next
End Function

Function SearchForChild(oAC As Object, sName$, sDescription$, lRole As Long) As
Object
    Dim oAC1
    Dim oACChild

```

```

Dim k%
Dim bFound As Boolean

If HasUnoInterfaces(oAC, "com.sun.star.accessibility.XAccessibleContext") Then
    oAC1 = oAC
ElseIf HasUnoInterfaces(oAC, "com.sun.star.accessibility.XAccessible") Then
    oAC1 = oAC.AccessibleContext
Else
    Exit Function ' unexpected situation, maybe caused by some intermediate
state
End If

For k=0 To oAC.AccessibleChildCount-1
    oACChild = oAC.getAccessibleChild(k)
    If NOT HasUnoInterfaces(oACChild,
"com.sun.star.accessibility.XAccessibleContext") Then
        If HasUnoInterfaces(oACChild, "com.sun.star.accessibility.XAccessible")
Then
            oACChild = oACChild.AccessibleContext
        Else
            Exit Function ' unexpected situation, maybe caused by some
intermediate state
        End If
    End If
    bFound = true

    If sName<>" " Then bFound = bFound AND (oACChild.AccessibleName =
sName)
    If sDescription<>" " Then bFound = bFound AND _
(oACChild.AccessibleDescription =
sDescription)
    If lRole > -1 Then bFound = bFound AND (oACChild.AccessibleRole =
lRole)

    If bFound Then
        SearchForChild = oACChild
        Exit Function
    End If
Next
End Function

```

10.7.3. Manipulating the Options dialog

The Options dialog (Tools | Options) is very complicated, even more so by the fact that the dialog can be in any state. The following macro collapses all of the tree structures to place the dialog into a known state.

Listing 10.16: *Collapse all of the trees in the options dialog.*

```

Sub CollapseAllTrees(oAC)

```

```

REM Author ms777
REM Modified by Andrew Pitonyak
Dim oACChild
Dim oACChild1
Dim k%
Dim k1%

'make a defined state: collaps all items in all trees
For k=0 To oAC.getAccessibleChildCount-1
    oACChild = oAC.getAccessibleChild(k)
    If HasUnoInterfaces(oACChild, _
        "com.sun.star.accessibility.XAccessibleContext") Then
        If oACChild.AccessibleRole =
com.sun.star.accessibility.AccessibleRole.TREE Then
            For k1=0 To oACChild.getAccessibleChildCount-1
                oACChild1 = oACChild.getAccessibleChild(k1)
                If HasUnoInterfaces(oACChild1, _
                    "com.sun.star.accessibility.XAccessibleAction") Then
                    If oACChild1.AccessibleStateSet.contains(_
                        com.sun.star.accessibility.AccessibleStateType.EXPANDED) Then
                        oACChild1.doAccessibleAction(0)
                    End If
                End If
            Next k1
        End If
    End If
Next k
End Sub

```

Finally, after opening the Options dialog, you need to manipulate it.

Listing 10.17: Set the user's Title in the Options dialog.

```

Sub TopWFormula_windowOpened(e As Object)
    Dim oAC
    Dim oACTree
    Dim oACOO
    Dim oACButtonOK
    Dim oACUserDataPanel
    Dim oACTitle
    Dim bResult As boolean

    REM Get the accessible window, which is the entire dialog.
    oAC = e.source.AccessibleContext
    CollapseAllTrees(oAC)
    'e.source.setVisible(false)

    'Open up the tree item "OpenOffice.org"
    oACTree = SearchOneSecForChild(oAC, "", "", _

```

```

        com.sun.star.accessibility.AccessibleRole.TREE)
oACOO = SearchOneSecForChild(oACTree, "OpenOffice.org", "", _
        com.sun.star.accessibility.AccessibleRole.LABEL)
oACOO.doAccessibleAction(0)

'Select the first entry in OpenOffice.org (UserData)
oACOO.selectAccessibleChild(0)

'now set the UserData
oACUserDataPanel = SearchOneSecForChild(oAC, "User Data", "", _
        com.sun.star.accessibility.AccessibleRole.PANEL)
oACTitle = SearchOneSecForChild(oACUserDataPanel, "Title/Position", _
        "Type your title in this field.", _
        com.sun.star.accessibility.AccessibleRole.TEXT)
If NOT IsNULL(oACTitle) AND NOT IsEmpty(oACTitle) Then
    bResult = oACTitle.setText("Meister aller Klassen")
EndIf

REM press the OK button
oACButtonOK = SearchOneSecForChild(oAC, "OK", "", _
        com.sun.star.accessibility.AccessibleRole.PUSH_BUTTON)
oACButtonOK.doAccessibleAction(0)
End Sub

```

10.7.4. Listing the supported printers

This is one of my favorite examples presented by ms777. Get the list of all supported printers by displaying the print dialog, pulling the printer list from the dialog, and then canceling the dialog. The macro in Listing 10.18 requires the macro shown in Listing 10.15.

Listing 10.18: *Get a list of all supported printers.*

```

Option Explicit
Public GetPrinterArray735 As Any

Sub PrintAllPrinters
    REM Author ms777.
    Dim arPrinter
    Dim s$
    Dim k%

    arPrinter = GetAllPrinters()
    s = ""
    For k = 0 To UBound(arPrinter)
        s = s + k + ": " + arPrinter(k) + Chr(10)
    Next k
    MsgBox s
End Sub

```

```

Function GetAllPrinters() As Any
    'On Error Resume Next
    Dim oFrame      ' Frame from the current window.
    Dim oToolkit    ' Container window's com.sun.star.awt.Toolkit
    Dim oDisp       ' Dispatch helper.
    Dim oList       ' XTopWindowListener that handles the interactions.

    REM Get the com.sun.star.awt.Toolkit
    oFrame = ThisComponent.getCurrentController().getFrame()
    oToolkit = oFrame.getContainerWindow().getToolkit()
    oList = createUnoListener("TopWPrint_",
"com.sun.star.awt.XTopWindowListener")
    oDisp = createUnoService("com.sun.star.frame.DispatchHelper")

    oToolkit.addTopWindowListener(oList)
    REM Insert an OLE object!
    oDisp.executeDispatch(oFrame, ".uno:Print", "", 0, Array())
    oToolkit.removeTopWindowListener(oList)
    GetAllPrinters = GetPrinterArray735
End Function

Sub TopWPrint_windowOpened(e As Object)
    Dim oAC
    Dim oACComboBox
    Dim oACList
    Dim oACCancel
    Dim k%

    oAC = e.source.AccessibleContext
    e.source.setVisible(false)

    oACComboBox = SearchOneSecForChild(oAC, "", "", _
com.sun.star.accessibility.AccessibleRole.COMBO_BOX)
    oACList = SearchOneSecForChild(oACComboBox, "", "", _
com.sun.star.accessibility.AccessibleRole.LIST)
    oACCancel = SearchOneSecForChild(oAC, "Cancel", "", _
com.sun.star.accessibility.AccessibleRole.PUSH_BUTTON)

    Dim sResult(oACList.AccessibleChildCount-1) As String

    For k = 0 to oACList.AccessibleChildCount-1
        sResult(k) = oACList.getAccessibleChild(k).AccessibleName
    Next k
    oACCancel.doAccessibleAction(0)
    GetPrinterArray735 = sResult()
End Sub

```

```

Sub TopWPrint_windowClosing(e As Object)
End Sub

Sub TopWPrint_windowClosed(e As Object)
End Sub

Sub TopWPrint_windowMinimized(e As Object)
End Sub

Sub TopWPrint_windowNormalized(e As Object)
End Sub

Sub TopWPrint_windowActivated(e As Object)
End Sub

Sub TopWPrint_windowDeactivated(e As Object)
End Sub

```

10.7.5. Finding an open window

The following macro is presented by ms777 as part of his inspection routines. I felt that it was important enough to pull it out. The idea is that you open a dialog, notice the title, and then call this macro to obtain a reference to the window. This macro is used in

Listing 10.19: Find an open dialog based on its title.

```

'----- GetWindowOpen
REM Iterate through the open dialogs and find the one that starts with
REM sTitle.
Function GetWindowOpen(sTitle as String) As Object
    Dim oToolkit
    Dim lCount As Long
    Dim k As Long
    Dim oWin

    oToolkit = Stardesktop.ActiveFrame.ContainerWindow.Toolkit
    lCount = oToolkit.TopWindowCount

    For k=0 To lCount -1
        oWin = oToolkit.getTopWindow(k)
        If HasUnoInterfaces(oWin, "com.sun.star.awt.XDialog") Then
            If left(oWin.Title, len(sTitle)) = sTitle Then
                GetWindowOpen = oWin
                Exit Function
            EndIf
        EndIf
    Next k
End Function

```

10.7.6. Inspecting accessible content (by ms777)

I wrote the macro in Listing 10.13 to inspect content. Ms777 then provided me with a macro that he wrote. I prefer his way. I did not know how to find an open window, as shown in Listing 10.19. The trick to using this next macro, is to first open the dialog, and then run the macro with name. I considered writing a dialog that allowed you to choose the window, but I do not have the time. I made minor modifications to declare every variable, and to provide some minor improvements in the displayed results.

Listing 10.20: Inspect the named dialog.

```
REM First, open the dialog that you want to inspect.
REM Next, run the following macro:
Sub InspectOpenWindow
    Dim oWin
    Dim oAC

    'This Function identifies the top window, whose title starts with "Options"
    'I used Tools | Options to open the window.
    oWin = GetWindowOpen("Options")
    oAC = oWin.AccessibleContext

    'This generates a hierarchical list of the accessibility tree
    call AnalyzeCreateSxc(oAC)
End Sub

'----- GetWindowOpen
REM Iterate through the open dialogs and find the one that starts with
REM sTitle.
Function GetWindowOpen(sTitle as String) As Object
    Dim oToolkit
    Dim lCount As Long
    Dim k As Long
    Dim oWin

    oToolkit = StarDesktop.ActiveFrame.ContainerWindow.Toolkit
    lCount = oToolkit.TopWindowCount

    For k=0 To lCount -1
        oWin = oToolkit.getTopWindow(k)
        If HasUnoInterfaces(oWin, "com.sun.star.awt.XDialog") Then
            If left(oWin.Title, len(sTitle)) = sTitle Then
                GetWindowOpen = oWin
                Exit Function
            EndIf
        EndIf
    Next k
```

```
End Function
```

```
'----- AnalyzeCreateSxc

Sub AnalyzeCreateSxc(oAC As Object)
    Dim kRowSheetOut As Long
    Dim kColSheetOut as Long
    Dim oDoc as Object
    Dim oSheetOut

    oDoc =
StarDesktop.LoadComponentFromUrl("private:factory/scalc", "_default", 0, Array())

    oSheetOut = oDoc.sheets.getByIndex(0)
    kRowSheetOut = 0
    kColSheetOut = 0
    call Analyze(oAC, oSheetOut, kRowSheetOut, kColSheetOut, "")
End sub

Sub Analyze(oAC as Object, oSheetOut as Object, kRowSheetOut as Long,
kColSheetOut as Long, sParentTrace as String)
    Dim sName As String
    Dim k As Long
    Dim kMax As Integer
    Dim lRole As Long
    Dim sProps() As String
    Dim sObjName As String
    Dim k1 As Long
    Dim k2 As Long
    Dim kColSheetOut As Long
    Dim oAC1
    Dim oCell
    Dim sTraceHelper As String

    If HasUnoInterfaces(oAC, "com.sun.star.accessibility.XAccessibleContext") Then

        oSheetOut.getCellByPosition(kColSheetOut, kRowSheetOut).String = _
            AccessibleObjectDescriptionString(oAC) + " (" + sParentTrace + ")"
        kRowSheetOut = kRowSheetOut + 1

        kMax = -1
        on error resume next
        kMax = oAC.getAccessibleChildCount()-1
        on error goto 0

        ' show maximum 30 childs
```



```

If kMax>=0 Then
    If kMax>30 Then
        k1 = 15
        k2 = kMax-15
    Else
        k1=-1
        k2=0
    EndIf
    For k=0 To k1
        kColSheetOut = kColSheetOut + 1
        sTraceHelper = IIF(Len(sParentTrace) = 0, "", ", ", ")
        Call Analyze(oAC.getAccessibleChild(k), oSheetOut, kRowSheetOut, _
            kColSheetOut, sParentTrace + sTraceHelper + k)
        kColSheetOut = kColSheetOut - 1
    Next k
    For k=k2 To kMax
        kColSheetOut = kColSheetOut + 1
        sTraceHelper = IIF(Len(sParentTrace) = 0, "", ", ", ")
        Call Analyze(oAC.getAccessibleChild(k), oSheetOut, kRowSheetOut, _
            kColSheetOut, sParentTrace + sTraceHelper + k)
        kColSheetOut = kColSheetOut - 1
    Next k
EndIf
EndIf

If HasUnoInterfaces(oAC, "com.sun.star.accessibility.XAccessible") Then
    oAC1 = oAC.AccessibleContext
    If EqualUnoObjects(oAC1, oAC) Then
        Exit Sub
    EndIf
    kColSheetOut = kColSheetOut + 1
    Call Analyze(oAC1, oSheetOut, kRowSheetOut, kColSheetOut, sParentTrace + ",
AC" )
    kColSheetOut = kColSheetOut - 1
EndIf
End sub

Function AccessibleObjectDescriptionString(oAC As Object) As String
    Dim s As String
    Dim sText As String
    Dim sProps()
    Dim k As Long
    Dim lRole As Long
    Dim lActionCount As Long

    s = ""
    'AccessibleName

```

```

On Error Resume Next
sText = oAC.getAccessibleName()
On Error Goto 0
If sText = "" Then
    sText = "--"
EndIf
s = s + sText

'AccessibleObject name
sProps = Split(oAC.Dbg_Properties, "")
s = s + " " + sProps(1)

'AccessibleRole
lRole=-1
On Error Resume Next
lRole = oAC.getAccessibleRole()
On Error Goto 0
If lRole <>-1 Then
    s = s + " Role: " + Choose(lRole+1, "UNKNOWN", "ALERT", "COLUMN_HEADER",
"CANVAS", "CHECK_BOX", "CHECK_MENU_ITEM", "COLOR_CHOOSER", "COMBO_BOX",
"DATE_EDITOR", "DESKTOP_ICON", "DESKTOP_PANE", "DIRECTORY_PANE", "DIALOG",
"DOCUMENT", "EMBEDDED_OBJECT", "END_NOTE", "FILE_CHOOSER", "FILLER",
"FONT_CHOOSER", "FOOTER", "FOOTNOTE", "FRAME", "GLASS_PANE", "GRAPHIC",
"GROUP_BOX", "HEADER", "HEADING", "HYPER_LINK", "ICON", "INTERNAL_FRAME",
"LABEL", "LAYERED_PANE", "LIST", "LIST_ITEM", "MENU", "MENU_BAR", "MENU_ITEM",
"OPTION_PANE", "PAGE_TAB", "PAGE_TAB_LIST", "PANEL", "PARAGRAPH",
"PASSWORD_TEXT", "POPUP_MENU", "PUSH_BUTTON", "PROGRESS_BAR", "RADIO_BUTTON",
"RADIO_MENU_ITEM", "ROW_HEADER", "ROOT_PANE", "SCROLL_BAR", "SCROLL_PANE",
"SHAPE", "SEPARATOR", "SLIDER", "SPIN_BOX", "SPLIT_PANE", "STATUS_BAR", "TABLE",
"TABLE_CELL", "TEXT", "TEXT_FRAME", "TOGGLE_BUTTON", "TOOL_BAR", "TOOL_TIP",
"TREE", "VIEW_PORT", "WINDOW")
EndIf

'AccessibleDescription
On Error Resume Next
s = s + " " + oAC.getAccessibleDescription()
On Error Goto 0

If HasUnoInterfaces(oAC, "com.sun.star.accessibility.XAccessibleAction") Then
    s = s + " Actions:"
    lActionCount = oAC.getAccessibleActionCount
    For k=0 To lActionCount-1
        s = s + " " + k + ": " + oAC.getAccessibleDescription(k)
    Next k
EndIf

AccessibleObjectDescriptionString = s
End Function

```

11. XForms

I have done little with XForms – I know little about them and provide minimal coverage here. I do know that things are not as I had initially expected. For example, I noticed that XForm documents with input fields do not contain text fields.

Instead of creating an empty text document, you need to create a text document with a special parameter (which I don't remember right now :-\) to create an XML form document

- XForm documents are usual form documents, so building them on the fly should be difficult, if you ever built database forms (IIRC, you did ...)
- To access the logical XForms in a XML form document, `css.xforms.XFormsSupplier` is your entry point - it's supported by your newly created document
- Conceptually, binding form controls to XForms nodes is done via abstract `css.form.binding.ValueBinding` mechanisms. The whole `css.form.binding` namespace describes the binding concept. `css.xforms.Binding`, in turn, describes a XForms-related `ValueBinding` which can be associated with a form control. (unrelated side note: Another implementation of the `ValueBinding` service is used to bind form controls to spreadsheet cells)
- Additionally, `css.form.validation` describes concepts for validating form control content on-the-fly, which is also used by `css.xforms.Binding`.

12. Database

I have an entire document on database access, go read it:

<http://www.pitonyak.org/database/AndrewBase.odt>

There is more content on my web page as well:

<http://www.pitonyak.org/Database>

13. Investment example

13.1. Internal Rate of Return (IRR)

If I take an amount P , and I invest it in a bank with a simple interest rate r , then in one year, I will earn _____ dollars in interest. In other words, at the end of one year, I will have $P(1+r)$ dollars. If I leave my money in for n years, the future value FV of my money is shown in *Equation 13.1*.

Equation 13.1

If I leave my money in for m days, where m is less than one year, the amount of interest is typically prorated (see *Equation 13.2*).

$$FV = P(1 + r \frac{m}{365})$$

Equation 13.2

Assume that you periodically place money into an account of some kind. Later, you want to know how well the account has done.

Table 13.1. Terms for calculating interest.

Term	Description
P_i	Amount of the i^{th} deposit (P stands for principle). For example, P_0 is the amount of the first deposit or payment.
t_i	Date (t stands for time) of the i^{th} deposit.
FV	Final (or future) value.
r	Annual rate of return (how much did I earn each year).
C_i	How much money I added as of the i^{th} deposit.

Adding all of the deposits produces the amount invested.

$$C_n = \sum_{i=0}^n P_i$$

Equation 13.3

13.1.1. Using only simple interest

An ultra simplified example of *Equation 13.2* extended to handle multiple deposits over time (even if it is NEVER used in this way) (see *Equation 13.4*).

$$FV = \sum_{i=0}^n P_i (1 + r(t - t_i)/365)$$

Equation 13.4

Now expand the terms.

$$FV = \sum_{i=0}^n P_i + \sum_{i=0}^n P_i (t - t_i) = 1000$$

Equation 13.5

For this specific example, we know everything but the rate of return, which is easy to solve. First, subtract both sides by the amount deposited.

$$FV - \sum_{i=0}^n P_i = \sum_{i=0}^n P_i (t - t_i)$$

Equation 13.6

Rearrange a little.

$$365 \frac{FV - \sum_{i=0}^n P_i}{\sum_{i=0}^n P_i (t - t_i)} = r$$

Equation 13.7

Listing 13.1, usable as a Calc function, calculates the rate of return shown in *Equation 13.7*.

Listing 13.1: Calculate the yearly rate of return.

```
Function YearlyRateOfReturn(FinalValue As Double, _
    CurrentDate As Date, DepositDates(), Deposits()) As Double
    Dim iRow As Integer
    Dim iCol As Integer
    Dim dSum As Double
    Dim d As Double
    Dim dTotalDeposit As Double
    Dim dRate As Double
    Dim nMaxRows As Integer
    Dim nLB As Integer

    If UBound(DepositDates()) <> UBound(Deposits()) Then
        MsgBox "The number of dates does not match the number of deposits"
        YearlyRateOfReturn = 0.0
        Exit Function
    End If

    dSum = 0
    dTotalDeposit = 0
    nLB = LBound(Deposits(), 1)
    For iRow = nLB To UBound(Deposits(), 1)
        d = 0
        For iCol = nLB To UBound(Deposits(), 2)
            d = d + Deposits(iRow, iCol)
        Next
        dTotalDeposit = dTotalDeposit + d
        dSum = dSum + (CurrentDate - DepositDates(iRow, nLB)) * d
    Next
```

```

Next

If dSum = 0 Then
    dRate = 0.0
Else
    dRate = 365 * (FinalValue - dTotalDeposit) / dSum
End If
'Print "Final rate = " & dRate
YearlyRateOfReturn = dRate
End Function

```

13.1.2. Compound the interest

Assume that my money compounds k times per year and I let it compound n times at a rate r .

$$FV = P \left(1 + \frac{r}{k}\right)^n$$

Equation 13.8

If I make n payments, however, then the formula is a much more complicated, primarily because I do not make payments on regular intervals. The best that I can say is that the formula is similar to *Equation 13.9*.

$$FV = \sum_{t=0}^n P_t \left(1 + \frac{r}{k}\right)^{n-t}$$

Equation 13.9

All variables in *Equation 13.9* are known except for r . An astute reader will realize that this is a polynomial equation with respect to r , and therefore, does not in general contain an easy solution.

Calc supports the IRR function to calculate the internal rate of return based on cash flow values at regular intervals. There are two problems with the IRR function; the payment interval must be regular, and an initial guess is required.

The best course of action seems to be:

1. Massage the data into a form usable by IRR.
2. Use *Listing 13.1* to generate an initial guess if required.

The first step strikes me as the most difficult step. If I have time, perhaps I will do this ??

14. Handlers and Listeners

A handler, for the sake of this chapter, is any code that uses a call back or is somehow related to handling some sort of event. For example, a key is pressed, and an event is fired indicating that a key was pressed.

14.1. *Warning, your handler may disappear*

Using a handler seems so easy:

1. Write your handler
2. Register or add your handler to some object
3. When finished, remove your handler.

Some handlers, such as the key handler, use the current controller. Unfortunately, sometimes the controller is disposed and a new one is created. For example, while switching to a print preview mode, the component is detached from the frame to create a new view of the document; the controller is disposed when the component is detached from the frame.

You can work around this with a frame action listener. Unfortunately, the details are complicated. There are more details here:

<http://www.openoffice.org/servlets/ReadMsg?list=dev&msgNo=21148>

14.2. *xKeyHandler example*

Leston Buell [bulbul@ucla.edu] wrote a key handler that watches key press events and translates specific key combinations into Esperanto characters.

A global variable is used to hold a reference to the key handler. This is important because a Global variable holds its value between macro executions. A reference to the handler is required so that it can be removed later.

Listing 14.1: Global variables for Esperanto translator.

```
REM Author:Leston Buell [bulbul@ucla.edu]  
Global oComposerDocView  
Global oComposerKeyHandler  
Global oComposerInputString
```

As you type, the characters will be buffered. After two keys have been pressed, the character will be translated into a single Esperanto character.

Listing 14.2: Translate a two character string to Esperanto.

```
Function GetTranslation( oString ) as String  
    Select Case oString  
        Case "^C", "Ch"  
            GetTranslation = "Ĉ"  
        Case "^c", "ch"  
            GetTranslation = "ĉ"  
        Case "^G", "Gh"
```



```

        GetTranslation = "Ĝ"
    Case "^g", "gh"
        GetTranslation = "ĝ"
    Case "^H", "Hh"
        GetTranslation = "Ĥ"
    Case "^h", "hh"
        GetTranslation = "ĥ"
    Case "^J", "Jh"
        GetTranslation = "Ĵ"
    Case "^j", "jh"
        GetTranslation = "ĵ"
    Case "^S", "Sh"
        GetTranslation = "Ŝ"
    Case "^s", "sh"
        GetTranslation = "ŝ"
    Case "uU", "Uh"
        GetTranslation = "Ŭ"
    Case "uu", "uh"
        GetTranslation = "ŭ"
End Select
End Function

```

After two keystrokes have been converted into an Esperanto character, it is inserted into the document. I am surprised that `oDocView` and `oKeyHandler` are passed as arguments because they are available from the global variables.

Listing 14.3: Insert the Esperanto character into the document.

```

Function InsertString( oString, oDocView, oKeyHandler )
    Dim oVCurs
    Dim oText
    Dim oCursor

    'oVCurs = ThisComponent.getCurrentController().getViewCursor()
    'oText = ThisComponent.getText()
    oVCurs = oDocView.getViewCursor()
    oText = oVCurs.getText()
    oCursor = oText.createTextCursorByRange( oVCurs.getStart() )
    'Text insertion re-fires the key events (twice!),
    'Remove the handler before insertion, then add it again afterwards.
    oDocView.removeKeyHandler( oKeyHandler )
    oText.insertString( oCursor.getStart(), oString, true )
    oDocView.addKeyHandler( oKeyHandler )
End Function

```

Call the compose macro to create and register the key handler to the current document's current controller. A reference to the key handler is stored in a global variable so that it can later be removed. A reference to the document's current controller is also saved so that it can

later be removed. After calling *Listing 14.4*, every key that you type will go to this key listener.

Listing 14.4: Register the translator.

```
Sub Compose
    oComposerDocView = ThisComponent.getCurrentController
    oComposerKeyHandler = createUnoListener( "Composer_", _
        "com.sun.star.awt.XKeyHandler" )
    oComposerDocView.addHandler( oComposerKeyHandler )
    oComposerInputString = ""
End Sub
```

Removing the key listener is easy.

Listing 14.5: Remove the translator.

```
Sub ExitCompose
    oComposerDocView.removeKeyHandler( oComposerKeyHandler )
    oComposerInputString = ""
End Sub
```

The “handler” methods that are called automatically are prefaced with “Composer_” as dictated in *Listing 14.4*. The listener that is created defines the methods that must be created. The first method does very little; false is returned that the event is not handled by this function.

Listing 14.6: The keyReleased method.

```
Function Composer_keyReleased( oEvt ) as Boolean
    Composer_keyReleased = False
End Function
```

As keys are pressed, they are stored in the oComposerInputString. The event contains the key that was just pressed. If oComposerInputString already contains one character, then two characters are present, and *Listing 14.2* is used to convert the string into an appropriate character. The converted character is inserted into the document using *Listing 14.3*.

Listing 14.7: The primary worker method.

```
Function Composer_keyPressed( oEvt ) as Boolean
    If len( oComposerInputString ) = 1 Then
        oComposerInputString = oComposerInputString & oEvt.KeyChar
        Dim translation
        translation = GetTranslation( oComposerInputString )
        InsertString( translation, oComposerDocView, oComposerKeyHandler )
        oComposerInputString = ""
        ExitCompose
    Else
        oComposerInputString = oComposerInputString & oEvt.KeyChar
    End If
    Composer_KeyPressed = True
End Function
```

End Function

14.3. Listener Write-Up by Paolo Mantovani

The text in this next section was written by Paolo Mantovani. I (Andrew Pitonyak) made a few minor modifications. Thank you Paolo for taking the time. This is one of the best write-ups that I have seen on the topic. The document contained the following disclaimer when I received it and I include it here as well!

© 2003 Paolo Mantovani

This document is released under the Public Documentation License Version 1.0

A copy of the License is available at <http://www.openoffice.org/licenses/pdl.pdf>

14.3.1. The CreateUnoListener function

The OOO Basic runtime environment provides a function called CreateUnoListener, which requires two string arguments: a prefix and a fully qualified name of a listener interface.

```
oListener = CreateUnoListener( sPrefix , sInterfaceName )
```

This function is described very well in the OOO Basic help.

```
sListenerName = "com.sun.star.lang.XEventListener"  
oListener = CreateUnoListener("prefix_", sListenerName)  
MsgBox oListener.Dbgs_supportedInterfaces  
MsgBox oListener.Dbgs_methods
```

The com.sun.star.lang.XEventListener interface is the base interface for all listeners; it is, therefore, the simplest listener. XEventListener works only as a base interface for other listeners, so you should not use it explicitly, but for this example it is perfect.

14.3.2. Nice, but what does it do?

OOO Basic macros are able to call API methods and properties. Usually a macro makes many API calls. On the other hand, API's are usually not able to call OOO Basic routines. Consider the following example:

Listing 14.8: Simple event listener.

```
Sub Example_Listener  
    sListenerName = "com.sun.star.lang.XEventListener"  
    oListener = CreateUnoListener("prefix_", sListenerName)  
    Dim oArg As New com.sun.star.lang.EventObject  
    oListenerdisposing( oArgument )  
End Sub  
  
Sub prefix_disposing( vArgument )  
    MsgBox "Hi all!!"  
End Sub
```

When `Example_Listener` calls `oListener.disposing()`, `“prefix_disposing”` is called. In other words, the `CreateUnoListener` function creates a service able to call your OOo Basic routines.

You must create subroutines and functions with names that match the names of the listener's method, with the addition of the prefix specified when you call `CreateUnoListener`. For example, the call to `CreateUnoListener` passes the first argument as `“prefix_”` and the subroutine `“prefix_disposing”` starts with `“prefix_”`.

The documentation for the `com.sun.star.lang.XEventListener` interface says that the argument must be a `com.sun.star.lang.EventObject` structure.

14.3.3. How do I know what methods to create?

UNO requires a listener to call your macros. When you want to use a listener to intercept events, you require an UNO object able to speak to your listener. The UNO object that calls your listener is called a broadcaster. UNO broadcaster objects support methods to add and remove the appropriate listeners.

To create a listener object, pass the fully qualified name of the listener interface to the `CreateUnoListener` function. Retrieve the methods supported by the listener by accessing the `Dbg_methods` property (or check the IDL documentation for the listener interface). Finally, implement a basic routine for each method; even the disposing method.

Many UNO services provides methods to register and unregister listeners. For example, the `com.sun.star.OfficeDocumentView` service supports the `com.sun.star.view.XSelectionSupplier` interface. This is the broadcaster. This interface provides the following methods:

<code>addSelectionChangeListener</code>	Registers an event listener, which is called when the selection changes.
<code>removeSelectionChangeListener</code>	Unregisters an event listener which was registered with <code>addSelectionChangeListener</code> .

Both methods take a `SelectionChangeListener` as an argument (that is an UNO service that supports the `com.sun.star.view.XSelectionChangeListener` interface)

The broadcaster object adds one or more arguments in the callee. The first argument is an UNO structure, the following are depending on the interface definition. Check the IDL documentation of the listener interface you are using and see the method's detail. Often, the structure passed is a `com.sun.star.lang.EventObject`. However, all event structures must extend the `com.sun.star.lang.EventObject`, so they have at least the source element.

14.3.4. Example 1: `com.sun.star.view.XSelectionChangeListener`

Following is a complete implementation of the selection change listener. This listener can be used with all OpenOffice.org documents.

Listing 14.9: Selection change listener.

```
Option Explicit

Global oListener As Object
Global oDocView As Object

'run this macro to start event intercepting
Sub Example_SelectionChangeListener
    Dim sName$
    oDocView = ThisComponent.getCurrentController

    'create a listener to intercept the selection change event
    sName = "com.sun.star.view.XSelectionChangeListener"
    oListener = CreateUnoListener( "MyApp_", sName )

    ' register the listener to the document controller
    oDocView.addSelectionChangeListener(oListener)
End Sub

'run this macro to stop event intercepting
Sub Remove_Listener
    ' removes the listener
    oDocView.removeSelectionChangeListener(oListener)
End Sub

'all listeners must support this event
Sub MyApp_disposing(oEvent)
    msgbox "disposing the listener"
End Sub

Sub MyApp_selectionChanged(oEvent)
    Dim oCurrentSelection As Object
    'the source property of the event struct
    'gets a reference to the current selection
    oCurrentSelection = oEvent.source
    MsgBox oCurrentSelection.dbg_properties
End Sub
```

Notice that all listener's methods must be implemented in your basic program because if the caller service doesn't find the appropriate routines, a runtime error is raised.

Related API references:

<http://api.openoffice.org/docs/common/ref/com/sun/star/view/OfficeDocumentView.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/view/XSelectionSupplier.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/view/XSelectionChangeListener.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/lang/EventObject.html>

14.3.5. Example 2: com.sun.star.view.XPrintJobListener

An object that can be printed (let's say a document object), may support the `com.sun.star.view.XPrintJobBroadcaster` interface. This interface allows you to register (and unregister) a `com.sun.star.view.XPrintJobListener` to intercept events while printing. When you intercept printing events, you get a `com.sun.star.view.PrintJobEvent` structure. This structure has the usual property “source”; the source of this event is a Print Job, that is a service that describes the current printing process and must support the `com.sun.star.view.XPrintJob` interface.

Listing 14.10: *Print job listener.*

Option Explicit

Global oPrintJobListener As Object

'run this macro to start event intercepting

Sub Register_PrintJobListener

oPrintJobListener = _

CreateUnoListener("MyApp_", "com.sun.star.view.XPrintJobListener")

'this function is defined in the "Tools" Library

'writedbginfo oPrintJobListener

ThisComponent.addPrintJobListener(oPrintJobListener)

End Sub

'run this macro to stop event intercepting

Sub Unregister_PrintJobListener

ThisComponent.removePrintJobListener(oPrintJobListener)

End Sub

'all listeners must support this event

Sub MyApp_disposing(oEvent)

'nothing to do here

End sub

'this event is called several times

'during the printing process

Sub MyApp_printJobEvent(oEvent)

'the source of the printJob event is a PrintJob,

'that is a service that supports the `com.sun.star.view.XPrintJob`

'interface.

'This service describes the current printing process.

```

MsgBox oEvent.source.Dbg_methods

Select Case oEvent.State

    Case com.sun.star.view.PrintableState.JOB_STARTED
        MsgBox "printing (rendering the document) has begun"

    Case com.sun.star.view.PrintableState.JOB_COMPLETED
        sMsg = "printing (rendering the document) "
        sMsg = sMsg & "has finished, spooling has begun"
        MsgBox sMsg

    Case com.sun.star.view.PrintableState.JOB_SPOOLED
        sMsg = "spooling has finished successfully."
        sMsg = sMsg & " This is the only state that "
        sMsg = sMsg & "can be considered as 'success'"
        sMsg = sMsg & "for a print job."
        MsgBox sMsg

    Case com.sun.star.view.PrintableState.JOB_ABORTED
        sMsg = "printing was aborted (e.g., by the user) "
        sMsg = sMsg & "while either printing or spooling."
        MsgBox sMsg

    Case com.sun.star.view.PrintableState.JOB_FAILED
        sMsg = "printing ran into an error."
        MsgBox sMsg

    Case com.sun.star.view.PrintableState.JOB_SPOOLING_FAILED
        sMsg = "the document could be printed but not spooled."
        MsgBox sMsg

End Select
End sub

```

Related API references:

<http://api.openoffice.org/docs/common/ref/com/sun/star/document/OfficeDocument.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/view/XPrintJobBroadcaster.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/view/XPrintJobListener.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/view/PrintJobEvent.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/view/XPrintJob.html>

14.3.6. Example 3: com.sun.star.awt.XKeyHandler

Handlers are a special type of listener. As listeners they can intercept an event, but in addition an handler acts as event consumer, in other words, an handler can “eat” the event. In difference to listeners, methods in handlers must get a result (boolean): a True result tells to

broadcaster that the event is consumed from the handler, this causes that broadcaster will not send the event to the rest of the handlers.

The `com.sun.star.awt.XKeyHandler` allows to intercept key events into a document. The example shows a key handler that acts as consumer for some key pressed events (keys “t”, “a”, “b”, “u”) :

Listing 14.11: Key handler.

```
Option Explicit
Global oDocView
Global oKeyHandler

Sub RegisterKeyHandler
    oDocView = ThisComponent.getCurrentController
    oKeyHandler = _
        createUnoListener("MyApp_", "com.sun.star.awt.XKeyHandler")

    '   writedbginfo oKeyHandler

    oDocView.addKeyHandler(oKeyHandler)
End Sub

Sub UnregisterKeyHandler
    oDocView.removeKeyHandler(oKeyHandler)
End Sub

Sub MyApp_disposing(oEvt)
    'nothing to do here
End Sub

Function MyApp_KeyPressed(oEvt) As Boolean
    select case oEvt.KeyChar
        case "t", "a", "b", "u"
            MyApp_KeyPressed = True
            msgbox "key " & oEvt.KeyChar & " not allowed!"
        case else
            MyApp_KeyPressed = False
    end select
End Function

Function MyApp_KeyReleased(oEvt) As Boolean
    MyApp_KeyReleased = False
End Function
```

Related API references:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XUserInputInterception.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XExtendedToolkit.html>

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XKeyHandler.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/KeyEvent.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/Key.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/KeyFunction.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/KeyModifier.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/InputEvent.html>

14.3.6.1. Andrew has a little something to add

The question came up, how can I intercept F1 or Alt+z. The KeyChar for the function keys have an ASCII value of zero. Check the KeyCode for special characters. Although I do not see it mentioned elsewhere, you should also check the MODIFIERS property to make certain that the control, alt, and shift keys are NOT used. In the following example, I compare directly to the MOD2 key modifier even though this is only a flag. I only want to trap Alt+z, not Ctrl+Alt+z or any other variant.

Listing 14.12: Trapping special characters in a key handler.

```
If oEvt.KeyCode = com.sun.star.awt.Key.F1 AND oEvt.MODIFIERS = 0 Then
    MsgBox "Ha ha, I will NOT allow you to use F1 today!"
    MyApp_KeyPressed = True
    Exit Function
End If
If oEvt.KeyChar = "z" AND _
    oEvt.MODIFIERS = com.sun.star.awt.KeyModifier.MOD2 Then
    MsgBox "Ha ha, I will NOT allow you to use Alt+z today!"
    MyApp_KeyPressed = True
    Exit Function
End If
```

Unfortunately, this code can fail to find Alt+z. If the caps lock is pressed, then shift+z returns “z” rather than “Z” and the modifier will have both MOD2 and MOD1. The following example traps Alt+z even when caps lock is used.

Listing 14.13: Trapping special characters in a key handler.

```
If oEvt.KeyChar = "z" AND _
    ((oEvt.MODIFIERS AND com.sun.star.awt.KeyModifier.MOD2) <> 0) AND _
    ((oEvt.MODIFIERS AND com.sun.star.awt.KeyModifier.MOD1) = 0) Then
    MsgBox "Ha ha, I will NOT allow you to use Alt+z today!"
    MyApp_KeyPressed = True
    Exit Function
End If
```

14.3.6.2. A note about key modifiers (Ctrl and Alt keys)

The key event handler indicates if the Ctrl or the Alt key was pressed, it does not differentiate between the left or right key. Also, pressing the Alt key alone causes the key handler to be called, but not the Ctrl key. Philipp Lohmann from Sun provided insight (edited response):

By itself, a modifier is not intended to generate a key event in VCL, but rather, it will generate a specialized "KeyModChange" (modifier changed) event. This is not bound to the AWT so it is not available to an AWT customer. Moreover, key mod change is not dispatched on every modifier change, but only on key releases. The KeyModChange was developed to differentiate between the left and right shift key press and release, which switches the writing direction; and this is the reason for the behavior.

In Windows, the Alt key also functions as a menu key – a single key press moves the focus to the menu. This functionality is emulated on other operating systems. A side effect is that the ALT key is sent as a key event.

14.3.7. Example 4: com.sun.star.awt.XMouseClickedHandler

This handler allows to intercept mouse clicks in a document.

Listing 14.14: Complete mouse click handler.

```
Option Explicit

Global oDocView As Object
Global oMouseClickedHandler As Object

Sub RegisterMouseClickedHandler
    oDocView = ThisComponent.currentController
    oMouseClickedHandler = _
        createUnoListener("MyApp_", "com.sun.star.awt.XMouseClickedHandler")
    ' writedbginfo oMouseClickedHandler
    oDocView.addMouseClickedHandler(oMouseClickedHandler)
End Sub

Sub UnregisterMouseClickedHandler
    on error resume next
    oDocView.removeMouseClickedHandler(oMouseClickedHandler)
    on error goto 0
End Sub

Sub MyApp_disposing(oEvt)
End Sub

Function MyApp_mousePressed(oEvt) As Boolean
    MyApp_mousePressed = False
End Function

Function MyApp_mouseReleased(oEvt) As Boolean
Dim sMsg As String
    With oEvt
        sMsg = sMsg & "Modifiers = " & .Modifiers & Chr(10)
        sMsg = sMsg & "Buttons = " & .Buttons & Chr(10)
    End With
End Function
```

```

sMsg = sMsg & "X = " & .X & Chr(10)
sMsg = sMsg & "Y = " & .Y & Chr(10)
sMsg = sMsg & "ClickCount = " & .ClickCount & Chr(10)
sMsg = sMsg & "PopupTrigger = " & .PopupTrigger & Chr(10)
'sMsg = sMsg & .Source.dbg_Methods
End With

ThisComponent.text.string = sMsg

MyApp_mouseReleased = False
End Function

```

Related API references:

<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XUserInputInterception.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/XMouseClickedHandler.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/MouseEvent.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/MouseButton.html>
<http://api.openoffice.org/docs/common/ref/com/sun/star/awt/InputEvent.html>

14.3.8. Example 5: Manual binding of events

Normally, programming in OOO Basic, you don't need listeners, because you can manually bind an event to a macro. For example, from the dialog “Configure” (menu “Tools”=>“Configure.”), selecting the “events” tab you can bind application events or document events. Furthermore, many objects that you can insert into a document offer a properties-dialog with a Tab “Events”. Finally, OOO Basic dialogs and controls have this as well.

It's useful to notice that in the manual binding, the underlying mechanism is the same as listeners, therefore, you can add an event parameter to your macros to get additional information about the event.

To run the following example, open a new Writer document, add a Text Edit control and manually assign the macro to the key pressed event of the control. Notice that the macro name and the event name are arbitrary.

Listing 14.15: Manually adding an event handler.

```

Option Explicit

' This macro is manually assigned to the key-pressed
' event of a text-edit control in the document.
Sub MyTextEdit_KeyPressed(oEvt)
Dim sMsg As String

With oEvt
sMsg = sMsg & "Modifiers = " & .Modifiers & Chr(10)
sMsg = sMsg & "KeyCode = " & .KeyCode & Chr(10)
sMsg = sMsg & "KeyChar = " & .KeyChar & Chr(10)

```

```

    sMsg = sMsg & "KeyFunc = " & .KeyFunc & Chr(10)
    sMsg = sMsg & .Source.Dbg_supportedInterfaces
End With

msgbox sMsg
End Sub

```

14.4. What happened to my ActiveSheet listener?

Jim Thompson provided the code fragment to create an Event Listener for changes to the "ActiveSheet" property in the current controller. The listener notices when a new sheet is selected in the same document and performs sheet-specific processing. Activating and deactivating page preview (File | Page Preview), however, disables the listener so a change to a new sheet is no longer detected. The following code acts as the listener and does not demonstrate the solution:

```

REM Author: Jim Thompson
REM Email: jimthompson5802@aol.com

Global oActiveSheetListener as Object
Global CurrentWorksheetName as String
Global oListeningController as Object

Sub Workbook_Open()
    Rem Workbook_Open procedure assigned to "Document Open" event
    Rem Activate various listeners for events during processing
    Rem Turn-on worksheet activation listener
    Call WorksheetActivationListenerOn
End Sub

Sub WorksheetActivationListenerOn
    CurrentWorksheetName = ""
    oListeningController = ThisComponent.CurrentController
    oActiveSheetListener = _
        createUnoListener("ACTIVESHEET_", _
            "com.sun.star.beans.XPropertyChangeListener")
    oListeningController.addPropertyChangeListener("ActiveSheet", _
        oActiveSheetListener)
End Sub

Sub WorksheetActivationListenerOff
    oListeningController.removePropertyChangeListener("ActiveSheet", _
        oActiveSheetListener)
End Sub

Sub ACTIVESHEET_propertyChange(oEvent)
    REM call appropriate worksheet deactivation procedure
    Select Case CurrentWorksheetName

```

```

        Case "Example5"
            Call Example5Code.Worksheet_Deactivate
        End select

' msgbox "sheet changed:  OldSheet =" & _
' CurrentWorksheetName & ", NewSheet=" & _
' oEvent.Source.ActiveSheet.Name
REM call appropriate worksheet activation procedure
Select case oEvent.Source.ActiveSheet.Name
    Case "Example1"
        Call Example1Code.Worksheet_Activate
    Case "Example5"
        Call Example5Code.Worksheet_Activate
End Select
CurrentWorksheetName = oEvent.Source.ActiveSheet.Name
End Sub

Sub ACTIVESHEET_disposing(oEvent)
    msgbox "Disposing ACTIVESHEET"
End Sub

```

According to Mathias Bauer, a document's Controller object is changed if the view is changed. The solution is to register a FrameActionListener with the frame that contains the Controller. Every time a new component (in this case, the Controller) is attached to the frame, the frame sends a notification.

15. Impress

15.1. Slide background color

To change a background property, create a new background to replace the existing background. The background may be set by the master page.

Listing 15.1: Change Impress background.

```
Sub ChangeBackground
    Dim oDoc as Object
    oDoc = ThisComponent

    Dim oDrawPages as Object, oDrawPage as Object
    oDrawPages = oDoc.getDrawPages()
    oDrawPage = oDrawPages.getByIndex(0)

    Dim oBackground as Object
    oBackground = oDoc.createInstance("com.sun.star.drawing.Background")
    oBackground.FillColor = RGB(250,0,0)

    oDrawPage.Background = oBackground
End Sub
```

If a Background is present, you can probably change its properties; I did not check.

16. Language

I know for certain that this section is not complete, is based on a very early version of OOo (not that it changes much), and it contains a few errors. My book, however, is much more accurate, complete, and up-to-date; buy it!

16.1. Comments

It is always a good practice to liberally comment your code. What is clear today will not be clear tomorrow. The single quote character and REM both indicate that a comment is about to start. All text after this will be ignored.

```
REM This is a comment
REM And this is another comment
' And yet another comment
' I could do this all day long
Dim i As Integer REM i is used as in index variable in loops
Print i          REM This will print the value of i
```

16.2. Variables

16.2.1. Names

Variable names are limited to 255 characters and they must start with a standard alphabet character and they may contain numbers. The underscore and space characters are also valid characters. No distinction is made between upper and lower case characters. Variable names with spaces must be enclosed in brackets “[]”. This has been enhanced in newer version of OOo.

16.2.2. Declaration

It is considered good practice to declare your variables before you use them. The “Option Explicit” statement forces you to do this. This line must exist in your code before any other. If you do not use “Option Explicit”, then it is possible that misspelled variable names will come back to haunt you as bugs.

To declare a variable you use Dim. The syntax for Dim is as follows:

```
[ReDim]Dim Name1 [(start To end)] [As Type][, Name2 [(start To end)] [As Type][,...]]
```

This allows you to declare a number of variables at one time. *Name* is any variable or array name. The *start* and *end* values may be in the range of -32768 to 32767. This defines the number of elements (inclusively) so both *Name1(start)* and *Name1(end)* are valid values. If ReDim is used, then the *start* and *end* values may be numeric expressions. Valid values for type include Boolean, Currency, Date, Double, Integer, Long, Object, Single, String, and Variant.

Variant is the default type if no type is specified unless the DefBool, DefDate, DefDbL, DefInt, DefLng, DefObj, or DefVar commands are used. These commands allow you to specify the data type based on the first letter of a variable name.

String objects are limited to 64K characters.

Variant objects may contain all types. The type is determined by what it assigned.

Object variables must be followed by a subsequent set.

The following example program demonstrates the problems that can arise if you do not declare your variables. The undeclared variable “truc” will default to type Variant. Execute this macro and see which types uses for the non-declared variable :

```
Sub TestNonDeclare
    Print "1 : ", TypeName(truc), truc
    truc= "ab217"
    Print "2 : ", TypeName(truc), truc
    truc= true
    Print "3 : ", TypeName(truc), truc
    truc= 5=5 ' should be a Boolean
    Print "4 : ", TypeName(truc), truc
    truc= 123.456
    Print "5 : ", TypeName(truc), truc
    truc=123
    Print "6 : ", TypeName(truc), truc
    truc= 1217568942 ' could be a Long
    Print "7 : ", TypeName(truc), truc
    truc= 123123123123123.1234 'should be a Currency
    Print "8 : ", TypeName(truc), truc
End Sub
```

This is a strong argument to explicitly declare all variables.

Warning Each variable type must be declared or it will default to type Variant. “Dim a, b As Integer” is equivalent to “Dim a As Variant, b As Integer”.

```
Sub MultipleDeclaration
    Dim a, b As Integer
    Dim c As Long, d As String
    Dim e As Variant
    Print TypeName(a) REM Empty, Variant by default
    Print TypeName(b) REM Integer, Declared Integer
    Print TypeName(c) REM Long, Declared Long
    Print TypeName(d) REM String, Declared String
    Print TypeName(e) REM Empty, Variant as declared
    Print TypeName(g) REM Empty, Variant by default
End Sub
```

16.2.3. Evil Global Variables And Statics

Global variables are usually discouraged because they may be modified by any routine any time anywhere and it is difficult to know which methods modify which variables when they

are used. Because of this I started placing the modifier “evil” before the term “global variable” while I was teaching at The Ohio State University. I used it as a tool to remind my students that although there is a time and place for global variables, you should think before you use them.

A global must be declared outside of a procedure. You can then use the Public and Private keywords to specify if this variable is global to all modules or just this one. If neither Public nor Private is explicitly stated, then Private is assumed. The syntax is the same as the Dim and ReDim statements.

Although variables are passed by reference unless it is requested otherwise, global variables appear to pass by value. This caused at least one bug in my code.

Every time a procedure is called, the variables local to the procedure are recreated. If you declare the variable Static, it will retain its value. In the example below, the Worker Sub counts the number of times it has been called. Remember that Numeric variables are initialized to zero and Strings are initialized to the empty string.

```
Option Explicit
Public Author As String    REM Global to ALL Modules
Private PrivateOne$        REM Global to THIS Module only
Dim PrivateTwo$            REM Global to THIS Module only

Sub PublicPrivateTest
    Author = "Andrew Pitonyak"
    PrivateOne = "Hello"
    Worker()
    Worker()
End Sub

Sub Worker()
    Static Counter As Long REM retains its value between calls
    Counter = Counter + 1  REM count each time Worker is called
    Print "Counter = " + Counter
    Print "Author = " + Author
End Sub
```

16.2.4. Types

Abstractly speaking, OpenOffice.org Basic supports numeric, string, boolean, and object variable types. Objects are primarily used to refer to internals such as documents, tables, etc... With an object, you can use the objects corresponding methods and properties. Numeric types are initialized to zero and strings are initialized to the empty string “”.

If you need to know a variables type at runtime, TypeName function returns a string representation of the variable type. If you need to know a variables type at runtime, VarType function returns an integer corresponding to the variable type.

Keyword	Variable Type	VarType	Auto Type	Defxxx
Boolean	Boolean	11		DefBool
Currency	Currency with 4 Decimal places	6	@	
Date	Date	7		DefDate
Double	Double Floating Point	5	#	DefDbl
Integer	Integer	2	%	DefInt
Long	Long	3	&	DefLng
Object	Object	9		DefObj
Single	Single Floating Point	4	!	
String	String	8	\$	
Variant	Can contain all types specified by the definition	12		DefVar
Empty	Variable is not initialized	0		
Null	No valid data	1		

The ExampleTypes macro demonstrates the behavior.

Listing 16.1: Example variable types.

```

Sub ExampleTypes
    Dim b As Boolean          REM Boolean 11
    Dim c As Currency         REM Currency 6
    Dim t As Date             REM Date 7
    Dim d As Double           REM Double 5
    Dim i As Integer          REM Integer 2
    Dim l As Long             REM Long 3
    Dim o As Object           REM Object 9
    Dim f As Single           REM Single 4
    Dim s As String           REM String 8
    Dim v As Variant          REM Empty 0
    Dim n As Variant : n = NULL REM Null 1
    Dim x As Variant : x = f   REM Single 4

    Dim oData()
    Dim sName

    oData = Array(b, "b", c, "c", t, "t", d, "d", _
        i, "i", l, "l", o, "o", f, "f", s, "s", _
        v, "v", n, "n", x, "x")
    For i = LBound(oData()) To UBound(oData()) Step 2
        sName = oData(i+1)
        s = s & "TypeName(" & sName & ")=" & TypeName(oData(i)) & CHR$(10)
    Next

```

```

s = s & CHR$(10)
For i = LBound(oData()) To UBound(oData()) Step 2
    sName = oData(i+1)
    s = s & " VarType(" & sName & ")=" & VarType(oData(i)) & CHR$(10)
Next
MsgBox s
End Sub

```

16.2.4.1. Boolean Variables

Although boolean variables use the values “True” or “False,” they are internally represented by the integer values “-1” and “0” respectively. If you assign anything to a boolean and it does not precisely evaluate to “0”, then the “True” value is stored in the boolean. Typical uses are as follows:

```

Dim b as Boolean
b = True
b = False
b = (5 = 3) 'Set to False
Print b    'Prints 0
b = (5 < 7) 'Set to True
Print b    'Prints -1
b = 7      'Sets to True because 7 is not 0

```

16.2.4.2. Integer Variables

Integer variables are 16-bit numbers yielding a range of -32768 to 32767. Assigning a floating point number to an Integer is done by rounding to the nearest integer value. Postfixing a variable name with an “%” character causes it to become a Integer variable.

```

Sub AssignFloatToInteger
    Dim i1 As Integer, i2%
    Dim f2 As Double
    f2= 3.5
    i1= f2
    Print i1 REM 4
    f2= 3.49
    i1= f2
    Print i1 REM 3
End Sub

```

16.2.4.3. Long Integer Variables

Long integers variables are 32-bit numbers yielding a range of -2,147,483,648 to 2,147,483,647. Assigning a floating point number to a Long is done by rounding to the nearest integer value. Postfixing a variable name with an “&” character causes it to become a Long variable.

```

Dim Age&
Dim Dogs As Long

```

16.2.4.4. Currency Variables

Currency variables are 64-bit fixed four decimal and fifteen non-decimal numbers. This yields a range from -922,337,203,658,477.5808 to +922,337,203, 658,477.5807. Postfixing a variable name with an “@” character causes it to become a Currency variable.

```
Dim Income@
Dim Cost As Currency
```

16.2.4.5. Single Variables

Single variables are 32-bit numbers. The greatest magnitude is 3.402823×10^{38} . The smallest non-zero magnitude is 1.401298×10^{-45} . Postfixing a variable with the “!” character causes it to become a Single Variable.

```
Dim Weight!
Dim Height As Single
```

16.2.4.6. Double Variables

Double variables are 64-bit numbers. The greatest magnitude for a double variable is $1.79769313486232 \times 10^{308}$. The smallest non-zero magnitude for a double variable is $4.94065645841247 \times 10^{-324}$. Postfixing a variable with the “#” character causes it to become a Double Variable.

```
Dim Weight#
Dim Height As Double
```

16.2.4.7. String Variables

String variables can hold character strings with up to 65,535 characters. Each character is stored as the corresponding Unicode value. Postfixing a variable with the “\$” character causes it to become a String variable.

```
Dim FirstName$
Dim LastName As String
```

16.2.5. Object, Variant, Empty, and Null

The two special values Empty and Null are of interest when thinking of variables of type Object and Variant. The Empty value indicates that no value has been assigned to the variable. This is testable with the function `IsEmpty(var)`. The Null value indicates that no valid value is present. This is testable with the function `IsNull(var)`.

When a variable of type Object is first declared, it contains the value Null. When a variable of type Variant is first declared, it is Empty.

```
Sub ExampleObjVar
    Dim obj As Object, var As Variant
    Print IsNull(obj)      REM True
    Print IsEmpty(obj)    REM False
```

```

obj = CreateUnoService("com.sun.star.beans.Introspection")
Print IsNull(obj)      REM False
'obj = Null            REM Not valid...
'Print IsNull(obj)     REM True

Print IsNull(var)      REM False
Print IsEmpty(var)     REM True
var = obj
Print IsNull(var)      REM True
Print IsEmpty(var)     REM False
var = 1
Print IsNull(var)      REM False
Print IsEmpty(var)     REM False
'var = Empty           REM Not valid!
'Print IsEmpty(var)    REM True
End Sub

```

16.2.6. Should I Use Object Or Variant

When writing code that interacts with the UNO objects, you must decide which type to use. Although most examples use Object, page 132 of the Developer's Guide suggests otherwise.

Always use the type Variant to declare variables for UNO objects, not the type Object. The OpenOffice.org Basic type Object is tailored for pure OpenOffice.org Basic objects and not for UNO OpenOffice.org Basic objects. The Variant variables are best for UNO objects to avoid problems that can result from the OpenOffice.org Basic specific behavior of the type Object:

```

Dim oService1 ' Ok
oService1 = CreateUnoService( "com.sun.star.anywhere.Something" )
Dim oService2 as Object ' NOT recommended
oService2 = CreateUnoService( "com.sun.star.anywhere.SomethingElse" )

```

Andreas Bregas adds that for most cases both works. The Developer's Guide prefers variant because there are some odd situations where the usage of type object leads to an error due to the old object type semantics. But if a program uses type object and runs correctly with this there should be no problem.

16.2.7. Constants

OpenOffice.org Basic already knows the values “True”, “False”, and “PI”. You can define your own constants. Each constant may be defined once, and only once. Constants are not type defined, they are simply inserted as typed.

```

Const Gravity = 9.81

```

16.2.8. Arrays

An array allows you to store many different values in a single variable. By default, the first item in an array is at location 0. You may, however, specify the starting and ending values. Here are some examples

```
Dim a(5) As Integer      REM 6 elements from 0 to 5 inclusive
Dim b$(5 to 10) As String REM 6 elements from 5 to 10 inclusive
Dim c(-5 to 5) As String REM 11 elements from -5 to 5 inclusive
Dim d(5 To 10, 20 To 25) As Long
```

If you have a variant array and you want to fill it quickly, use the Array function. This returns a Variant array with the included data. This is how I build a list of data.

```
Sub ExampleArray
    Dim a(), i%
    a = Array(0, 1, 2)
    a = Array("Zero", 1, 2.0, Now)
    REM String, Integer, Double, Date
    For i = LBound(a()) To UBound(a())
        Print TypeName(a(i))
    Next
End Sub
```

16.2.8.1. Option Base

You may change the default lower bound of an array to start at 1 rather than zero. This must be done before any other executable statement in the program.

Syntax: Option Base { 0 | 1 }

16.2.8.2. LBound(arrayname[,Dimension])

Returns the lower bound of an array. The optional second parameter which is the dimension of the array for which you desire a lower bound is 1 based (not zero based).

```
LBound(a()) REM 0
LBound(b$()) REM 5
LBound(c()) REM -5
LBound(d()) REM 5
LBound(d(), 1) REM 5
LBound(d(), 2) REM 20
```

16.2.8.3. UBound(arrayname[,Dimension])

Returns the upper bound of an array. The optional second parameter which is the dimension of the array for which you desire an upper bound is 1 based (not zero based).

```
UBound(a()) REM 5
UBound(b$()) REM 10
UBound(c()) REM 5
```

```

UBound(d())      REM 10
UBound(d(), 1)   REM 10
UBound(d(), 2)   REM 25

```

16.2.8.4. Is This Array Defined

If an array is really an empty list, then the lower bound of the array will be larger than the upper bound of the array.

16.2.9. DimArray, Changing The Dimension

The DimArray function is used to set or change the number of dimensions of a Variant array. DimArray(2, 2, 4) is the same as DIM a(2, 2, 4).

```

Sub ExampleDimArray
    Dim a(), i%
    a = Array(0, 1, 2)
    Print "" & LBound(a()) & " " & UBound(a())      REM 0 2
    a = DimArray()
    ' Empty array
    i = 4
    a = DimArray(3, i)
    Print "" & LBound(a(),1) & " " & UBound(a(),1)   REM 0, 3
    Print "" & LBound(a(),2) & " " & UBound(a(),2)   REM 0, 4
End Sub

```

16.2.10. ReDim, Changing The Number Of Elements

The ReDim statement is used to change the size of an array.

```

Dim e() As Integer      REM I did not specify the size
ReDim e(5) As Integer    REM 0 to 5 is valid
ReDim e(10) As Integer   REM 0 to 10 is valid

```

The Preserve keyword may be used with the ReDim statement to preserve the contents of the array when it is re-dimensioned.

```

Sub ReDimExample
    Dim a(5) As Integer
    Dim b()
    Dim c() As Integer
    a(0) = 0
    a(1) = 1
    a(2) = 2
    a(3) = 3
    a(4) = 4
    a(5) = 5
    REM a is dimensioned from 0 to 5 where a(i) = i
    PrintArray("a at start", a())
    REM a is dimensioned from 1 to 3 where a(i) = i

```

```

ReDim Preserve a(1 To 3) As Integer
PrintArray("a after ReDim", a())
REM Array() returns a variant type
REM b is dimensioned from 0 to 9 where b(i) = i+1
b = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
PrintArray("b at initial assignment", b())
REM b is dimensioned from 1 to 3 where b(i) = i+1
ReDim Preserve b(1 To 3)
PrintArray("b after ReDim", b())
REM The following is NOT valid
REM because the array is already dimensioned
REM to a different size
REM a = Array(0, 1, 2, 3, 4, 5)

REM c is dimensioned from 0 to 5 where a(i) = i
REM If a "ReDim" had been done on c, then this would NOT work
c = Array(0, 1, 2, 3, 4, 5)
PrintArray("c, of type Integer after assignment", c())
REM Ironically, this allowed but c will contain no data!
ReDim Preserve c(1 To 3) As Integer
PrintArray("c after ReDim", c())
End Sub

Sub PrintArray (lead$, a() As Variant)
    Dim i%, s$
    s$ = lead$ + Chr(13) + LBound(a()) + " to " + _
        UBound(a()) + ":" + Chr(13)
    For i% = LBound(a()) To UBound(a())
        s$ = s$ + a(i%) + " "
    Next
    MsgBox s$
End Sub

```

The Array function mentioned above only works to create a Variant array. To initialize an array of a known type, you can use the following method.

```

Sub ExampleSetIntArray
    Dim iA() As Integer
    SetIntArray(iA, Array(9, 8, 7, 6))
    PrintArray("", iA)
End Sub

Sub SetIntArray(iArray() As Integer, v() As Variant)
    Dim i As Long
    ReDim iArray(LBound(v()) To UBound(v())) As Integer
    For i = LBound(v) To UBound(v)
        iArray(i) = v(i)
    Next
End Sub

```


16.2.11. Testing Objects

To determine the type of a variable, you can use the boolean functions `IsArray`, `IsDate`, `IsEmpty`, `IsMissing`, `IsNull`, `IsNumeric`, `IsObject`, and `IsUnoStruct`. The `IsArray` function returns true if the parameter is an array. The `IsDate` function returns true if it is possible to convert the object into a `Date`. A string with a properly formatted date will therefore return true for the `IsDate` function. The `IsEmpty` method is used to test if a Variant type object has been initialized. The `IsMissing` function indicates if an Optional parameter is missing. The `IsNull` method tests whether a Variant contains the special Null value, indicating that the variable contains no data. `IsNumeric` is used to test if a string contains numeric values. The `IsUnoStruct` function takes the string name of an UNO structure and returns true only if it is a valid name.

16.2.12. Comparison Operators

Conditionals generally work as expected but they do not perform short circuit evaluation. The following standard conditional operators are used

<i>Symbol</i>	<i>Meaning</i>
=	Equal To
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
<>	Not Equal To
Is	Are these the same Object

The AND operator performs a logical operation on Boolean types and bitwise operations on numeric types. The OR operator performs a logical operation on Boolean types and bitwise operations on numeric types. The XOR operator performs a logical operation on Boolean types and bitwise operations on numeric types. Remember that this is “Exclusive OR”. The NOT operator performs a logical operation on Boolean types and bitwise operations on numeric types. A simple test shows that the standard precedence roles exist, namely that AND has greater precedence than the OR operators.

?? On 7/28/03, Andrew has decided that this is incorrect! Oops on me!

```
Option Explicit
Sub ConditionTest
    Dim msg As String
    msg = "AND has "
    msg = msg & IIf(False OR True AND False, "equal", "greater")
    msg = msg & " precedence than OR" & Chr(13) & "OR does "
    msg = msg + IIf(True XOR True OR True, "", "not ")
End Sub
```

```

msg = msg + "have greater precedence than XOR" + Chr(13)
msg = msg & "XOR does "
msg = msg + IIF(True OR True XOR True, "", "not ")
msg = msg + "have greater precedence than OR"
MsgBox msg
End Sub

```

16.3. Functions and SubProcedures

A Function is a Sub procedure that can return a value. This allows it to be used in an expression. Subs and Functions start as follows:

Start Syntax: Function FuncName[(Var1 [As Type][, Var2 [As Type][,...]])] [As Type]

Start Syntax: Sub SubName[(Var1 [As Type][, Var2 [As Type][,...]])]

Functions declare a return value type because they return a value. To assign the return value, use a statement of the form “FuncName = return_value”. Although you may perform this assignment multiple times, it is the last one that is returned.

To immediately leave the procedure use an appropriate Exit statement.

16.3.1. Optional Parameters

A parameter may be declared as optional using the Optional keyword. The IsMissing method is then used to determine if a parameter was passed.

```

Sub testOptionalParameters()
    Print TestOpt()      REM MMM
    Print TestOpt(,)     REM MMM
    Print TestOpt(,,)    REM MMM
    Print TestOpt(1)     REM 1MM
    Print TestOpt(1,)    REM 1MM
    Print TestOpt(1,,)   REM 1MM
    Print TestOpt(1,2)   REM 12M
    Print TestOpt(1,2,)  REM 12M
    Print TestOpt(1,2,3) REM 123
    Print TestOpt(1,,3)  REM 1M3
    Print TestOpt(,2,3)  REM M23
    Print TestOpt(,,3)   REM MM3
    Print TestOptI()     REM MMM
    Print TestOptI(,)    REM 488MM (Error)
    Print TestOptI(,,)   REM 488488M (Error)
    Print TestOptI(1)    REM 1MM
    Print TestOptI(1,)   REM 1MM
    Print TestOptI(1,,)  REM 1488M (Error)
    Print TestOptI(1,2)  REM 12M
    Print TestOptI(1,2,) REM 12M
    Print TestOptI(1,2,3) REM 123
    Print TestOptI(1,,3) REM 14883 (Error)

```

```

Print TestOptI(,2,3) REM 48823 (Error)
Print TestOptI(,,3) REM 4884883 (Error)
End Sub
Function TestOpt(Optional v1 As Variant, Optional v2 As Variant, Optional v3 As Variant) As String
    Dim s As String
    s = "" & IIF(IsMissing(v1), "M", Str(v1))
    s = s & IIF(IsMissing(v2), "M", Str(v2))
    s = s & IIF(IsMissing(v3), "M", Str(v3))
    TestOpt = s
End Function
Function TestOptI(Optional i1 As Integer, Optional i2 As Integer, Optional i3 As Integer) As String
    Dim s As String
    s = "" & IIF(IsMissing(i1), "M", Str(i1))
    s = s & IIF(IsMissing(i2), "M", Str(i2))
    s = s & IIF(IsMissing(i3), "M", Str(i3))
    TestOptI = s
End Function

```

Warning As of version 1.0.3.1, IsMissing will fail with Optional parameters if the type is not Variant and the missing optional parameter is represented by two consecutive commas. I first investigated this behavior after speaking with Christian Anderson [ca@ofs.no]. This is issue 11678 in issuezilla.

16.3.2. Parameters By Reference Or Value

If a variable is passed by value, I can change the parameter in the called procedure and the original variable will not change. If I pass a reference instead, then if I change the parameter I also change the original variable. The default behavior is to pass by reference. To pass by value, use the ByVal keyword before the parameter declaration. If the parameter is a constant such as “4” and you modify it in the called procedure it may, or may not, really change. According to Andreas Bregas (ab@openoffice.org) this is a bug so I have opened an issue in issuezilla (http://www.openoffice.org/project/www/issues/show_bug.cgi?id=12272).

```

Option Explicit
Sub LoopForever
    Dim l As Long
    l = 4
    LoopWorker(l)
    Print "Passed l by value and it is still " + l
    LoopForeverWorker(l)
    ' l is now 1 so this will print 1.
    Print "Passed l by reference and it now is " + l
    ' This will loop forever because 4 is a constant
    ' and you can NOT change it.
    Print "Passing a constant parameter by reference, this will be fun"
    Print LoopForeverWorker(4)

```

```

End Sub

Sub LoopWorker(ByVal n As Long)
    Do While n > 1
        Print n
        n = n - 1
    Loop
End Sub

Sub LoopForeverWorker(n As Long)
    Do While n > 1
        ' This is fun when n is a constant.
        Print n
        n = n - 1
    Loop
End Sub

```

16.3.3. Recursion

Your functions can be recursive as of 1.1.1. Different versions of OOo on different operating systems support different recursion levels, so be careful.

```

Option Explicit
Sub DoFact
    Print "Recursive = " + RecursiveFactorial(4)
    Print "Normal Factorial = " + Factorial(4)
End Sub

Function Factorial(n As Long) As Long
    Dim answer As Long
    Dim i As Long
    i = n
    answer = 1
    Do While i > 1
        answer = answer * i
        i = i - 1
    Loop
    Factorial = answer
End Function

' This will fail because you can not use recursion
Function RecursiveFactorial(n As Long) As Long
    If n > 1 Then
        RecursiveFactorial = n * RecursiveFactorial(n-1)
    Else
        RecursiveFactorial = 1
    End If
End Function

```

16.4. Flow Control

16.4.1. If Then Else

The If construct is used to execute a block of code based on an expression. Although you can use GoTo or GoSub to jump out of an If block, you can not jump into an If block.

Syntax:

```
If condition=true Then
    Statementblock
[ElseIf condition=true Then]
    Statementblock
[Else]
    Statementblock
End If
```

Syntax:

```
If condition=true Then Statement
```

Example:

```
If x < 0 Then
    MsgBox "The number is negative"
ElseIf x > 0 Then
    MsgBox "The number is positive"
Else
    MsgBox "The number is zero"
End If
```

16.4.2. IIf

The IIf construct is used to return an expression based on a condition. This is similar to the “?” syntax in C.

Syntax: IIf (Condition, TrueExpression, FalseExpression)

This is very similar to the following code:

```
If (Condition) Then
    object = TrueExpression
Else
    object = FalseExpression
End If
max_age = IIf(johns_age > bills_age, johns_age, bills_age)
```

16.4.3. Choose

The choose statement allows selecting from a list of values based on an index.

Syntax: Choose (Index, Selection1[, Selection2, ... [,Selection_n]])

If the index is 1, then the first item is returned. If the index is 2, then the second item is returned. You can figure out the rest!

16.4.4. For....Next

The on-line help contains an excellent complete description, read it.

Repeat a block of statements a specified number of times.

Syntax:

```
For counter=start To end [Step step]
    statement block
[Exit For]
statement block
Next [counter]
```

The numeric “counter” is initially assigned the “start” value. If the “step” value is not given, then the counter is incremented by one until it passes the “end” value. If the “step” value is given, then the “step” is added to the “start” value until it passes the “end” value. The statement blocks are executed once for each increment.

The “counter” is optional on the “Next” statement, and it automatically refers to the most recent “For” statement.

You may prematurely leave a for statement by using the “Exit For” statement. This will exit the most recent “For” statement.

Example:

The following example fills an array with random integers. The array is then sorted using two nested loops .

```
Sub ForNextExampleSort
    Dim iEntry(10) As Integer
    Dim iCount As Integer, iCount2 As Integer, iTemp As Integer
    Dim bSomethingChanged As Boolean

    ' Fill the array with the integers between -10 and 10
    For iCount = LBound(iEntry()) To UBound(iEntry())
        iEntry(iCount) = Int((20 * Rnd) -10)
    Next iCount

    ' Sort the array
    For iCount = LBound(iEntry()) To UBound(iEntry())

        'Assume that the array is sorted
        bSomethingChanged = False
        For iCount2 = iCount + 1 To UBound(iEntry())
            If iEntry(iCount) > iEntry(iCount2) Then
                iTemp = iEntry(iCount)
```

```

        iEntry(iCount) = iEntry(iCount2)
        iEntry(iCount2) = iTemp
        bSomethingChanged = True
    End If
Next iCount2
'If the array is already sorted then stop looping!
If Not bSomethingChanged Then Exit For
Next iCount
For iCount = 1 To 10
    Print iEntry(iCount)
Next iCount
End Sub

```

16.4.5. Do Loop

The on-line help contains an excellent complete description, read it.

The Loop construct has a few different forms and is used to continue executing a block of code while a condition is true. The most common form checks the condition before the loop starts and as long as the condition is true will repeatedly execute the block of code. If the condition is false, then the loop will never be executed.

```

Do While condition
    Block
Loop

```

A similar but much less common form checks the condition before the loop starts and as long as the condition is false will repeatedly execute the block of code. If the condition evaluates to true immediately, then the loop is never run.

```

Do Until condition
    Block
Loop

```

You may also place the check at the end of the loop in which case the block of code will always be executed at least once. To always execute the loop at least once and then continue as long as the condition is true, use the following construct:

```

Do
    Block
Loop While condition

```

To always execute the loop at least once and then continue as long as the condition is false, use the following construct:

```

Do
    Block
Loop Until condition

```

In a “Do Loop”, you can force an immediate exit from the loop with the “Exit Do” statement.

16.4.6. Select Case

The Select Case statement is similar to the “case” and “switch” statements in other languages. This mimics multiple “Else If” blocks in an “If” statement. A single condition expression is specified and this is compared against multiple values for a match as follows:

```
Select Case condition_expression
  Case case_expression1
    StatementBlock1
  Case case_expression2
    StatementBlock2
  Case Else
    StatementBlock3
End Select
```

The condition_expression is the expression that will be compared in each Case statement. I am not aware of any particular data type limitations other than the condition type must be compatible with the expression type. The first statement block to match is executed. If no condition matches, then the optional Case Else will match.

16.4.6.1. Case Expressions

A case expression is usually a constant such as “Case 4” or “Case "hello"”. Multiple values may be specified by separating them with commas: “Case 3, 5, 7”. If you want to check a range of values, there is a “To” keyword “Case 5 To 10”. Open ended ranges may be checked as “Case < 10” or with the “Is” keyword “Case Is < 10”.

Warning	Be careful when using a range in a Case statement. The on-line help has repeatedly contained incorrect examples such as “Case i > 2 AND i < 10”. This is difficult to understand and code correctly.

16.4.6.2. Incorrect Simple Example

I have seen many incorrect examples so I will spend some time to show a few examples of things that will not work. I will start with a very simple example. Consider the following:

Correct	Correct	Incorrect
<pre>i = 2 Select Case i Case 2</pre>	<pre>i = 2 Select Case i Case Is = 2</pre>	<pre>i = 2 Select Case i Case i = 2</pre>

The bad example fails because “Case i = 2” reduces to “Case Is = (i = 2)”. The expression (i=2) evaluates to True, which is -1 so this is evaluated as “Case Is = -1” in this example.

If you understand this simple incorrect example, then you are ready for difficult examples.

16.4.6.3. Incorrect Range Example

The following incorrect example was in the on-line help.


```
Case Is > 8 AND iVar < 11
```

This does not work because it is evaluated as :

```
Case Is > (8 AND (iVar < 11))
```

The expression (iVar<11) is evaluated as true or false. Remember that true= -1 and false=0. The AND is then bitwise applied between 8 and -1 (true) or 0 (false), which results in either 8 or 0. This expression reduces to one of two expressions.

If iVar is less than 11 :

```
Case Is > 8
```

If iVar is greater or equal to 11 :

```
Case Is > 0
```

16.4.6.4. Incorrect Range Example

I have also seen this incorrect example in print.

```
Case i > 2 AND i < 10
```

This does not work because it is evaluated as

```
Case Is = (i > 2 AND i < 10)
```

16.4.6.5. Ranges, The Correct Way

The statement

```
Case Expression
```

is probably correct if it can be written

```
Case Is = (Expression)
```

My initial solution follows:

```
Case Iif(Boolean Expression, i, i+1)
```

I was proud of myself until I was given the following brilliant solution by Bernard Marcelly:

```
Case i XOR NOT (Boolean Expression)
```

After my initial confusion, I realized how brilliant this really is. Do not be tempted to simplify this to the obvious reduction of “i AND ()” because it will fail if i is 0. I made that mistake.

```
Sub DemoSelectCase
Dim i%
i = Int((15 * Rnd) -2)
Select Case i%
Case 1 To 5
Print "Number from 1 to 5"
Case 6, 7, 8
```

```

    Print "Number from 6 to 8"
Case IIf(i > 8 AND i < 11, i, i+1)
    Print "Greater than 8"
Case i% XOR NOT(i% > 8 AND i% < 11 )
    Print i%, "Number is 9 or 10"
Case Else
    Print "Out of range 1 to 10"
End Select
End Sub

```

16.4.7. While...Wend

There is nothing special about the While...Wend construct, it has the following form:

```

While Condition
    Code
Wend

```

Tip

This construct has limitations that do not exist in the Do While...Loop construct and offers no particular benefits. You can not use the Exit construct, nor can you exit with a GoTo.

16.4.8. GoSub

The GoSub statement causes execution to jump to a defined subroutine label in the current subroutine. You can not jump outside the current subroutine. When the Return statement is reached, execution will continue from the point of the original call. If a Return statement is encountered and no previous GoSub was made, an error occurs. In other words, Return is not a substitute for Exit Sub or Exit Function. It is generally assumed that the use of functions and subroutines produce more understandable code than the use of GoSub and GoTo.

```

Option Explicit
Sub ExampleGoSub
    Dim i As Integer
    GoSub Line2
    GoSub Line1
    MsgBox "i = " + i, 0, "GoSub Example"
    Exit Sub
Line1:
    i = i + 1
    Return
Line2:
    i = 1
    Return
End Sub

```

Tip

GoSub is a persistent remnant from old dialects, retained for compatibility. GoSub is strongly discouraged because it tends to produce

unreadable code. Subs or Functions are preferable.

16.4.9. GoTo

The GoTo statement causes execution to jump to a defined label in the current subroutine. You can not jump outside the current subroutine.

```
Sub ExampleGoTo
    Dim i As Integer
    GoTo Line2
Line1:
    i = i + 1
    GoTo TheEnd
Line2:
    i = 1
    GoTo Line1
TheEnd:
    MsgBox "i = " + i, 0, "GoTo Example"
End Sub
```

Tip

GoTo is a persistent remnant from old dialects, retained for compatibility. GoTo is strongly discouraged because it tends to produce unreadable code. Subs or Functions are preferable.

16.4.10. On GoTo

Syntax: On N GoSub Label1[, Label2[, Label3[,...]]]

Syntax: On N GoTo Label1[, Label2[, Label3[,...]]]

This causes the execution to branch a label based on the the numeric expression N . If ($N=0$) then no branching occurs. The numeric expression N must be in the range of 0 and 255. This is similar to the “computed goto,” “case,” and “switch,” statements in other languages. Do not try to jump out of the current subroutine or function.

```
Option Explicit
Sub ExampleOnGoTo
    Dim i As Integer
    Dim s As String
    i = 1
    On i+1 GoSub Sub1, Sub2
    s = s & Chr(13)
    On i GoTo Line1, Line2
    REM The exit causes us to exit if we do not continue execution
    Exit Sub
Sub1:
    s = s & "In Sub 1" : Return
Sub2:
    s = s & "In Sub 2" : Return
```

```

Line1:
    s = s & "At Label 1" : GoTo TheEnd
Line2:
    s = s & "At Label 2"
TheEnd:
    MsgBox s, 0, "On GoTo Example"
End Sub

```

16.4.11. Exit

The Exit statement is used to exit a Do Loop, For Next, Function, or a Sub. Attempting to exit a non-enclosing construct will cause an error. For example, you can not exit a For loop if you are not in one. The forms are as follows:

Exit DO Continue execution following Loop statement.

Exit For Continue execution following the Next statement.

Exit Function Immediately exit the current function.

Exit Sub Immediately exit the current Sub.

```

Option Explicit
Sub ExitExample
    Dim a%(100)
    Dim i%
    REM Fill the array with 100, 99, 98, ..., 0
    For i = LBound(a()) To UBound(a())
        a(i) = 100 - i
    Next i
    Print SearchIntegerArray(a(), 0 )
    Print SearchIntegerArray(a(), 10 )
    Print SearchIntegerArray(a(), 100)
    Print SearchIntegerArray(a(), 200)
End Sub

Function SearchIntegerArray( list(), num%) As Integer
    Dim i As Integer
    SearchIntegerArray = -1
    For i = LBound(list) To UBound(list)
        If list(i) = num Then
            SearchIntegerArray = i
            Exit For
        End If
    Next i
End Function

```

16.4.12. Error Handling

Your macro may encounter several types of errors. Some errors you should check for, such as missing files, and some you should simply trap. To trap errors in macros, use the “On Error” statement.

On [Local] {Error GoTo Labelname | GoTo 0 | Resume Next}

On Error allows you to specify how errors should be handled including the ability to setup your own error handler. If “Local” is used, then this defines an error handling routine local to the containing subroutine or function. If “Local” is not used then the error handling affects the entire module.

Tip

A procedure may contain several On Error statements. Each On Error may treat errors differently. (The on-line help incorrectly states that error handling must occur at start of the procedure).

16.4.12.1. Specify How To Handle The Error

To ignore all errors, use “On Error Resume Next”. When an error occurs, the statement that caused the error will be skipped and the next statement will be executed.

To specify your own error handler, use “On Error GoTo Label”. To define a Label in OOo Basic, type some text on a line by itself and follow it with a colon. Line labels must be unique. When an error occurs, execution will be transferred to the label.

After specifying a method of handling errors, you can undo this using “On Error GoTo 0”. The next time an error occurs, your handler will not be invoked. This is not the same as “On Error Resume Next”, it means that the next error will be handled in the default manner (stopping macro execution with an error message).

16.4.12.2. Write The Error Handler

When an error occurs and execution is transferred to your error handler, there are some functions that help you determine what happened and where.

Error([num]) : Returns the error message as a string. You may optionally provide an error number to retrieve the error message for a specific error number. The message text is in the localized language.

Err() : Returns the error number of the last error.

Erl() : Returns the line number where the last error occurred.

After the error has been handled, you must decide how to proceed.

Do nothing and allow execution to proceed.

Exit the subroutine or function using “Exit Sub” or “Exit Function”.

Use “Resume” to execute the same line again. Be careful with this, if you have not corrected the error, it is possible that you will be stuck in an infinite loop.

```
Sub ExampleResume
    Dim x%, y%
    x = 4 : y = 0
    On Local Error Goto oopsy
    x = x / y
    Print x
    Exit Sub
oopsy:
    y = 2
    Resume
End Sub
```

Use “Resume Next” to resume macro execution on the line immediately following error.

```
Sub ExampleResumeNext
    Dim x%, y%
    x = 4 : y = 0
    On Local Error Goto oopsy
    x = x / y
    Print x
    Exit Sub
oopsy:
    x = 7
    Resume Next
End Sub
```

Use “Resume Label:” to continue execution at a specified label.

```
Sub ExampleResumeLabel
    Dim x%, y%
    x = 4 : y = 0
    On Local Error Goto oopsy
    x = x / y
GoHere:
    Print x
    Exit Sub
oopsy:
    x = 7
    Resume GoHere:
End Sub
```

16.4.12.3. An Example

The following example demonstrates many excellent error handling examples.

```
'*****
'Author:   Bernard Marcellly
'email:    marcellly@club-internet.fr
```

```

Sub ErrorHandlingExample
    Dim v1 As Double
    Dim v2 As Double
    Dim v0 As Double

    On Error GoTo TreatError1
    v0 = 0 : v1= 45 : v2= -123 ' initialize to some value
    v2= v1 / v0 ' divide by zero => error
    Print "Result1:", v2

    On Error Goto TreatError2 ' change error handler
    v2= 456 ' initialize to some value
    v2= v1 / v0 ' divide by zero = error !
    Print "Result2:", v2 ' will not be executed

Label2:
    Print "Result3:", v2 ' jumped to by error handling

    On Error Resume Next ' ignore any error
    v2= 963 ' initialize to some value
    v2= v1 / v0 ' divide by zero = error !
    Print "Result4:", v2 ' will be executed

    On Error Goto 0 ' disable current error handler
    REM standard error handling is now active
    v2= 147 ' initialize to some value
    v2= v1 / v0 ' divide by zero = error !
    Print "Result5:", v2 ' will not be executed
    Exit Sub

TreatError1:
    Print "TreatError1 : ", error
    v2= 0
    Resume Next ' continue after statement on error

TreatError2:
    Print "TreatError2 : line ", erl, "error number", err
    v2= 123456789
    Resume Label2
End Sub

```

16.5. Miscellaneous

This section contains bits and pieces of things that I only know because I have seen examples but have not found examples for. ???

Many statements may exist on the same line if they are separated by a “:” (colon).

For single line statements, the “If Then” construct does not require the closing “End If”.

```
Sub SimpleIf
  If 4 = 4 Then Print "4 = 4" : Print " Hello you" REM This prints
  If 3 = 2 Then Print "3 = 2" REM This does not
End Sub
```

Libraries, dialogs, IDE, Import and Export of Macros.

With object ... End With

Copying an object will simply copy the reference. Copying a structure makes a new copy. See EqualUnoObjects for an example.?? This can cause a problem and then the object will have to be copied back!

17. Compatibility With Visual BASIC

This chapter was started for my published book. The chapter was cut so I never finished the chapter. I have not updated this to include things such as “compatibility mode”, which works as advertised.

The language structures in OpenOffice.org BASIC are very similar to those used in Visual BASIC. The methods used for accessing the underlying documents, however, are vastly different and have essentially no compatibility with each other. Entire books have been written dealing with the differences between Visual BASIC 6 (VB6), Visual BASIC.NET (VB.NET), and Visual BASIC for Applications (VBA). This chapter is only an overview of issues concerning compatibility between the OpenOffice.org BASIC and Visual BASIC. I use VBA, VB6 and VB.NET to refer to the specific versions and VB to generically refer to either or both versions.

To convert VB macros that do not access the underlying documents, my first step is to bring them into OOo and fix the syntax errors. The second step is to remove the errors introduced due to differences in behavior. Thorough testing is required to avoid subtle problems. Significant code changes are required to convert the sections that access the underlying document structures.

VBA is the variant used by Microsoft Office. VB.NET was released after VBA so it is possible that a later version of VBA will follow in the direction of VB.NET. Some of the keywords supported by OOo BASIC and deprecated when moving from VB6 and VBA to VB.NET are in Table 17.1.

Table 1. OOo BASIC Keywords deprecated when moving to VB to VB.NET.

Table 17.1: OOo BASIC Keywords deprecated when moving to VB to VB.NET.

Word	Word	Word	Word	Word	Word	Word
Atn	Currency	DefBool	DefDate	DefDbl	DefInt	DefLng
DefObj	DefVar	Empty	Eqv	GoSub	Imp	IsEmpty
IsMissing	IsNull	IsObject	Let	Line	LSet	MsgBox
Now	Null	On? GoSub	On?GoTo	Option Base	Private	Rnd
RSet	Set	Sgn	Sqr	Wend		

17.1. Data types

Table 17.2: VB.NET uses different names for some numerical functions.

OOo BASIC	VB	VB.NET	Return Value
Byte	Byte	Byte	0 through 255, OOo BASIC uses the CByte function to create one.

OOo BASIC	VB	VB.NET	Return Value
Integer	Integer	Short	–32,768 through 32,767
Long	Long	Integer	–2,147,483,648 through 2,147,483,647
		Long	–9,223,372,036,854,775,808 through 9,223,372,036,854,775,807
Currency	Currency		+/–922,337,203,658,477.5808 fixed point number. Same as a VB.NET Long. The number is multiplied internally by 10,000, thereby eliminating the need for the decimal point, and then stored in binary form as an integer. This prevents the rounding errors that can occur when decimal fractions are stored as binary floating-point numbers.
	Decimal	Decimal	+/–79,228,162,514,264,337,593,543,950,335 for numbers with no decimal places. The smallest possible non-zero number is 0.000000000000000000000001.
String	String	String	OOo BASIC has a limit of 65,535 Unicode characters, VB may contain approximately 2 billion Unicode characters.

17.2. Variables

- VB.NET does not support Variant variables so variables with no declared type default to type Object. Other VB versions are compatible with OOo BASIC.
- VB.NET replaced the Currency data type with the Decimal data type. Other VB versions are compatible with OOo BASIC.
- VB.NET does not support the keywords DefBool, DefDate, DefDbl, DefInt, DefLng, DefObj, and DefVar. Other VB versions are compatible with OOo BASIC.
- VB.NET does not support the keywords Set and Let.
- VB.NET does not support the keywords NULL or Empty.
- Although OOo BASIC supports the syntax for fixed length strings (Dim s As String * 100), they are still variable length strings. All VB versions support fixed length strings.
- OOo BASIC strings are limited to 65,535 Unicode characters. In VB, variable length strings may contain approximately 2 billion Unicode characters. Loss of data is a very real problem in OOo BASIC compared to VB.

17.3. Arrays

- VB.NET deprecated Option Base. Other VB are not totally compatible with OOo BASIC.

- VBA and VB6 support Option Base but but, unlike OOo BASIC, they do not change the upper bound of an array, only the lower bound.
- VB only supports ReDim Preserve when changing the index size on the last dimension. OOo BASIC supports changing any dimension of a multi-dimension array.
- VB only supports ReDim to change the dimension of an array whose dimensions are not explicitly declared. OOo BASIC is more flexible.

17.4. Subroutine and Function Constructs

- VB allows a Sub or Function to be preceded by optional scoping keywords such as Public; OOo BASIC does not.
- VB supports the optional keyword ByRef. This keyword is not supported by OOo BASIC. Passing parameters by reference is the default behavior so the keyword is redundant.
- VB supports the keyword ParamArray, OOo BASIC does not.
- VB supports default parameters, OOo BASIC does not.
- VB.NET does not support the function IsMissing; a method of declaring default parameters is used instead. Other VB versions are compatible with OOo BASIC.

17.5. Operators

- VB.NET does not support the EQV or IMP operators. Other VB versions are compatible with OOo BASIC.
- VB.NET supports extra operators such as Like, AndAlso, and OrElse.
- VB has different precedence rules. For example, AND is higher than OR, which is higher than XOR.
- VB supports an Option Compare statement that controls how strings are compared. This is not compatible with OOo BASIC. Use the StrComp function instead.
- VB.NET follows standard mathematical convention and gives exponentiation a higher precedence than negation. For example, $-2^2 = -4$ in VB and 4 in OOo BASIC.
- Flow Control
- VB supports a For Each ... Next Loop construct not supported by OOo BASIC.
- VB.NET does not support the keyword GoSub.
- VB.NET does not support the On GoTo and On GoSub statements.
- Error Handling

- VB uses an Err object to obtain error information. OOo BASIC uses three functions Err, Error, and Erl.
- Some versions of VB support On Error GoTo -1, which functions the same as On Error GoTo 0.
- Some versions of VB require that all error handlers use a unique name, OOo BASIC does not require this.
- OOo BASIC does not allow an On Error commands outside of a subroutine or function.

17.6. Subroutines and Functions

17.6.1. Numerical Subroutines and Functions

Although VB remains largely compatible with OOo BASIC, VB.NET changed the names and methods to access some of the common functions.

Table 17.3: VB.NET uses different names for some numerical functions.

OOo BASIC	VB	VB.NET	Return Value
ABS	ABS	Math.Abs	The absolute value of a specified number.
ATN	ATN	Math.Atan	The angle whose tangent is the specified number.
COS	COS	Math.Cos	The cosine of the specified angle.
Exp	Exp	Math.Exp	The base of natural logarithms raised to a power.
Log	Log	Math.Log	The logarithm of a number. In VB.NET this method can be overloaded to return either the natural (base e) logarithm or the logarithm of a specified base.
not supported	Round	Math.Round	Value containing the number nearest the specified value.
Sgn	Sgn	Math.Sign	Integer value indicating the sign of a number.
SIN	SIN	Math.Sin	The sine of an angle.
Sqr	Sqr	Math.Sqrt	The square root of a number.
TAN	TAN	Math.Tan	The tangent of an angle.

- VB contains more functions, such as CCur to convert to the Currency type.
- There are differences in the whole number types. For example, although CInt returns an Integer in both languages, an integer in VB.NET is equivalent to an OOo BASIC Long.
- The rounding rules are different in VB, numbers are rounded to the nearest even number when the decimal point is exactly .5; this is called IEEE rounding.

In VB Date\$ and Time\$ return a string and value but Date and Time return numerical based types suitable for mathematical operations. OOo Basic supports all four functions, but they all return a string.

The Date and Time functions are documented to set the system date and time. This is not currently supported.

The CHR function is frequently written as CHR\$. In VB, CHR\$ returns a string and can not handle null input values and CHR returns a variant able to accept and propagate null values. In OOo Basic, they are the same; they both return strings and they both generate a runtime error with a null input value.

In VB, LSet allows you to overlay data from one user-defined type with data from another. This takes all the bytes from one data structure and overlays them on top of another, ignoring the underlying structure. In OOo BASIC, LSet only manipulates strings.

VB supports all of OOo BASIC format specifiers, and more.

Table 17.4: OOo BASIC Keywords deprecated when moving to VB to VB.NET.

Word	Word	Word	Word	Word	Word
IsEmpty	IsNull	IsObject	Line	LSet	MsgBox
Now	RSet		Wend		

17.7. Compatibility mode and private variables

OOo supports a compatibility mode, which I describe in my book and I do not feel like taking the time to rewrite here, but, I will write a little about it.

Declare a variable private to a module by declaring it at the head of the module before the subroutines and functions as follows:

```
Private priv_1 As String
DIM priv_2 As String
```

Unfortunately, in OOo Basic, a bug allows private variables to act as public variables; meaning they are visible in other modules and libraries. I recommend that all modules start with “Option Explicit”, which forces you to declare all variables before use. Unfortunately, a variable declared private in another module will be visible so not even “Option Explicit” will notice the variable.

There is concern with fixing the Private declaration because it could break existing macros, so this will not be fixed.

Use “Option Compatible” to enable compatibility mode for the module. This enables defaults that are similar to VB; for example, default array dimensions and the behavior of some file functions. Unfortunately, “Option Compatible” does not change the way the compiler recognizes variables. Use “CompatibilityMode(true)” to enable compatibility mode during

run time, which affects how variables are found. In other words, private variables really are private.

18. Operators and Precedence

OpenOffice.org Basic supports the basic numerical operators -, +, /, *, and ^. The operators use the standard precedence orders, but I have indicated them here anyway. The Logical operators return 0 for false (no bits set) and -1 for true (all bits set). For a more complete description, see the section listing operators and functions.

Precedence	Operator	Description
0	AND	Bitwise on numerics and logical on Boolean
0	OR	Bitwise on numerics and logical on Boolean
0	XOR	Bitwise on numerics and logical on Boolean
0	EQV	Logical and/or Bitwise equivalence
0	IMP	Logical Implication (buggy as of 1.0.3.1)
1	=	Logical
1	<	Logical
1	>	Logical
1	<=	Logical
1	>=	Logical
1	<>	Logical
2	-	Numerical Subtraction
2	+	Numerical Addition and String Concatenation
2	&	String Concatenation
3	*	Numerical Multiplication
3	/	Numerical Division
3	MOD	Numerical remainder after division
4	^	Numerical Exponentiation

```
Sub TestPrecedence
  Dim i%
  Print 1 + 2 OR 1      REM Prints 3
  Print 1 + (2 OR 1)   REM Prints 4
  Print 1 + 2 AND 1    REM Prints 1
  Print 1 + 2 * 3      REM Prints 7
  Print 1 + 2 * 3 ^2   REM Prints 19
  Print 1 = 2 OR 4     REM Prints 4
  Print 4 AND 1 = 1    REM Prints 4
End Sub
```

Warning

Boolean values are internally stored as integers with False = 0 and True = -1. This allows numerical operators to be used with Boolean values but I discourage this (1 + True = False). Use boolean operators instead.

19. String Manipulations

offers a few methods for the manipulation of strings.

Function	Description
Asc(s\$)	ASCII value of the first character in the string.
Chr\$(i)	Return the character corresponding to the ASCII code.
CStr(Expression)	Convert the numeric expression to a string.
Format(number [, f])	Format the number based on the format string.
Hex(Number)	String that represents the hexadecimal value of a number.
InStr([i,] s\$, f\$[, c])	Position of f in s, 0 if not found. Can be case-insensitive. Return type is Long value coerced into an Integer so negative values may be returned for large strings.
LCase(s\$)	Returns string as all lower case.
Left(s\$, n)	Return the leftmost n characters from s. n is an integer but the string may be 64K in size.
Len(s\$)	Returns the length of the string s.
LSet s\$ = Text	Left align a string. Broken in 1.0.3.1, fixed in 1.1
LTrim(s\$)	Return a string with no leading spaces. Does not modify the string.
Mid(s\$, i[, n])	Substring from location i of length n.
Mid(s\$, i, n, r\$)	Replace the substring with r with limitations. I use to delete portions.
Oct(Number)	String that represents the Octal value of a number.
Right(s\$, n)	Return the rightmost n characters from s. n is an integer but the string may be 64K in size.
RSet s\$ = Text	Right align a string.
RTrim(s\$)	Return string with no trailing spaces.
Space(n)	Returns a string that consists of a specified amount of spaces.
Str(Expression)	Convert the numeric expression to a string.
StrComp(x\$, y\$[, c])	Return -1 if x>y, 0 if x=y, and 1 if x<y. If c=1 then case-insensitive.
String(n, {i s\$})	Create a string with n characters. If an integer is used, this is considered the ASCII character to repeat. If a string is used, then the first character is repeated n times.
Trim(s\$)	Return a string with no leading or trailing space from the string.
UCase(s\$)	Returns string as all upper case.
Val(s\$)	Convert the string to a number.

In the on-line help, the example for case conversion is incorrect. Here is how it should read.

```

Sub ExampleLUCase
    Dim sVar As String
    sVar = "Las Vegas"
    Print LCase(sVar) REM Returns "las vegas"
    Print UCase(sVar) REM Returns "LAS VEGAS"
end Sub

```


19.1. Remove Characters From String

This will remove characters from a string. The silly thing about this macro is that it is better written using the built in mid() method. The difference is that the mid() method modifies the current string whereas this returns a new string. I still should have done this using the mid() method, but I did not know about it until later.

```
'Remove a certain number of characters from a string
Function RemoveFromString(s$, index&, num&) As String
    If num = 0 Or Len(s) < index Then
        'If removing nothing or outside the range then return the string
        RemoveFromString = s
    ElseIf index <= 1 Then
        'Removing from the start
        If num >= Len(s) Then
            RemoveFromString = ""
        Else
            RemoveFromString = Right(s, Len(s) - num)
        End If
    Else
        'Removing from the middle
        If index + num > Len(s) Then
            RemoveFromString = Left(s, index - 1)
        Else
            RemoveFromString = Left(s, index - 1) + Right(s, Len(s) - index - num + 1)
        End If
    End If
End Function
```

19.2. Replace Text In String

This may be used to delete areas of a string by specifying the replacement string as an empty string. My initial thought was that I could use the mid() method for this as well, but it turns out that the mid method can not cause the string to become larger than it currently is. Because of this, I had to write this macro. It does not modify the existing string, but creates a new string with the replacement in place.

```
REM s$ is the input string to be modified
REM index is a long indicating where the replacement should be made in the
REM string. (1 based)
REM     If index is <= 1 then the text is inserted in front of the string.
REM     If index > Len(s) then it is inserted at the end.
REM num is a long indicating how many characters to replace.
REM     If num is zero length then nothing is removed, but the new string
REM     is inserted.
REM replaces is the string to place into the string.
Function ReplaceInString(s$, index&, num&, replaces$) As String
    If index <= 1 Then
```

```

'Place this in front of the string
If num < 1 Then
    ReplaceInString = replaces + s
ElseIf num > Len(s) Then
    ReplaceInString = replaces
Else
    ReplaceInString = replaces + Right(s, Len(s) - num)
End If
ElseIf index + num > Len(s) Then
    ReplaceInString = Left(s, index - 1) + replaces
Else
    ReplaceInString = Left(s, index - 1) + replaces + Right(s, Len(s) - index -
num + 1)
End If
End Function

```

19.3. Printing The ASCII Values Of A String

This may seem like an odd macro to have, but I used this macro to decide how text was stored in a document. This will print the entire string as a set of ASCII numbers.

```

Sub PrintAll
    PrintAscii(ThisComponent.text.getString())
End Sub
Sub PrintAscii(TheText As String)
    If Len(TheText) < 1 Then Exit Sub
    Dim msg$, i%
    msg = ""
    For i = 1 To Len(TheText)
        msg = msg + Asc(Mid(TheText, i, 1)) + " "
    Next i
    Print msg
End Sub

```

19.4. Remove All Occurrences Of A String

```

REM This deletes all occurrences of bad$ from s$
REM This modifies the string s$
Sub RemoveFromString(s$, bad$)
    Dim i%
    i = InStr(s, bad)
    Do While i > 0
        Mid(s, i, Len(bad), "")
        i = InStr(i, s, bad)
    Loop
End Sub

```

20. Numeric Manipulations

Function	Description
Abs(Number)	Return the absolute value of the number as a Double.
Asc(s\$)	Return the ASCII Integer of the first character in the string.
Atn(x)	Return the angle, in radians, whose tangent is x
Blue(color)	Returns the Blue component of the given color code.
CByte(Expression)	Convert a string or number to a byte.
CDbl(Expression)	Convert a string or number to a Double.
CInt(Expression)	Convert a string or number to an Integer.
CLng(Expression)	Convert a string or number to a Long.
Cos(x)	Calculates the cosine of an angle specified in radians.
CSng(Expression)	Convert a string or number to a single precision number.
CStr(Expression)	Convert a string or number to a String.
Exp(Expression)	Base of the natural logarithm (e = 2.718282) raised to a power.
Fix(Expression)	Return the integer portion of a number after truncation.
Format(number [, f\$])	Format the number based on the format string.
Green(color)	Returns the Green component of the given color code.
Hex(Number)	String that represents the hexadecimal value of a number.
Int(Number)	Rounds the integer toward negative infinity. See Also: Fix().
IsNumeric (Var)	Tests whether the given expression is a number.
Log(Number)	Natural logarithm of a number.
Oct(Number)	String that represents the Octal value of a number.
Randomize [Number]	Initializes the random-number generator.
Red(color)	Returns the Red component of the given color code.
RGB (Red, Green, Blue)	Long color value consisting of red, green, and blue components.
Rnd [(Expression)]	Return a random number between 0 and 1.
Sgn (Number)	Returns 1, -1, or 0 if the number is positive, negative, or zero.
Sin(x)	Calculates the sine of an angle specified in radians.
Sqr(Number)	Square root of a numeric expression.
Tan(x)	Calculates the tangent of an angle specified in radians.
n = TwipsPerPixelX	Returns the number of twips representing the width of a pixel.
n = TwipsPerPixelY	Returns the number of twips representing the height of a pixel.
Val(s\$)	Convert a string to a number.

21. Date Manipulations

Function	Description
CDate(Expression)	Convert a string or number to a date.
CDateFromIso(String)	Return the internal date number from a string containing a date in ISO format.
CDateToIso(Number)	Returns the date in ISO format from a serial date number that was generated with DateSerial or DateValue.
Date	Return the current system date.
Date = s\$	Set the current system date.
DateSerial(y%, m%, d%)	Return a date from the year, month, and day.
DateValue([date])	Long from a date usable to determine the difference between dates.
Month(Number)	Day of month from a time generated by DateSerial or DateValue.
GetSystemTicks()	Returns the system ticks provided by the operating system.
Hour(Number)	Hour from a time generated by TimeSerial or TimeValue.
Minute(Number)	Minute from a time generated by TimeSerial or TimeValue.
Month(Number)	Month from a time generated by DateSerial or DateValue.
Now	Current system date and time as a Date value.
Second(Number)	Second from a time generated by TimeSerial or TimeValue.
Time	Current system time
Timer	Number of seconds that have elapsed since midnight.
TimeSerial (h, m, s)	Serial time value from the specified hour, minute, and second.
TimeValue (s\$)	Serial time value from a formatted string.
Wait millisec	Pause for the given number of milliseconds
WeekDay(Number)	Day of week from a time generated by DateSerial or DateValue.
Year(Number)	Year from a time generated by DateSerial or DateValue.

22. File Manipulations

Function	Description
ChDir (s\$)	Changes the current directory or drive.
ChDrive(s\$)	Changes the current drive.
Close #n% [, #n2%[,...]]	Close files opened with the Open statement.
ConvertFromURL(s\$)	Converts a file URL to a system file name.
ConvertToURL(s\$)	Converts a system file name to a file URL.
CurDir([s\$])	Returns the current directory of the specified drive.
Dir [s\$ [, Attrib%]]	Perform a directory listing.
EOF(n%)	Has the file pointer reached the end of the file?
FileAttr (n%, Attribut%)	Return the file attribute of an open file.
FileCopy from\$, to\$	Copy a file.
FileDateTime(s\$)	Return a string of the file date and time.
FileExists(s\$)	Determine if a file or directory exists.
FileLen (s\$)	Length of file in bytes.
FreeFile	Next available file number. Prevents simultaneous use.
Get [#]n%, [Pos], v	Read a record or bytes from a file.
GetAttr(s\$)	Returns a bit pattern which identifies the file type.
Input #n% v1[, v2[, [...]]]	Read data from an open sequential file.
Kill f\$	Delete a file form a disk.
Line Input #n%, v\$	Read a string from a sequential file into a variable.
Loc (FileNumber)	Returns the current position in an open file.
Lof (FileNumber)	Returns the current size of an open file.
MkDir s\$	Create a new directory.
Name old\$, new\$	Rename an existing file or directory.
Open s\$ [#]n%	Open a file. Most parameters not listed, this is very flexible.
Put [#] n%, [pos], v	Writes a record a sequence of bytes to a file.
Reset	Closes all open files and flushes all buffers to disk.
RmDir f\$	Remove a directory.
Seek[#]n%, Pos	Move the file pointer.
SetAttr f\$, Attribute%	Set the file attributes.
Write [#]n%, [Exprs]	Write data to a sequential file.

23. Operators, Statements, and Functions

23.1. Subtraction operator (-)

Summary:

Subtract two numerical values. The standard mathematical precedence is used as shown on page 419.

Syntax:

Result = Expression1 - Expression2

Parameter:

Result : Result of the subtraction.

Expression1, Expression2 : Any numerical expressions.

Example:

```
Sub SubtractionExample
  Print 4 - 3      '1
  Print 1.23e2 - 23 '100
End Sub
```

23.2. Multiplication operator (*)

Summary:

Multiply two numerical values. The standard mathematical precedence is used as shown on page 419.

Syntax:

Result = Expression1 * Expression2

Parameter:

Result : Result of the multiplication.

Expression1, Expression2 : Any numerical expressions.

Example:

```
Sub MultiplicationExample
  Print 4 * 3      '12
  Print 1.23e2 * 23 '2829
End Sub
```

23.3. Addition operator (+)

Summary:

Add two numerical values. Although this works with boolean values because they are represented as integers I recommend against it. Experimentally, it appears to mimic the result of the Or operator but I recommend against it because conversion may yield problems. The operations are done in the integer domain and then converted back to a boolean. This may yield problems. The standard mathematical precedence is used as shown on page 419.

Syntax:

Result = Expression1 + Expression2

Parameter:

Result : Result of the addition.

Expression1, Expression2 : Any numerical expressions.

Example:

```
Sub SubtractionExample
    Print 4 - 3      '1
    Print 1.23e2 - 23 '100
End Sub
```

23.4. Power operator (^)**Summary:**

Raise a number to a power. Let the equation $x = y^z$ represents the operator. If z is an integer, then x is the result of multiplying y by itself z times. The standard mathematical precedence is used as shown on page 419.

Syntax:

Result = Expression ^ Exponent

Parameter:

Result : Result of the exponentiation.

Expression: Any numerical expression.

Exponent: Any numerical expression.

Example:

```
Sub ExponentiationExample
    Print 2 ^ 3      '8
    Print 2.2 ^ 2     '4.84
    Print 4 ^ 0.5     '2
End Sub
```

23.5. Division operator (/)**Summary:**

Divide two numerical values. Be careful with the results because a division may not produce an integer result when you expect one. Use the Int() function if this is important. The standard mathematical precedence is used as shown on page 419.

Syntax:

Result = Expression1 / Expression2

Parameter:

Result : Result of the division.

Expression1, Expression2 : Any numerical expressions.

Example:

```
Sub DivisionExample
    Print 4 / 2 '2
    Print 11/2 '5.5
End Sub
```

23.6. AND Operator

Summary:

Perform a logical AND on boolean values and a bitwise AND on numerical values. A bitwise AND on a double, appears to cause a conversion to an integer type. Numerical overflow is possible. The standard mathematical precedence is used as shown on page 419 and the operation table is shown below.

x	y	x AND y
TOSI	TOSI	TOSI
TOSI	EPÄTOSI	EPÄTOSI
EPÄTOSI	TOSI	EPÄTOSI
EPÄTOSI	EPÄTOSI	EPÄTOSI
1	1	1
1	0	0
0	1	0
0	0	0

Syntax:

Result = Expression1 AND Expression2

Parameter:

Result : Result of the operation.

Expression1, Expression2 : Numerical or Boolean expression.

Example:

```
Sub AndExample
    Print (3 AND 1) 'Prints 1
    Print (True AND True) 'Prints -1
    Print (True AND False) 'Prints 0
End Sub
```


23.7. Abs Function

Summary:

Return the absolute value of a numeric expression. If the parameter is a string, it is first converted to a number, probably using the Val function. If the number is non-negative, then it is returned, otherwise, the negative of the number is returned.

Syntax:

Abs (Number)

Return value:

Double

Parameter:

Number: Any numeric expression.

Example:

```
Sub AbsExample
    Print Abs(3)      '3
    Print Abs(-4)     '4
    Print Abs("-123") '123
End Sub
```

23.8. Array Function

Summary:

Create a Variant array from the parameter list. This is the quickest method to create an array of constants.

Warning	If you assign the returned Variant array to a non-Variant array, you can not preserve the data if you re-dimension the array. I consider it a bug that you can assign the Variant array to a non-Variant array.

See also the DimArray Function.

Syntax:

Array (Argument list)

Return value:

Variant array containing the argument list.

Parameter:

Argument list: List of values separated by commas from which to create a list.

Example:

```
Sub ArrayExample
    Dim a(5) As Integer
    Dim b() As Variant
```

```

Dim c() As Integer
REM Array() returns a variant type
REM b is dimensioned from 0 to 9 where b(i) = i+1
b = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
PrintArray("b at initial assignment", b())
REM b is dimensioned from 1 to 3 where b(i) = i+1
ReDim Preserve b(1 To 3)
PrintArray("b after ReDim", b())
REM The following is NOT valid because the array is already dimensioned
REM to a different size, but you can ReDim it, go figure!
REM a = Array(0, 1, 2, 3, 4, 5)

REM c is dimensioned from 0 to 5.
REM This concerns me because "Hello" is a String value, but
REM it is allowed as of 1.0.2
c = Array(0, 1, 2, "Hello", 4, 5)
PrintArray("c, Variant assigned to an Integer array", c())
REM Ironically, this allowed but c will contain no data!
ReDim Preserve c(1 To 3) As Integer
PrintArray("c after ReDim", c())
End Sub

Sub PrintArray (lead$, a() As Variant)
    Dim i%, s$
    s$ = lead$ + Chr(13) + LBound(a()) + " to " + UBound(a()) + ":" + Chr(13)
    For i% = LBound(a()) To UBound(a())
        s$ = s$ + a(i%) + " "
    Next
    REM I MsgBox rather than Print because I have an embeded Chr(13)
    MsgBox s$
End Sub

```

23.9. Asc Function

Summary:

Return the ASCII value (American Standard Code for Information Interchange) of the first character in a string, the rest are ignored. A run-time error is reported if the string has zero length. 16 bit Unicode characters are allowed. This is essentially the inverse of the CHR\$ function.

Syntax:

Asc (Text As String)

Return value:

Integer

Parameter:

Text: Any valid string expression.

Example:

```
Sub AscExample
    Print Asc("ABC")    '65
End Sub
```

23.10. ATN Function

Summary:

Arctangent of a numeric expression with a return value in the range $-\pi/2$ to $\pi/2$ (radians). This is the inverse of the tangent (Tan) function. For the non-mathematically minded, this is a trigonometric function.

Syntax:

ATN(Number)

Return value:

Double

Parameter:

Number:

Example:

```
Sub ExampleATN
    Dim dLeg1 As Double, dLeg2 As Double
    dLeg1 = InputBox("Enter the length of the adjacent leg: ", "Adjacent")
    dLeg2 = InputBox("Enter the length of the opposite leg: ", "Opposite")
    MsgBox "The Angle is " + Format(ATN(dLeg2/dLeg1), "##0.0000") _
        + " radians" + Chr(13) + "The Angle is " _
        + Format(ATN(dLeg2/dLeg1) * 180 / Pi, "##0.0000") + " degrees"
End Sub
```

23.11. Beep Statement

Summary:

Generate a system beep (sound).

Syntax:

Beep

Example:

```
Sub ExampleBeep
    Beep
    Beep
End Sub
```

23.12. Blue Function

Summary:

Colors are represented by a long integer. Return the blue component of the specified color code. See also RGB, Red, and Green.

Syntax:

Blue (Color As Long)

Return value:

Integer in the range of 0 to 255.

Parameter:

Color value: Long integer expression representing a color.

Example:

```
Sub ExampleColor
    Dim lColor As Long
    lColor = RGB(255,10,128)
    MsgBox "The color " & lColor & " consists of:" & Chr(13) & _
        "Red = " & Red(lColor) & Chr(13) & _
        "Green= " & Green(lColor) & Chr(13) & _
        "Blue= " & Blue(lColor) & Chr(13) , 64,"Colors"
End Sub
```

23.13. ByVal Keyword

Summary:

Parameters to user defined subroutines and functions are passed by reference. If the subroutine or function modifies the parameter, it is also modified in the calling program. This can cause strange results if the calling parameter is a constant or if the caller does not expect it. The ByVal keyword specifies that the parameter should be passed by value and not by reference.

Syntax:

Sub Name(ByVal ParmName As ParmType)

Example:

```
Sub ExampleByVal
    Dim j As Integer
    j = 10
    ModifyParam(j)
    Print j REM 9
    DoNotModifyParam(j)
    Print j REM 9
End Sub
Sub ModifyParam(i As Integer)
```

```

    i = i - 1
End Sub
Sub DoNotModifyParam(ByVal i As Integer)
    i = i - 1
End Sub

```

23.14. Call Keyword

Summary:

Transfers program control to a sub, function or DLL procedure. The Call is optional unless a DLL in which case it must first be defined. The parameters may be enclosed in parenthesis and they must be enclosed in parenthesis is a function is executed as an expression.

See Also: Declare

Syntax:

[Call] Name [Parameter]

Parameter:

Name: Name of the sub, function or DLL to call

Parameter: The type and number of parameters is dependent on the called routine.

Example:

```

Sub ExampleCall
    Call CallMe "This text will be displayed"
End Sub
Sub CallMe(s As String)
    Print s
End Sub

```

23.15. CBool Function

Summary:

Convert the parameter to a boolean. If the expression is a numeric, 0 maps to False and anything else maps to True. If the expression evaluates to a String, then “true” and “false” (case insensitive) map to True and False. Strings with any other value generate a run time error. This is useful to force the result to be a boolean. If I call a function that returns a number, such as InStr, I could write “If CBool(InStr(s1, s2)) Then” rather than “If InStr(s1, s2) <> 0 Then”.

Syntax:

CBool (Expression)

Return value:

Boolean

Parameter:

Expression: Numeric, Boolean,

Example:

```
Sub ExampleCBool
    Print CBool(1.334e-2)      'True
    Print CBool("TRUE")      'True
    Print CBool(-10 + 2*5)    'False
    Print CBool("John" <> "Fred") 'True
End Sub
```

23.16. CByte Function

Summary:

Convert a string or a numeric expression to the type Byte. String expressions are converted to a numeric and doubles are rounded. If the expression is too large or negative, an error is generated.

Syntax:

Cbyte(expression)

Return value:

Byte

Parameter:

Expression : a string or a numeric expression.

Example:

```
Sub ExampleCByte
    Print Int(CByte(133))      '133
    Print Int(CByte("AB"))    '0
    'Print Int(CByte(-11 + 2*5)) 'Error, out of range
    Print Int(CByte(1.445532e2)) '145
    Print CByte(64.9)          'A
    Print CByte(65)            'A
    Print Int(CByte("12"))    '12
End Sub
```

23.17. CDate Function

Summary:

Convert to a Date. Numeric expressions contain the date, beginning from December 31, 1899 to the left of the decimal and time to the right of the decimal. String expressions must be formatted as defined by the DateValue and TimeValue function conventions. In other words, the string formatting is locale dependent. CDateFromIso is not dependent on your locale so it safer if you desire to code globally.

Syntax:

CDate (Expression)

Return value:

Date

Parameter:

Expression: Any string or numeric expression to be converted.

Example:

```
sub ExampleCDate
    MsgBox cDate(1000.25) REM 09.26.1902 06:00:00
    MsgBox cDate(1001.26) REM 09.27.1902 06:14:24
    Print DateValue("06/08/2002")
    MsgBox cDate("06/08/2002 15:12:00")
end sub
```

23.18. CDateFromIso Function

Summary:

Returns the internal date number from a string containing a date in ISO format.

Syntax:

CDateFromIso(String)

Return value:

Internal date number

Parameter:

String : a string containing an ISO date. The year may have two or four figures.

Example:

```
sub ExampleCDateFromIso
    MsgBox cDate(37415.70) REM 08 June 2002 16:48:00
    Print CDateFromIso("20020608") REM 08 June 2002
    Print CDateFromIso("020608") REM 08 June 1902
    Print Int(CDateFromIso("20020608")) REM 37415
end sub
```

23.19. CDateToIso Function

Summary:

Return the date in ISO format from a serial date number generated with DateSerial or DateValue.

Syntax:

CDateToIso(Number)

Return value:

String

Parameter:

Number : Integer that contains the serial date number.

Example:

```
Sub ExampleCDateToIso
    MsgBox "" & CDateToIso(Now) , 64, "ISO Date"
End Sub
```

23.20. CDb1 Function

Summary:

Converts any numerical expression or string expression to a double type. Strings must be formatted based on the locale. In the USA, “12.34” will work but this will fail elsewhere.

Syntax:

CDbl (Expression)

Return value:

Double

Parameter:

Expression : Any string or numeric expression to be converted.

Example:

```
Sub ExampleCDbl
    MsgBox CDb1(1234.5678)
    MsgBox CDb1("1234.5678")
End Sub
```

23.21. ChDir statement is deprecated

Summary:

Change the current directory or drive. If you only want to change the current drive, enter the drive letter followed by a colon. The ChDir statement is deprecated, do not use it.

Syntax:

ChDir(Text)

Parameter:

Text: Any string expression that specifies the directory path or drive.

Example:

```
Sub ExampleChDir
    Dim sDir as String
    sDir = CurDir
    ChDir( "C:\temp" )
End Sub
```



```

msgbox CurDir
ChDir( sDir )
msgbox CurDir
End Sub

```

23.22. ChDrive statement is deprecated

Summary:

Change the current drive. The drive letter must be expressed as an uppercase letter. You may use the OnError statement to catch any errors. This is deprecated, do not use this.

Syntax:

ChDrive(Text)

Parameter:

Text: String expression containing the drive letter. URL notation is accepted.

Example:

```

Sub ExampleCHDrive
    On Local Error Goto NoDrive
    ChDrive "Z" REM Only possible if a drive 'Z' exists.
    Print "Completed"
    Exit Sub
NoDrive:
    Print "Sorry, the drive does not exist"
    Resume Next
End Sub

```

23.23. Choose Function

Summary:

Return a selected value from a list of arguments. This is a quick method to select a value from a list. If the index is out of bounds (less than 1 or greater than n), then a null is returned.

Syntax:

Choose (Index, Selection_1[, Selection_2, ... [,Selection_n]])

Return value:

The type will be whatever type Selection_i happens to be.

Parameter:

Index: A numeric expression specifying the value to return.

Selection_i: A value to return.

Example:

In this example, the variable “o” is not given a type so it takes the type of Selection_i. Selection_1 is type “String” and Selection_2 is type Double. If I define “o” to have a type, then the return value is cast to that type.

```
Sub ExampleChoose
    Dim sReturn As String
    Dim sText As String
    Dim i As Integer
    Dim o
    sText = InputBox ("Enter a number (1-3):", "Example")
    i = Int(sText)
    o = Choose(i, "One", 2.2, "Three")
    If IsNull(o) Then
        Print "Sorry, '" + sText + "' is not valid"
    Else
        Print "Obtained '" + o + "' of type " + TypeName(o)
    End If
End Sub
```

23.24. Chr Function

Summary:

Return the character corresponding to the specified character (ASCII or Unicode) code.

This is used to create special string sequences such as control codes to send to printers, tabs, new lines, carriage returns, etc. This also provides a method to insert a double quote into a string. This is sometimes written as “Chr\$()”.

See Also the Asc Function.

Syntax:

Chr(Expression)

Return value:

String

Parameter:

Expression: Numeric variables representing a valid 8 bit ASCII value (0-255) or a 16 bit Unicode value.

Example:

```
Example:
sub ExampleChr
    REM Show "Line 1" and "Line 2" on separate lines.
    MsgBox "Line 1" + Chr$(13) + "Line 2"
End Sub
```

23.25. CInt Function

Summary:

Converts any numerical expression or string expression to an Integer type. Strings must be formatted based on the locale. In the USA, “12.34” will work.

See Also: Fix

Syntax:

CInt(Expression)

Return value:

Integer

Parameter:

Expression : Any string or numeric expression to be converted.

Example:

```
Sub ExampleCInt
    MsgBox CInt(1234.5678)
    MsgBox CInt("1234.5678")
End Sub
```

23.26. CLng Function

Summary:

Converts any numerical expression or string expression to Long type. Strings must be formatted based on the locale. In the USA, “12.34” will work.

Syntax:

CLong(Expression)

Return value:

Long

Parameter:

Expression : Any string or numeric expression to be converted.

Example:

```
Sub ExampleCLng
    MsgBox CLng(1234.5678)
    MsgBox CLng("1234.5678")
End Sub
```

23.27. Close Statement

Summary:

Close files previously opened with the Open statement. Multiple files may be closed simultaneously.

See also Open, EOF, Kill, and FreeFile

Syntax:

Close #FileNumber As Integer[, #FileNumber2 As Integer[,...]]

Parameter:

FileNumber : Integer expression that specifies a previously opened file.

Example:

```
Sub ExampleCloseFile
    Dim iNum1 As Integer, iNum2 As Integer
    Dim sLine As String, sMsg As String
    'Next available file number!
    iNum1 = FreeFile
    Open "c:\data1.txt" For Output As #iNum1
    iNum2 = FreeFile
    Open "c:\data2.txt" For Output As #iNum2
    Print #iNum1, "Text in file one for number " + iNum1
    Print #iNum2, "Text in file two for number " + iNum2
    Close #iNum1, #iNum2
    Open "c:\data1.txt" For Input As #iNum1
    iNum2 = FreeFile
    Open "c:\data2.txt" For Input As #iNum2
    sMsg = ""
    Do While not EOF(iNum1)
        Line Input #iNum1, sLine
        If sLine <> "" Then sMsg = sMsg+"File: "+iNum1+": "+sLine+Chr(13)
    Loop
    Close #iNum1
    Do While not EOF(iNum2)
        Line Input #iNum2, sLine
        If sLine <> "" Then sMsg = sMsg+"File: "+iNum2+": "+sLine+Chr(13)
    Loop
    Close #iNum2
    MsgBox sMsg
End Sub
```

23.28. Const Statement

Summary:

Constants improve the readability of a program by assigning names to constants and also by providing a single point of definition. Constants may include a type definition but this is not required. A constant is defined once and can not be modified.

Syntax:

Const Text [As type] = Expression[, Text2 [As type] = Expression2[, ...]]

Parameter:

Text: Any constant name which follows the standard variable naming conventions.

Example:

```
Sub ExampleConst
    Const iVar As String = 1964
    Const sVar = "Program", dVar As Double = 1.00
    MsgBox iVar & " " & sVar & " " & dVar
End Sub
```

23.29. ConvertFromURL Function

Summary:

Converts a file URL to a system file name.

Syntax:

ConvertFromURL(filename)

Return value:

String

Parameter:

Filename: File name as a URL

Example:

```
Sub ExampleFromUrl
    Dim s$
    s = "file:///c:/temp/file.txt"
    MsgBox s & " => " & ConvertFromURL(s)
    s = "file:///temp/file.txt"
    MsgBox s & " => " & ConvertFromURL(s)
End Sub
```

23.30. ConvertToURL Function

Summary:

Converts a system file name to a URL.

Syntax:

ConvertToURL(filename)

Return value:

String

Parameter:

Filename: File name as a system name.

Example:

```
Sub ExampleToUrl
    Dim s$
    s = "c:\temp\file.txt"
    'file:///c:/temp/file.txt
    MsgBox s & " => " & ConvertToURL(s)
    s = "\temp\file.txt"
    'file:///temp/file.txt
    MsgBox s & " => " & ConvertToURL(s)
End Sub
```

23.31. Cos Function

Summary:

Cosine of a numeric expression with a return value in the range -1 to 1. For the non-mathematically minded, this is a trigonometric function.

Syntax:

Cos(Number)

Return value:

Double

Parameter:

Number:

Example:

```
Sub ExampleCos
    Dim dLeg1 As Double, dLeg2 As Double, dHyp As Double
    Dim dAngle As Double
    dLeg1 = InputBox("Enter the length of the adjacent leg: ", "Adjacent")
    dLeg2 = InputBox("Enter the length of the opposite leg: ", "Opposite")
    dHyp = Sqr(dLeg1 * dLeg1 + dLeg2 * dLeg2)
    dAngle = Atn(dLeg2 / dLeg1)
    MsgBox "Adjacent Leg = " + dLeg1 + Chr(13) + _
        "Opposite Leg = " + dLeg2 + Chr(13) + _
        "Hypotenuse = " + Format(dHyp, "##0.0000") + Chr(13) + _
        "Angle = " + Format(dAngle*180/Pi, "##0.0000") + " degrees" + Chr(13) + _
        "Cos = " + Format(dLeg1 / dHyp, "##0.0000") + Chr(13) + _
        "Cos = " + Format(Cos(dAngle), "##0.0000")
End Sub
```

23.32. CreateUnoDialog Function

Summary:

Create a Basic UNO object that represents a UNO dialog control during Basic runtime.

Dialogs are defined in the dialog libraries. To display a dialog, a "live" dialog must be created from the library.

Syntax:

CreateUnoDialog(oDlgDesc)

Return value:

Object: Dialog to execute!

Parameter:

oDlgDesc : Dialog description previously defined in a library.

Example:

```
Sub ExampleCreateDialog
    Dim oDlgDesc As Object, oDlgControl As Object
    DialogLibraries.LoadLibrary("Standard")
    ' Get dialog description from the dialog library
    oDlgDesc = DialogLibraries.Standard
    Dim oNames(), i%
    oNames = DialogLibraries.Standard.getElementNames()
    i = lBound( oNames() )
    while( i <= uBound( oNames() ) )
        MsgBox "How about " + oNames(i)
        i = i + 1
    wend
    oDlgDesc = DialogLibraries.Standard.Dialog1
    ' generate "live" dialog
    oDlgControl = CreateUnoDialog( oDlgDesc )
    ' display "live" dialog
    oDlgControl.execute
End Sub
```

23.33. CreateUnoService Function

Summary:

Instantiates an UNO service with the ServiceManager.

Syntax:

oService = CreateUnoService(UNO service name)

Return value:

The requested service.

Parameter:

String name of the requested service

Example:

This example was provided by Laurent Godard. Like him, I was unable to figure out how to include text in the email message because as of 1.1.1, you can not include text. The mail service was created to send attachments, hopefully this will be fixed or changed in the future.

```
Sub SendSimpleMail()
    Dim vMailSystem, vMail, vMessage

    vMailSystem=createUnoService("com.sun.star.system.SimpleSystemMail")
    vMail=vMailSystem.querySimpleMailClient()
    'You want to know what else you can do with this, see
    'http://api.openoffice.org/docs/common/ref/com/sun/star/system/XSimpleMailMessage.html
    vMessage=vMail.createSimpleMailMessage()
    vMessage.setRecipient("andrew@pitonyak.org")
    vMessage.setSubject("This is my test subject")

    'Attachments are set by a sequence which in basic means an array
    'I could use ConvertToURL() to build the URL!
    Dim vAttach(0)
    vAttach(0) = "file:///c:/macro.txt"
    vMessage.setAttachment(vAttach())

    'DEFAULTS Launch the currently configured system mail client.
    'NO_USER_INTERFACE Do not show the interface, just do it!
    'NO_LOGON_DIALOG No logon dialog but will throw an exception if one is
    required.
    vMail.sendSimpleMailMessage(vMessage,
    com.sun.star.system.SimpleMailClientFlags.NO_USER_INTERFACE)
End Sub
```

23.34. CreateUnoStruct Function

Summary:

Create an instance of an UNO structure type.

For com.sun.star.beans.PropertyValue it is better to use the following construct!

```
Dim oStruct As New com.sun.star.beans.PropertyValue
```

Syntax:

oStruct = CreateUnoStruct(UNO type name)

Parameter:

String name of the requested structure.

Return value:

The requested structure

Example:


```

oStruct = CreateUnoStruct("com.sun.star.beans.PropertyValue")
'*****
'Do you want to choose a certain printer
'Dim mPrinter(0) As New com.sun.star.beans.PropertyValue
'mPrinter(0).Name="Name"
'mPrinter(0).value="Other printer"
'oDoc.Printer = mPrinter()
'You could have done it this way after it was defined!
'mPrinter(0) = CreateUnoStruct("com.sun.star.beans.PropertyValue")

```

23.35. CSng Function

Summary:

Converts any numerical expression or string expression to Single type. Strings must be formatted based on the locale. In the USA, “12.34” will work.

Syntax:

CSng(Expression)

Return value:

Single

Parameter:

Expression : Any string or numeric expression to be converted.

Example:

```

Sub ExampleCLng
    MsgBox CSng(1234.5678)
    MsgBox CSng("1234.5678")
End Sub

```

23.36. CStr Function

Summary:

Convert any expression to a string expression. This is usually used to convert a number to a string.

Input Type	Output Value
Boolean	“True” or “False”
Date	String with the date and time
Null	run-time error
Empty	empty string “”
Any Number	The number as a string. Trailing zeros to the right of the decimal are dropped.

Syntax:

CStr (Expression)

Return value:

String

Parameter:

Expression: Any valid string or numeric expression to be converted.

Example:

```
Sub ExampleCSTR
    Dim sVar As String
    MsgBox CDbl(1234.5678)
    MsgBox CInt(1234.5678)
    MsgBox CLng(1234.5678)
    MsgBox CSng(1234.5678)
    sVar = CStr(1234.5678)
    MsgBox sVar
End Sub
```

23.37. CurDir Function

Summary:

Return the String representing the current path of the specified drive. If the parameter is missing or zero-length, then the path for the current drive is returned. This function is currently broken! On my machine it always returns the OOo /program/ .

Syntax:

CurDir [(Text As String)]

Return value:

String

Parameter:

Text: Optional case-insensitive string expression specifying an existing drive for which to return the current path. This should be a single character.

Example:

```
Sub ExampleCurDir
    MsgBox CurDir("c")
    MsgBox CurDir("p")
    MsgBox CurDir()
End Sub
```

23.38. Date Function

Summary:

Return or change the current system date. The date format is locale-dependent.

Syntax:

Date

Date = Text As String

Return value:

String

Parameter:

Text: New system date as defined in your locale settings.

Example:

```
Sub ExampleDate
    MsgBox "The date is " & Date
End Sub
```

23.39. DateSerial Function

Summary:

Convert a year, month, and day to a Date object. The internal representation is a Double where 0 is December 30,1899. View this double as the number of days from this base date. Negative numbers are before and positive numbers are after.

See also DateValue, Date, and Day.

Warning

Two digit years are considered 19xx. This is not consistent with the DateValue function.

Syntax:

DateSerial (year, month, day)

Return value:

Date

Parameter:

Year: Integer year. Values between 0 and 99 are interpreted as the years 1900 to 1999. All other years must contain four digits.

Month: Integer indicating the month. Valid values from 1 to 12.

Day: Integer indicating the day. Valid values from 1 to 28, 29, 30 or 31 (month dependent).

Example:

```
Sub ExampleDateSerial
    Dim lDate as Long, sDate as String, lNumDays As Long
    lDate = DateSerial(2002, 6, 8)
    sDate = DateSerial(2002, 6, 8)
    MsgBox lDate REM returns 37415
    MsgBox sDate REM returns 06/08/2002
```

```

lDate = DateSerial(02, 6, 8)
sDate = DateSerial(02, 6, 8)
MsgBox lDate REM returns 890
MsgBox sDate REM returns 06/08/1902
end sub

```

23.40. DateValue Function

Summary:

Convert a date string to a single numeric value usable to determine the number of days between two dates.

See also DateSerial, Date, and Day

Warning	Two digit years are considered 20xx. This is not consistent with the DateSerial function.

Syntax:

DateValue [(date)]

Return value:

Long

Parameter:

Date: String representation of the date.

Example:

```

Sub ExampleDateValue
    Dim s(), i%, sMsg$, l1&, l2&
    REM These all map to June 8, 2002.
    s = Array("06.08.2002", "6.08.02", "6.08.2002", "June 08, 2002", _
        "Jun 08 02", "Jun 08, 2002", "Jun 08, 02", "06/08/2002")
    REM If you use these values, it will generate an error
    REM which contradicts the included help
    REM s = Array("6.08, 2002", "06.08, 2002", "06,08.02", "6,08.2002",
    "Jun/08/02")
    sMsg = ""
    For i = LBound(s()) To UBound(s())
        sMsg = sMsg + DateValue(s(i)) + "<=" + s(i) + Chr(13)
    Next
    MsgBox sMsg
    Print "I was married " + (DateValue(Date) - DateValue("6/8/2002")) + " days ago"
end sub

```

23.41. Day Function

Summary:

Return the day of the month based on a serial date number generated with DateSerial or DateValue.

Syntax:

Day (Number)

Return value:

Integer

Parameter:

Number: Serial date number such as returned by DateSerial

Example:

```
Sub ExampleDay
    Print Day(DateValue("6/8/2002")) REM 8
    Print Day(DateSerial(02,06,08)) REM 8
end sub
```

23.42. Declare Statement

Summary:

Used to declare and define a subroutine in a DLL (Dynamic Link Library) to be executed from OpenOffice.org Basic. The ByVal keyword must be used if parameters are to be passed by value rather than by reference.

See also: FreeLibrary, Call

Syntax:

Declare {Sub | Function} Name Lib "Libname" [Alias "Aliasname"] [Parameter] [As Type]

Parameter/Element:

Name: A different name than defined in the DLL, used to call the subroutine from Basic.

Aliasname: Name of the subroutine as defined in the DLL.

Libname: File or system name of the DLL. This library is automatically loaded the first time the function is used.

Argumentlist: List of parameters representing arguments that are passed to the procedure when it is called. The type and number of parameters is dependent on the executed procedure.

Type: Defines the data type of the value returned by a Function procedure. This can be excluded if a type-declaration character is entered after the name.

Example:

```
Declare Sub MyMessageBeep Lib "user32.dll" Alias "MessageBeep" ( long )
Sub ExampleDeclare
    Dim lValue As Long
```

```

    lValue = 5000
    MyMessageBeep( lValue )
    FreeLibrary("user32.dll" )
End Sub

```

23.43. DefBool Statement

Summary:

Set the default variable type, according to a letter range, if no type-declaration character or keyword is specified. You can allow all variables that start with a “b”, for example, to automatically be of type Boolean.

See also: DefBool, DefDate, DefDbL, DefInt, DefLng, DefObj, and DefVar.

Syntax:

DefBool CharacterRange1[, CharacterRange2[,...]]

Parameter:

CharacterRange: Letters specifying the range of variables for which to set a default data type.

Example:

```

REM Prefix definition for variable types:
DefBool b
DefDate t
DefDbL d
DefInt i
DefLng l
DefObj o
DefVar v
DefBool b-d,q
Sub ExampleDefBool
    cOK = 2.003
    zOK = 2.003
    Print cOK REM True
    Print zOK REM 2.003
End Sub

```

23.44. DefDate Statement

Summary

Set the default variable type, according to a letter range, if no type-declaration character or keyword is specified. You can allow all variables that start with a “t”, for example, to automatically be of type Date.

See also: DefBool, DefDate, DefDbL, DefInt, DefLng, DefObj, and DefVar.

Syntax:

DefDate Characterrange1[, Characterrange2[,...]]

Parameter:

Characterrange: Letters specifying the range of variables for which to set a default data type.

Example:

See ExampleDefBool

23.45. DefDbl Statement

Summary

Set the default variable type, according to a letter range, if no type-declaration character or keyword is specified. You can allow all variables that start with a “d”, for example, to automatically be of type Double.

See also: DefBool, DefDate, DefDbL, DefInt, DefLng, DefObj, and DefVar.

Syntax:

DefDbl Characterrange1[, Characterrange2[,...]]

Parameter:

Characterrange: Letters specifying the range of variables for which to set a default data type.

Example:

See ExampleDefBool

23.46. DefInt Statement

Summary

Set the default variable type, according to a letter range, if no type-declaration character or keyword is specified. You can allow all variables that start with a “i”, for example, to automatically be of type Integer.

See also: DefBool, DefDate, DefDbL, DefInt, DefLng, DefObj, and DefVar.

Syntax:

DefInt Characterrange1[, Characterrange2[,...]]

Parameter:

Characterrange: Letters specifying the range of variables for which to set a default data type.

Example:

See ExampleDefBool

23.47. DefLng Statement

Summary

Set the default variable type, according to a letter range, if no type-declaration character or keyword is specified. You can allow all variables that start with a “l”, for example, to automatically be of type Long.

See also: DefBool, DefDate, DefDbL, DefInt, DefLng, DefObj, and DefVar.

Syntax:

DefLng Characterrange1[, Characterrange2[,...]]

Parameter:

Characterrange: Letters specifying the range of variables for which to set a default data type.

Example:

See ExampleDefBool

23.48. DefObj Statement

Summary

Set the default variable type, according to a letter range, if no type-declaration character or keyword is specified. You can allow all variables that start with a “o”, for example, to automatically be of type Object.

See also: DefBool, DefDate, DefDbL, DefInt, DefLng, DefObj, and DefVar.

Syntax:

DefObj Characterrange1[, Characterrange2[,...]]

Parameter:

Characterrange: Letters specifying the range of variables for which to set a default data type.

Example:

See ExampleDefBool

23.49. DefVar Statement

Summary

Set the default variable type, according to a letter range, if no type-declaration character or keyword is specified. You can allow all variables that start with a “v”, for example, to automatically be of type Variant.

See also: DefBool, DefDate, DefDbL, DefInt, DefLng, DefObj, and DefVar.

Syntax:

DefVar Characterrange1[, Characterrange2[,...]]

Parameter:

Characterrange: Letters specifying the range of variables for which to set a default data type.

Example:

See ExampleDefBool

23.50. Dim Statement

Summary:

Declare variables. The type of each variable is declared separately and the default type is Variant. The following defines a, b, and c to be of type Variant and d to be of type Date.

```
Dim a, b, c, d As Date
```

A variables type may also be specified by the use of an appended character. This is mentioned in the section on variable types.

Dim is used to declare local variables within Subs. Global variables outside of Subs are declared with the PUBLIC or PRIVATE statements.

Unless the “Option Explicit” statement is present, non-array variables may be used without declaration and their default type is variant.

Single and multi-dimensional arrays are supported.

See also: Public, Private, ReDim

Syntax:

```
[ReDim]Dim Name_1 [(start To end)] [As Type][, Name_2 [(start To end)] [As Type][,...]]
```

Parameter/Keyword:

Name_i: Variable or array name.

Start, End: Integer constants from -32768 to 32767 an inclusive array range. At the procedural level, the ReDim allows numerical expressions so these may be reset at runtime.

Type: Key word used to declare the data type of a variable. The supported variable types are Boolean, Currency, Date, Double, Integer, Long, Object, Single, String, or Variant.

Example:

```
Sub ExampleDim
    Dim s1 As String, i1 As Integer, i2%
    Dim a1(5) As String REM 0 to 6
    Dim a2(3, -1 To 1) As String REM (0 to 3, -1 to 1)
    Const sDim as String = " Dimension:"
    For i1 = LBound(a2(), 1) To UBound(a2(), 1)
        For i2 = LBound(a2(), 2) To UBound(a2(), 2)
            a2(i1, i2) = Str(i1) & ":" & Str(i2)
        Next
    Next
    For i1 = LBound(a2(), 1) To UBound(a2(), 1)
        For i2 = LBound(a2(), 2) To UBound(a2(), 2)
            Print a2(i1, i2)
        Next
    Next
End Sub
```

23.51. DimArray Function

Summary:

Create a Variant array. This works just like Dim(Argument list). If no arguments are present, an empty array is created. If parameters are present, a dimension is created for each parameter.

See also: Array

Syntax:

DimArray (Argument list)

Return value:

Variant array

Parameter:

Argument list: Optional comma separated list of integers.

Example:

DimArray(2, 2, 4) is the same as DIM a(2, 2, 4)

23.52. Dir Function

Summary:

Return the name of a file, directory, or all files and folders of a drive or directory that match the specified search path. Possible uses include verifying that a file or directory exists, or to list the files and folders in a specific directory.

If no files or directories match, a zero length string is returned. Repeatedly call the Dir function until a zero length string is returned to iterate through all of the return values.

The volume and directory attributes are exclusive meaning that if you request one of these it is the only thing that you will retrieve. I can not determine which has greater precedence because as of 1.0.3, the volume attribute does nothing.

The attributes are a subset of those available in GetAttr.

See Also: GetAttr

Syntax:

Dir [(Text As String[, Attrib As Integer])]

Return value:

String

Parameter:

Text: String that specifies the search path, directory or file. The URL notation is accepted.

Attrib: Integer expression for file attributes. The Dir function returns only files or directories that match the specified attributes. Add the attributes to combine them.

Attribute	Description
0	Normal files.
2	Hidden files.
4	System files.
8	Returns the name of the volume (Exclusive).
16	Return directories (Exclusive).

Example:

```

Sub ExampleDir
    REM Displays all files and directories
    Dim sFile as String, sPath As String
    Dim sDir as String, sValue as String
    Dim sFullPath As String
    Dim iFile as Integer
    sFile= "Files: "
    sDir="Directories:"
    iFile = 0
    sPath = CurDir
    REM 0 : Normal files.
    REM 2 : Hidden files.
    REM 4 : System files.
    REM 8 : Returns the name of the volume
    REM 16 : Returns the name of the directory only.
    REM This example will ONLY return directories because the 16 is included!
    REM Remove the 16 and you will receive the files
    sValue = Dir$(sPath & getPathSeparator() & "*", 0 + 2 + 4 + 16)
    Do
        If sValue <> "." and sValue <> ".." Then
            sFullPath = sPath + getPathSeparator() + sValue
            If NOT FileExists(sFullPath) Then
                Print sFullPath & " does not exist"
            ElseIf (GetAttr( sPath & getPathSeparator() & sValue) AND 16) > 0 Then
                REM here the directories
                sDir = sDir & chr(13) & sValue
            Else
                REM here the files
                If iFile Mod 3 = 0 Then sFile = sFile + Chr(13)
                iFile = iFile + 1
                sFile = sFile + sValue &"; "
            End If
        End If
    Loop While sValue <> ""
End Sub

```

```

    sValue = Dir$
    Loop Until sValue = ""
    MsgBox sDir,0,sPath
    MsgBox "" & iFile & " " & sFile,0,sPath
End Sub

```

Tip	The method getPathSeparator() is included even though it does not appear in the help list. I have not yet found where it is defined, but it is used in the distributed tools and the help.
Warning	Some operating systems include the directories “.” and “..” which refer to the current directory and the parent directory respectively. If you write code that traverses a directory, you probably do not want to follow these or you will be stuck in an infinite loop.
Warning	When you obtain a directory listing, files are not returned, even if the on-line help example seems to indicate that it does.

23.53. Do...Loop Statement

Summary:

Construct to repeat statements.

See Also: Loop Control Page 403.

Syntax:

Do [{While | Until} condition = True]

statement block

[Exit Do]

statement block

Loop

Syntax:

Do

statement block

[Exit Do]

statement block

Loop [{While | Until} condition = True]

23.54. End Statement

Summary:

Mark the end of a procedure, block or even a subroutine or function.

See Also: Exit

Syntax:

<i>Form</i>	<i>Function</i>
End	End by itself ends program execution. It may be entered anywhere. This is optional.
End Function	Mark the end a function
End If	Mark the end of an If...Then...Else block.
End Select	Mark the end of a Select Case block.
End Sub	Mark the end a subroutine.

Example:

```
Sub ExampleEnd
    Dim s As String
    s = InputBox ("Enter an integer :", "White Space Checker")
    If IsWhiteSpace(Val(s)) Then
        Print "ASCII " + s + " is white space"
    Else
        Print "ASCII " + s + " is not white space"
    End If
End Sub
Print "I will never get here"
End Sub

Function IsWhiteSpace(iChar As Integer) As Boolean
    Select Case iChar
        Case 9, 10, 13, 32, 160
            IsWhiteSpace = True
        Case Else
            IsWhiteSpace = False
    End Select
End Function
```

23.55. Environ Function

Summary:

Return the value of an environment variable. Environment variables are operating system dependent. On a Macintosh computer the function returns an empty string.

Syntax:

Environ (Environment As String)

Return value:

String

Parameter:

Environment: Environment variable for which to return the value.

Example:

```
Sub ExampleEnviron
    MsgBox "Path = " & Environ("PATH")
End Sub
```

23.56. EOF Function

Summary:

Use EOF to avoid trying to read past the end of a file. When the end of the file is reached, EOF returns True (-1).

See also Open, Close, Kill, and FreeFile

Syntax:

EOF (intexpression As Integer)

Return value:

Boolean

Parameter:

Intexpression: Integer expression that evaluates to the number of an open file.

Example:

```
REM This example modified from the on-line help.
REM The on-line help example does not work.
Sub ExampleEof
    Dim iNumber As Integer
    Dim aFile As String
    Dim sMsg as String, sLine As String
    aFile = "c:\DeleteMe.txt"
    iNumber = Freefile
    Open aFile For Output As #iNumber
    Print #iNumber, "First line of text"
    Print #iNumber, "Another line of text"
    Close #iNumber
    iNumber = Freefile
    Open aFile For Input As iNumber
    While Not Eof(iNumber)
        Line Input #iNumber, sLine
        If sLine <>"" Then
            sMsg = sMsg & sLine & chr(13)
        End If
    Wend
    Close #iNumber
    MsgBox sMsg
End Sub
```

23.57. EqualUnoObjects Function

Summary:

Test if the two UNO objects represent the same UNO object instance.

Syntax:

EqualUnoObjects(oObj1, oObj2)

Return value:

Boolean

Example:

```
Sub ExampleEqualUnoObjects
    Dim oIntrospection, oIntro2, Struct2
    REM Copy of objects -> same instance
    oIntrospection = CreateUnoService( "com.sun.star.beans.Introspection" )
    oIntro2 = oIntrospection
    Print EqualUnoObjects( oIntrospection, oIntro2 )
    REM Copy of structs as value -> new instance
    Dim Struct1 As New com.sun.star.beans.Property
    Struct2 = Struct1
    Print EqualUnoObjects( Struct1, Struct2 )
End Sub
```

23.58. EQV Operator

Summary:

Calculates the logic equivalence of two expressions. In a bit-wise comparison, the EQV operator sets the corresponding bit in the result only if a bit is set in both expressions, or neither expression. The standard mathematical precedence is used as shown on page 419 and the operation table is shown below.

x	y	x EQV y
TOSI	TOSI	TOSI
TOSI	EPÄTOSI	EPÄTOSI
EPÄTOSI	TOSI	EPÄTOSI
EPÄTOSI	EPÄTOSI	TOSI
1	1	1
1	0	0
0	1	0
0	0	1

Syntax:

Result = Expression1 EQV Expression2

Parameter:

Result: Numeric variable to contain the result of the comparison.

Expression1, expression2 : Expressions to be compared.

Example:

```
Sub ExampleEQV
    Dim vA as Variant, vB as Variant, vC as Variant, vD as Variant
    Dim vOut as Variant
    vA = 10: vB = 8: vC = 6: vD = Null
    vOut = vA > vB EQV vB > vC REM returns -1
    Print vOut
    vOut = vB > vA EQV vB > vC REM returns -1
    Print vOut
    vOut = vA > vB EQV vB > vD REM returns 0
    Print vOut
    vOut = (vB > vD EQV vB > vA) REM returns -1
    Print vOut
    vOut = vB EQV vA REM returns -1
End Sub
```

23.59. *Erl Function*

Summary:

Return the line number in which an error occurred during program execution.

See Also: Err

Syntax:

Erl

Return value:

Integer

Example:

```
Sub ExampleErl
    On Error GoTo ErrorHandler
    Dim iVar as Integer
    iVar = 0
    iVar = 4 / iVar
    Exit Sub
ErrorHandler:
    REM Error 11 : Division by Zero
    REM In line: 8
    REM ....
    MsgBox "Error " & err & ": " & error$ + chr(13) + _
```



```

    "In line : " + Erl + chr(13) + Now , 16 , "An error occurred"
End Sub

```

23.60. Err Function

Summary:

Return the error number of the last error.

See Also: Erl

Syntax:

Err

Return value:

Integer

Example:

```

Sub ExampleErr
    On Error GoTo ErrorHandler
    Dim iVar as Integer
    iVar = 0
    iVar = 4 / iVar
    Exit Sub
ErrorHandler:
    REM Error 11 : Division by Zero
    REM In line: 8
    REM ....
    MsgBox "Error " & err & ": " & error$ + chr(13) + _
        "In line : " + Erl + chr(13) + Now , 16 , "An error occurred"
End Sub

```

23.61. Error statement does not work as indicated.

Summary:

Simulates the occurrence of an error during program execution. This does NOT work!

Syntax:

Error errornumber As Integer

Return value:

Integer

Parameter:

Errornumber: Integer expression that specifies the number of the error to be simulated.

Example:

23.62. Error Function

Summary:

Returns the error message corresponding to a given error code.

Syntax:

Error (Expression)

Return value:

String

Parameter:

Expression: Optional integer containing the error message for an error number. If this is missing, the most recent error message is returned.

Example:

```
Sub ExampleError
    On Error GoTo ErrorHandler
    Dim iVar as Integer
    iVar = 0
    iVar = 4 / iVar
    Exit Sub
ErrorHandler:
    REM Error 11 : Division by Zero
    REM In line: 8
    REM ....
    MsgBox "Error " & err & ": " & error$ + chr(13) + _
        "In line : " + Erl + chr(13) + Now , 16 , "An error occurred"
End Sub
```

23.63. Exit Statement

Summary:

The Exit statement is used to leave a Do...Loop, For...Next, Function or Subroutine construct. In other words, I can immediately exit any of these constructs. If I am inside of a Sub and I determine that the arguments are bad, I can immediately leave the Sub.

See Also: End

Syntax:

Form	Function
Exit Do	Exit the inner-most Do...Loop.
Exit For	Exit the inner-most For...Next loop.
Exit Function	Exit a function and continue execution following the function call.
Exit Sub	Exit a subroutine and continue execution following

<i>Form</i>	<i>Function</i>
	the subroutine call.

Example:

```

Sub ExampleExit
    Dim sReturn As String
    Dim sListArray(10) as String
    Dim siStep as Single
    REM Build array ("B", "C", ..., "L")
    For siStep = 0 To 10 REM Fill array with test data
        sListArray(siStep) = chr(siStep + 66)
    Next siStep
    sReturn = LinSearch(sListArray(), "M")
    Print sReturn
    Exit Sub REM This really is a useless statement!
End Sub

REM Returns the index of the entry or (LBound(sList()) - 1) if not found
Function LinSearch( sList(), sItem As String ) as integer
    Dim iCount As Integer
    REM LinSearch searches a TextArray:sList() for a TextEntry:
    For iCount=LBound(sList()) To UBound(sList())
        If sList(iCount) = sItem Then
            LinSearch = iCount
            Exit Function REM Probably a good use of Exit here!
        End If
    Next
    LinSearch = LBound(sList()) - 1
End Function

```

23.64. Exp Function

Summary:

Return the base of the natural logarithm ($e = 2.718282$) raised to a power.

See Also: Log

Syntax:

Exp (Number)

Return value:

Double

Parameter:

Number: Any numeric expression.

Example:

```

Sub ExampleExp
    Dim d as Double, e As Double

```

```

e = Exp(1)
Print "e = " & e
Print "ln(e) = " & Log(e)
Print "2*3 = " & Exp(Log(2.0) + Log(3.0))
Print "2^3 = " & Exp(Log(2.0) * 3.0)
end sub

```

23.65. FileAttr Function

Summary:

The first purpose of the FileAttr function is to determine the access mode of file opened with the Open statement. Setting the second parameter to 1, requests this value.

Value	Access Mode Description
1	INPUT (file open for input)
2	OUTPUT (file open for output)
4	RANDOM (file open for random access)
8	APPEND (file open for appending)
32	BINARY (file open in binary mode)

The second purpose of the FileAttr function is to determine the MS-DOS file attribute of a file that was opened with the Open statement. This value is operating system dependent. Setting the second parameter to 2, requests this value.

Warning The file attribute is operating system dependent. If the operating system is a 32-Bit-Version, it is not possible to use the FileAttr-Function to determine the MS-Dos file attribute so a zero is returned.

See Also: Open

Syntax:

FileAttr (FileNumber As Integer, Attribute As Integer)

Return value:

Integer

Parameter:

FileNumber: Number used in the Open statement.

Attribute: Integer indicating what information to return. 1 requests the access mode and 2 requests the file access number.

Example:

```

Sub ExampleFileAttr
    Dim iNumber As Integer
    iNumber = Freefile

```

```

Open "file:///c:/data.txt" For Output As #iNumber
Print #iNumber, "Random Text"
MsgBox AccessModes(FileAttr(#iNumber, 1)),0,"Access mode"
MsgBox FileAttr(#iNumber, 2),0,"File attribute"
Close #iNumber
End Sub
Function AccessModes(x As Integer) As String
Dim s As String
s = ""
If (x AND 1) <> 0 Then s = "INPUT"
If (x AND 2) <> 0 Then s = "OUTPUT"
If (x AND 4) <> 0 Then s = s & " RANDOM"
If (x AND 8) <> 0 Then s = s & " APPEND"
If (x AND 32) <> 0 Then s = s & " BINARY"
AccessModes = s
End Function

```

23.66. FileCopy Statement

Summary:

Copy a file. You can not copy a file that is open.

Syntax:

FileCopy TextFrom As String, TextTo As String

Parameter:

TextFrom: String specifying source file name.

TextTo: String specifying the destination file name.

Example:

```

Sub ExampleFilecopy
Filecopy "c:\Data.txt", "c:\Temp\Data.sav"
End Sub

```

23.67. FileDateTime Function

Summary:

Return a string that with the date and time a file was created or last modified, returned in an system dependent format "MM/DD/YYYY HH:MM:SS" on my computer. Consider using the DateValue function with this string.

Syntax:

FileDateTime(Text As String)

Return value:

String

Parameter:

Text: File specification (no wild-cards allowed). URL notation is allowed.

Example:

```
Sub ExampleFileDateTime
    REM 04/23/2003 19:30:03
    MsgBox FileDateTime("file://localhost/C:/macro.txt")
End Sub
```

23.68. FileExists Function

Summary:

Determine if a file or a directory exists.

Syntax:

FileExists(Filename As String | DirectoryName As String)

Return value:

Boolean

Parameter:

Filename | DirectoryName: File or directory specification (no wild-cards allowed).

Example:

```
Sub ExampleFileExists
    MsgBox FileExists("C:\autoexec.bat")
    MsgBox FileExists("file://localhost/c:/macro.txt")
    MsgBox FileExists("file:///d:/private")
End Sub
```

23.69. FileLen Function

Summary:

Determine the length of a file. If the file is currently open, the file length before it was opened is returned. To determine the current file length of an open file, use the Lof function instead.

Syntax:

FileLen(Filename As String)

Return value:

Long

Parameter:

Filename: File specification (no wild-cards allowed).

Example:

```
Sub ExampleFileExists
```

```

MsgBox FileLen("C:\autoexec.bat")
MsgBox FileLen("file://localhost/c:/macro.txt")
End Sub

```

23.70. FindObject Function

Summary:

Give a variable name and it will return a reference to the object. See FindPropertyObject. Running the code shown below should demonstrate that this does not work very well.

```

Sub TestTheThing
    Dim oTst As Object
    Dim oDoc As Object
    oTst = FindObject("oDoc")

    REM yes
    If oTst IS oDoc Then Print "oTst and oDoc are the same"

    oDoc = ThisComponent
    oTst = FindObject("oDoc")
    REM no
    If oTst IS oDoc Then Print "oTst and oDoc are the same"
    REM no
    If oTst IS ThisComponent Then Print "oTst and ThisComponent are the same"
    REM yes
    If oDoc IS ThisComponent Then Print "oDoc and ThisComponent are the same"

    oDoc = ThisComponent
    oTst = FindObject("ThisComponent")
    REM yes
    If oTst IS oDoc Then Print "oTst and oDoc are the same"
    REM yes
    If oTst IS ThisComponent Then Print "oTst and ThisComponent are the same"
    REM yes
    If oDoc IS ThisComponent Then Print "oDoc and ThisComponent are the same"

    REM this shows ThisComponent
    RunSimpleObjectBrowser(oTst)
    oDoc = ThisComponent
    '?? March 8, 2009
    'Document Info object is deprecated, convert to use document properties.
    oTst = ThisComponent.DocumentInfo
    oTst = FindPropertyObject(oDoc, "DocumentInfo")
    If IsNull(oTst) Then Print "Is Null"
    If oTst IS ThisComponent.DocumentInfo Then Print "They are the same"
End Sub

```

23.71. FindPropertyObject Function

Summary:

Now I do have an idea and boy is this stuff strange. Oh yeah, and it does not work very well. In other words, consider it broken!

An object contains data objects. For example, a spreadsheet document has a property called DrawPages that I can reference directly with the command ThisComponent.DrawPages. I can use FindPropertyObject to obtain a reference to this object.

```
obj = FindPropertyObject(ThisComponent, "DrawPages")
```

I can now access the DrawPages object with the variable obj. I have found this to be buggy!

Example:

```
Sub StrangeThingsInStarBasic

Dim oSBObj1 As Object
Dim oSBObj2 As Object
Dim oSBObj3 As Object

Set oSBObj1 = Tools
RunSimpleObjectBrowser(oSBObj1)
'we also have a Name property!!
Print oSBObj1.Name ' @SBRTL ?? what is it?

'apropos...
Set oSBObj2 = FindObject("Gimmicks")
Print oSBObj2.Name ' @SBRTL again...

'you can change this name prop, but it does nothing
' oSBObj2.Name = "Ciao" : Print oSBObj2.Name
' oSBObj2.Name = "@SBRTL" : Print oSBObj2.Name

'need Gimmicks library Loaded, now
GlobalScope.BasicLibraries.LoadLibrary("Gimmicks")

'other old, deprecated, undocumented, almost broken stuff....

'userfields is a module in the Gimmicks library
Set oSBObj3 = FindPropertyObject(oSBObj2, "Userfields")
Print (oSBObj3 Is Gimmicks.Userfields)

'need Gimmicks library Loaded, now
GlobalScope.BasicLibraries.LoadLibrary("Gimmicks")

'the StartChangesUserfields function is in the module Userfields
'a fully qualified call!
oSBObj2.Userfields.StartChangesUserfields
```


End Sub

23.72. Fix Function

Summary:

Return the integer portion of a numeric expression by removing the fractional part.

See Also: CInt, Int

Syntax:

Fix(Expression)

Return value:

Double

Parameter:

Expression: Number for which to return the integer portion.

Example:

```
sub ExampleFix
    Print Fix(3.14159)    REM returns 3.
    Print Fix(0)         REM returns 0.
    Print Fix(-3.14159)  REM returns -3.
End Sub
```

23.73. For...Next Statement

Summary:

Construct to repeat statements with an auto-incrementing counter.

See Also: For....Next on Page 402.

Syntax:

For counter=start To end [Step step]

statement block

[Exit For]

statement block

Next [counter]

23.74. Format Function

Summary:

Convert a number to a string formatted according to the optional format string. Multiple formats may be included in a single format string. Each individual format is separated by a

“,”. The first format is used for positive numbers, the second is for negative numbers, and the third is for zero. If only one format code is present, it applies to all numbers.

Code	Description
0	If Number has a digit at the position of the 0 in the format code, the digit is displayed; otherwise a zero appears. This means that leading and trailing zeros are displayed, leading digits are not truncated, and trailing decimals are rounded.
#	This works like the 0, but leading and trailing zeros are not displayed.
.	The decimal placeholder determines the number of decimal places to the left and right of the decimal separator.
%	Multiply the number by 100 and insert the percent sign (%) where it appears in the format code.
E- E+ e- e+	If the format code contains at least one digit placeholder (0 or #) to the right of the symbol, the number is formatted in the scientific notation. The letter E or e is inserted between the number and the exponent. The number of placeholders for digits to the right of the symbol determines the number of digits in the exponent. If the exponent is negative, a minus sign is displayed directly before an exponent. If the exponent is positive, a plus sign is only displayed before exponents with E+ or e+.
,	The comma is a placeholder for the thousands separator. It separates thousands from hundreds in a number with at least four digits. The thousands delimiter is displayed if the format code contains the placeholder surrounded by digit placeholders (0 or #).
- + \$ () space	(+), minus (-), dollar (\$), space, or brackets entered directly in the format code are displayed as the literal character.
\	The backslash displays the next character in the format code. In other words, it prevents the next character from being seen as a special character. The backslash is not displayed, unless you enter a double backslash (\\) in the format code. Characters that must be preceded by a backslash in the format code in order to be displayed as literal characters are the date- and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, /, :), numeric-formatting characters (#, 0, %, E, e, comma, period) and string-formatting characters (@, &, <, >, !). You may also enclose characters in double quotes.
General Number	Numbers are displayed as entered.
Currency	A dollar sign is placed in front of the number; negative numbers are enclosed in parentheses. Two decimals are displayed. (Actually, this is locale specific)
Fixed	At least one digit is displayed in front of the decimal separator. Two decimals are displayed.
Standard	Displays numbers with a locale specific thousands separator. Two decimals are displayed.
Scientific	Displays numbers in scientific notation. Two decimals are displayed.

Warning	No conversion takes place if the parameter is not a number (such as if it is a String) and an empty string is returned.
Warning	As of version 1.0.3.1, Format(123.555, “.##”) produces “.12356” which I consider a bug. Changing the format to “#.##” fixes this problem. Always use a leading “#” or “0” as appropriate. Scientific notation is just wrong (broken) Currency is placing the dollar sign to the right. Currently not able to escape special characters.

Syntax:

Format (Number [, Format As String])

Return value:

String

Parameter:

Number: Numeric expression to convert to a formatted string.

Format: Desired format. If omitted, the Format function works like the Str function.

Example:

```

Sub ExampleFormat
    MsgBox Format(6328.2, "##,##0.00")      REM = 6,328.20
    MsgBox Format(123456789.5555, "##,##0.00") REM = 123,456,789.56
    MsgBox Format(0.555, ".##")            REM .56
    MsgBox Format(123.555, "#.##")         REM 123.56
    MsgBox Format(0.555, "0.##")           REM 0.56
    MsgBox Format(0.1255555, "%#.##")      REM %12.56
    MsgBox Format(123.45678, "##E-####")   REM 12E1
    MsgBox Format(.0012345678, "0.0E-####") REM 1.2E3 (broken)
    MsgBox Format(123.45678, "#.e-####")   REM 1e2
    MsgBox Format(.0012345678, "#.e-####") REM 1e3 (broken)
    MsgBox Format(123.456789, "#.## is ###") REM 123.45 is 679 (strange)
    MsgBox Format(8123.456789, "General Number") REM 8123.456789
    MsgBox Format(8123.456789, "Fixed")     REM 8123.46
    MsgBox Format(8123.456789, "Currency") REM 8,123.46$ (broken)
    MsgBox Format(8123.456789, "Standard") REM 8,123.46
    MsgBox Format(8123.456789, "Scientific") REM 8.12E03
    MsgBox Format(0.00123456789, "Scientific") REM 1.23E03 (broken)
End Sub

```

23.75. FreeFile Function

Summary:

Return the next available file number for opening a file. This ensures that the file number that you use is not already in use.

See also Open, EOF, Kill, and Close.

Syntax:

FreeFile

Return value:

Integer

Example:

See the example for Close.

23.76. FreeLibrary Function

Summary:

Release a DLL loaded by a Declare statement. The DLL will automatically reload if one of its functions is called. Only DLLs loaded during the Basic runtime may be freed.

See Also: Declare

Syntax:

FreeLibrary (LibName As String)

Parameter:

LibName: Name of the DLL.

Example:

```
Declare Sub MyMessageBeep Lib "user32.dll" Alias "MessageBeep" ( long )
Sub ExampleDeclare
    Dim lValue As Long
    lValue = 5000
    MyMessageBeep( lValue )
    FreeLibrary("user32.dll" )
End Sub
```

23.77. Function Statement

Summary:

Define a user defined function as opposed to a subroutine. Functions can return values, subroutines can not.

See Also: Sub

Syntax:

Function Name[(VarName1 [As Type][, VarName2 [As Type][,...]])] [As Type]
statement block

[Exit Function]

statement block

End Function

Return value:

What ever type is declared

Example:

```
Function IsWhiteSpace(iChar As Integer) As Boolean
    Select Case iChar
    Case 9, 10, 13, 32, 160
        IsWhiteSpace = True
    Case Else
        IsWhiteSpace = False
    End Select
End Function
```

23.78. Get Statement

Summary:

Reads a record from a relative file, or a sequence of bytes from a binary file, into a variable. If the position parameter is omitted, data is read from the current position in the file. For files opened in binary mode, the position is the byte position in the file.

See Also: PUT

Syntax:

Get [#] FileNumber As Integer, [Position], Variable

Parameter:

FileNumber: Integer expression that determines the file number. I use FreeFile to get this.

Position: For files opened in Random mode, this is the record number to be read.

Variable: Variable to be read. The Object variable type may not be used here.

Example:

```
'?? This is broken!
Sub ExampleRandomAccess2
    Dim iNumber As Integer, aFile As String
    Dim sText As Variant REM Must be a variant
    aFile = "c:\data1.txt"
    iNumber = Freefile
    Open aFile For Random As #iNumber Len=5
    Seek #iNumber,1 REM Position at beginning
    Put #iNumber,, "1234567890" REM Fill line with text
    Put #iNumber,, "ABCDEFGHIJ"
```

```

Put #iNumber,, "abcdefghij"
REM This is how the file looks now!
REM 08 00 0A 00 31 32 33 34 35 36 37 38 39 30 08 00 ....1234567890..
REM 0A 00 41 42 43 44 45 46 47 48 49 4A 08 00 0A 00 ..ABCDEFGHJIJ...
REM 61 62 63 64 65 66 67 68 69 6A 00 00 00 00 00 00 abcdefghij
REM
Seek #iNumber,1
Get #iNumber,,sText
Print "on open:" & sText
Close #iNumber
iNumber = Freefile
Open aFile For Random As #iNumber Len=5
Get #iNumber,,sText
Print "reopened: " & sText
Put #iNumber,, "ZZZZZ"
Get #iNumber,1,sText
Print "anoter get "& sText
Get #iNumber,1,sText
Put #iNumber,20,"This is the text in record 20"
Print Lof(#iNumber)
Close #iNumber
End Sub

```

23.79. GetAttr Function

Summary:

Return a bit pattern identifying the file type. The attributes are a superset of those used in the Dir function.

Attribute	Description
0	Normal file.
1	Read-Only file
2	Hidden files.
4	System file.
8	Volume name.
16	Directory.
32	Archive bit (file changed since last backed up)

See Also: Dir

Warning Broken as of 1.0.3.1. Test with the version that you use.

Syntax:

GetAttr (Text As String)

Return value:

Integer

Parameter:

Text: String expression containing an unambiguous file specification. URL notation is valid.

Example:

```
Sub ExampleGetAttr
    REM Should say " Read-Only Hidden System Archive"
    REM says " Read-Only"
    Print FileAttributeString(GetAttr("C:\IO.SYS"))
    REM Should say " Archive" says "Normal"
    Print FileAttributeString(GetAttr("C:\AUTOEXEC.BAT"))
    REM "Directory"
    Print FileAttributeString(GetAttr("C:\WINDOWS"))
End Sub
Function FileAttributeString(x As Integer) As String
    Dim s As String
    If (x = 0) Then
        s = "Normal"
    Else
        s = ""
        If (x AND 16) <> 0 Then s = "Directory"
        If (x AND 1) <> 0 Then s = s & " Read-Only"
        If (x AND 2) <> 0 Then s = s & " Hidden"
        If (x AND 4) <> 0 Then s = s & " System"
        If (x AND 8) <> 0 Then s = s & " Volume"
        If (x AND 32) <> 0 Then s = s & " Archive"
    End If
    FileAttributeString = s
End Function
```

23.80. *GetProcessServiceManager Function*

Summary:

Obtain the central UNO service manager. This is required when you must instantiate a service using CreateInstance with arguments.

?? find a better example than this! Show an example that takes an argument!

Syntax:

oServiceManager = GetProcessServiceManager()

Return value:**Example:**

```

oServiceManager = GetProcessServiceManager()
oIntrospection =
oServiceManager.CreateInstance("com.sun.star.beans.Introspection");
this is the same as the following statement:
oIntrospection = CreateUnoService("com.sun.star.beans.Introspection")

```

23.81. GetSolarVersion Function

Summary:

Return the internal build number of the current OpenOffice.org version. You could write your macro to work around known bugs based on different versions. Unfortunately, the function GetSolarVersion frequently stays the same even when the versions change. Version 1.0.3.1 returns “641” and 1.1RC3 returns 645, but this is not always enough granularity. The following macro returns the actual OOo version.

```

Function OOVersion() As String
'Retreives the running OOo version
'Author : Laurent Godard
'e-mail : listes.godard@laposte.net
'
Dim oSet, oConfigProvider
Dim oParm(0) As New com.sun.star.beans.PropertyValue
Dim sProvider$, sAccess$
sProvider = "com.sun.star.configuration.ConfigurationProvider"
sAccess = "com.sun.star.configuration.ConfigurationAccess"
oConfigProvider = createUnoService(sProvider)
oParm(0).Name = "nodepath"
oParm(0).Value = "/org.openoffice.Setup/Product"
oSet = oConfigProvider.CreateInstanceWithArguments(sAccess, oParm())

OOVersion=oSet.getbyname("ooSetupVersion")
End Function

```

Syntax:

s = GetSolarVersion()

Return value:

String

Example:

```

Sub ExampleGetSolarVersion
REM as of 1.0.3.1, this is "641"
Print GetSolarVersion()
End Sub

```

23.82. GetSystemTicks Function

Summary:

Return the system ticks provided by the operating system. The number of system ticks returned within a certain time frame is always dependent on the operating system.

Syntax:

GetSystemTicks()

Return value:

Long

Example:

This example will attempt to measure how many ticks per second. On Windows XP and version 1.0.3.1 of OpenOffice.org, I see 1000 ticks per second.

```
Sub ExampleGetSystemTicks
    Dim lTick As Long, lMillisToWait As Long
    Dim lSecsToWait As Long, lTicksPerSec As Long
    lSecsToWait = 60
    lMillisToWait = lSecsToWait * 1000
    lTick = GetSystemTicks()
    wait(lMillisToWait)
    lTick = (GetSystemTicks() - lTick)
    lTicksPerSec = lTick / lSecsToWait
    MsgBox "Each second has about " & lTicksPerSec & " Ticks Per Second"
End Sub
```

23.83. GlobalScope Statement

Summary:

Basic macros and dialogs are organized in libraries. A library may contain modules and/or dialogs. In Basic, the library container is called “BasicLibraries” and in dialogs the container is called “DialogLibraries”. Although both library containers exist at both the application and the document level, in basic they are automatically loaded but not in the document. To call these global library containers from within a document, you must use the keyword GlobalScope.

Syntax:

GlobalScope

Example:

```
' calling Dialog1 in the document library Standard
oDlgDesc = DialogLibraries.Standard.Dialog1
' calling Dialog2 in the application library Library1
oDlgDesc = GlobalScope.DialogLibraries.Library1.Dialog2
```

23.84. GoSub Statement

Summary:

Transfer macro execution to a label within the current Sub or Function. The statements following the label are executed until the next Return statement; thereafter the program continues with the statement following the GoSub statement.

See Also: Section on flow control which explains why GoSub is generally avoided.

Tip	GoSub is a persistent remnant from old Basic dialects, retained for compatibility. GoSub is strongly discouraged because it tends to produce unreadable code. Subs or Functions are preferable.

Syntax:

```
Sub/Function
  REM arbitrary statements here
  GoSub Label
  REM arbitrary statements here
  GoSub Label
Exit Sub/Function

Label:
  statement block
Return
End Sub/Function
```

Example:

```
Sub ExampleGoSub
  Print "Before the gosub"
  GoSub SillyLabel
  Print "After the gosub"
Exit Sub
SillyLabel:
  Print "After Silly Label"
Return
End Sub
```

23.85. GoTo Statement

Summary:

Transfer macro execution to a label within the current Sub or Function. The statements following the label are executed.

See Also: Section on flow control which explains why GoTo is generally avoided.

Tip	GoTo is a persistent remnant from old Basic dialects, retained for compatibility. GoTo is strongly discouraged because it tends to produce unreadable code. Subs or Functions are preferable.

Syntax:

```
Sub/Function
  REM arbitrary statements here
```

```

GoTo Label
REM arbitrary statements here
GoTo Label
Exit Sub/Function
Label:
    statement block
End Sub/Function

```

Example:

```

Sub ExampleGoTo
    Print "Before the goto"
    GoTo SillyLabel
    Print "After the goto"    REM Never executed
    Exit Sub                 REM Never executed
SillyLabel:
    Print "After Silly Label"
End Sub

```

23.86. Green Function

Summary:

Colors are represented by a long integer. Return the green component of the specified color code. See also RGB, Red, and Blue.

Syntax:

Green(Color As Long)

Return value:

Integer in the range of 0 to 255.

Parameter:

Color value: Long integer expression representing a color.

Example:

```

Sub ExampleColor
    Dim lColor As Long
    lColor = RGB(255,10,128)
    MsgBox "The color " & lColor & " consists of:" & Chr(13) & _
        "Red = " & Red(lColor) & Chr(13) & _
        "Green= " & Green(lColor) & Chr(13) & _
        "Blue= " & Blue(lColor) & Chr(13) , 64,"Colors"
End Sub

```

23.87. HasUnoInterfaces Function

Summary:

Test if a Basic UNO object supports specified UNO interfaces. Returns true only if all of the specified UNO interfaces are supported.

Syntax:

HasUnoInterfaces(oTest, UNO-Interface-Name 1 [, UNO-Interface-Name 2, ...])

Return value:

Boolean

Parameter:

oTest : The Basic UNO object to test.

UNO-Interface-Name: list of UNO interface names.

Example:

```
Sub CloseOpenDocument
    If HasUnoInterfaces(oDoc, "com.sun.star.util.XCloseable") Then
        oDoc.close(true)
    Else
        oDoc.dispose
    End If
End Sub
```

23.88. Hex Function

Summary:

Returns a string that represents the hexadecimal value of a number. If the parameter is not a number, it is converted into a number if possible.

Syntax:

Hex(Number)

Return value:

String

Parameter:

Number: Numeric expression to be converted to a hexadecimal number. May be a string.

Example:

```
Sub ExampleHex
    Dim i1%, i2%, iNum%, s$, sFormat$, sTemp$
    iNum = 0
    s = ""
    For i1=0 To 15
        For i2=0 To 15
            s = s & " " & PrependChar(Hex(iNum), "0", 2)
            iNum = iNum + 1
        Next
        s = s & Chr(13)
    Next
    MsgBox s, 64, "Hex Table"
```

```

    Print Hex("64")
End Sub
Function PrependChar(s$, sPrependString$, iTotLen%) As String
    If Len(s) < iTotLen Then
        PrependChar = String(iTotLen - Len(s), sPrependString) & s
    Else
        PrependChar = s
    End If
End Function

```

23.89. Hour Function

Summary:

Return the hour from a time value generated by TimeSerial or TimeValue.

Syntax:

Hour(Number)

Return value:

Integer

Parameter:

Number: Numeric expression that contains a serial time value.

Example:

```

Sub ExampleHour
    Print "The current hour is " & Hour( Now )
    Print Hour(TimeSerial(14,08,12))
    Print Hour(TimeValue("14:08:12"))
End Sub

```

23.90. If Statement

Summary:

Defines one or more statement blocks to be executed if a given condition is True. Although you can use GoTo or GoSub to jump out of an If block, you can not jump into an If block.

Syntax:

```

If condition=true Then
    Statementblock
[ElseIf condition=true Then]
    Statementblock
[Else]
    Statementblock
End If

```

Syntax:

```
If condition=True Then Statement
If condition=False Then Statement
```

Example:

```
Sub ExampleIf
  Dim i%
  i% = 4
  If i < 5 Then
    Print "i is less than 4"
    If i = 4 Then Print "i is 4"
    If i < 3 Then
      Print "i is less than 3"
    End If
  ElseIf i = 5 Then
    Print "i is 5"
  Else
    Print "i is greater than 5"
  End If
End Sub
```

23.91. IIf Statement

Summary:

Return one of two possible function results, depending on the logical value of the evaluated expression. Although I love this command, I have occasionally seen behavior that left me uneasy as to its reliability as of 1.0.3.1.

Syntax:

IIf (Expression, ExpressionTrue, ExpressionFalse)

Return value:

ExpressionTrue or ExpressionFalse

Parameter:

Expression: Conditional expression to be evaluated.

ExpressionTrue: Returned if the Expression above is true.

ExpressionFalse : Returned if the Expression above is false.

Example:

```
Sub IIfExample
  Print IIf(3>4,"Yes", "No") REM No
  Print IIf(4>2,"Yes", "No") REM Yes
End Sub
```

23.92. Imp Operator

Summary:

Perform a logical implication on two expressions. In the study of logic, it is said that x implies y if y is true whenever x is true. If x is false, however, the value of y is irrelevant and the expression is considered true. Consider the statement: If you are large (x), you will not be mugged (y). If you are large and you are mugged ($x=\text{True}$, $y=\text{False}$) then this statement is false. If you are large and not mugged ($x=\text{True}$, $y=\text{True}$) then this statement is true. If you are not large ($x=\text{False}$) then nothing can invalidate this statement.

x	y	x IMP y
TOSI	TOSI	TOSI
TOSI	EPÄTOSI	EPÄTOSI
EPÄTOSI	TOSI	TOSI
EPÄTOSI	EPÄTOSI	TOSI
1	1	1
1	0	0
0	1	1
0	0	1

Tip

(Not x Or y) is logically equivalent to (x Imp y). This is not as important as it used to be since the Imp function appears to work in 2.4.

Syntax:

Result = Expression1 Imp Expression2

Parameter:

Expression1, Expression2 : Numeric or boolean expressions.

Example:

```
Sub ExampleImp
    Dim bFalse As Boolean, bTrue As Boolean
    Dim i1%, i2%
    bFalse = False : bTrue = True
    Print bTrue Imp bTrue REM -1
    Print bTrue Imp bFalse REM 0
    Print bFalse Imp bTrue REM 0
    Print bFalse Imp bFalse REM -1
    i1 = 1 : i2 = 0
    Print i1 Imp i1 REM -1
    Print i1 Imp i2 REM -2
    Print i2 Imp i1 REM -2
    Print i2 Imp i2 REM -1
```

```
Print 1 Imp 0      REM -1
End Sub
```

23.93. Input Statement

Summary:

The Input statement is used to sequentially read numeric or string records from an open file and assign the data to one or more variables. The carriage return (Asc=13), line feed (Asc=10), and comma act as delimiters. When a numeric is read, a space is also used as a delimiter. Reading a non-numeric value into a numeric variable sets the value of the variable to zero.

It is not possible to read commas or quotation marks (") using the Input# statement. If you want to do this, then use the Line Input statement.

See Also: Open, Line Input#, Close, Eof, Get

Syntax:

Input #FileName var1[, var2[, var3[,...]]]

Parameter:

FileName: Number of the file from which data is to be read.

var: Numeric or String variables into which the read data will be placed.

23.94. InputBox Function

Summary:

Prompt the user for input in a dialog box. On cancel, a zero-length string is returned. If no position is provided, then the dialog is centered on the screen.

Syntax:

InputBox (Msg [, Title[, Default[, x_pos, y_pos As Integer]]])

Return value:

String

Parameter:

Msg: Message string displayed in the dialog box.

Title: Shown in the title bar of the dialog box.

Default: Default input string displayed in the text box.

x_pos: Absolute horizontal position in twips. This is an integer.

y_pos: Absolute vertical position in twips. This is an integer.

Example:

```
Sub ExampleInputBox
```



```

Dim s$
s = InputBox ("Prompt:", "Title", "default")
MsgBox ( s , 64, "Confirmation of phrase")
End Sub

```

23.95. InStr Function

Summary:

Returns the position of a string within another string.

The Instr function returns the position at which the match was found. If the string was not found, the function returns 0.

Warning As of 1.1.2, the return type is an integer but the return value may be as large as a string which is 64K. A negative number is returned if the value is too large.

```

Sub BugInStr
    Dim b$, i&
    b$ = String(40000, "a") & "|" REM character 40,001
    is an "|"
    i = instr(b, "|") REM -25535
    MsgBox cstr(i) & " or " & (65536 + i) REM -25535 or 40001
End Sub

```

Syntax:

InStr([Start As Integer,] Text1 As String, Text2 As String[, Compare])

Return value:

Integer

Parameter:

Start: Optional start position within the string. Defaults to 1, the first character.

Text1: String expression in which to search.

Text2: String expression for which to search.

Compare: If 1, then a case-insensitive compare, the default value 0 is a binary comparison.

Example:

```

Sub ExampleInStr
    Dim s$
    s = "SbxInteger getTruck(SbxLong) "
    RemoveFromString(s, "Sbx")
    Print s
End Sub

REM This deletes all occurrences of bad$ from s$
REM This modifies the string s$

```

```

Sub RemoveFromString(s$, bad$)
    Dim i%
    i = Instr(s, bad)
    Do While i > 0
        Mid(s, i, Len(bad), "")
        i = Instr(i, s, bad)
    Loop
End Sub

```

Warning You can not use the “Compare” parameter unless you also use the “Start” parameter.

23.96. Int Function

Summary:

Returns the largest integer that is not greater than the parameter. This means that it rounds the number toward negative infinity. Negative numbers, therefore, become greater in magnitude and positive numbers become smaller in magnitude.

See Also: CInt, Fix

Syntax:

Int (Number)

Return value:

Double

Parameter:

Number: Any valid numeric expression.

Example:

```

Sub ExampleINT
    Print " " & Int(3.14159) & " " & Fix(3.14)      REM 3 3
    Print " " & Int(0) & " " & Fix(0)              REM 0 0
    Print " " & Int(-3.14159) & " " & Fix(-3.1415) REM -4 -3
    Print " " & Int(2.8) & " " & Fix(2.8)          REM 2 2
End Sub

```

Warning -3.4 rounds to -4. Use Fix if you want to drop the fractional portion.

23.97. IsArray Function

Summary:

Tests if a variable is an array.

Syntax:

IsArray(Var)

Return value:

boolean

Parameter:

Var: Any variable to be tested whether it was declared as an array.

Example:

```
Sub ExampleIsArray
    Dim sDatf(10) as String, i
    Print IsArray(sDatf()) 'True
    Print IsArray(i())    'False
End Sub
```

23.98. IsDate Function

Summary:

Tests whether a numeric or string can be converted to a Date.

Syntax:

IsDate(Expression)

Return value:

boolean

Parameter:

Expression: Any numeric or string expression to be tested.

Example:

```
Sub ExampleIsDate
    Print IsDate("12.12.1997") 'True
    Print IsDate("12121997")   'False
End Sub
```

23.99. IsEmpty Function

Summary:

Tests if a Variant variable contains the Empty value, indicating that the variable has not been initialized.

See also: “Object, Variant, Empty, and Null” on page 392.

Syntax:

IsEmpty(Var)

Return value:

boolean

Parameter:

Var: Any variable to be tested.

Example:

```
Sub ExampleIsEmpty
    Dim v1 As Variant, v2 As Variant, v3 As Variant
    v2 = Null : v3 = "hello"
    Print IsEmpty(v1) ' True
    Print IsEmpty(v2) ' False
    Print IsEmpty(v3) ' False
    v2 = Empty ' ?? Broken as of version 1.0.3.1 and 1.1.1.
    Print IsEmpty(v2) ' Should say true
End Sub
```

23.100. IsMissing Function

Summary:

Tests if a subroutine or function was called with, or without, an optional parameter. The parameter must be declared with the “Optional” keyword for this to work. As of version 1.0.3.1, there were some minor bugs as mentioned in the section on optional parameters on page 398.

Syntax:

IsMissing(var)

Return value:

boolean

Parameter:

var: Variable to check

Example:

```
Function FindCreateNumberFormatStyle (sFormat As String, Optional doc, Optional
locale)
    Dim oDoc As Object
    Dim aLocale As New com.sun.star.lang.Locale
    Dim oFormats As Object
    'If it was not sent, then use ThisComponent
    oDoc = IIf(IsMissing(doc), ThisComponent, doc)
    oFormats = oDoc.getNumberFormats()
    ....
End Function
```

23.101. IsNull Function

Summary:

Tests whether a Variant or Object contains the special Null value, indicating that the variable contains no data. An uninitialized Object is NULL, an uninitialized Variant is Empty but it may be set to contain the NULL value.

See also: IsEmpty, include Macro GetSomeObjInfo

Syntax:

IsNull(Var)

Return value:

boolean

Parameter:

var: Variable to check

Example:

```
Sub ExampleIsNull
    Dim v1 as Variant, v2 As Variant, v3 As Variant, o As Object
    v2 = Null : v3 = "hello"
    Print IsNull(v1) ' False
    Print IsNull(v2) ' True
    Print IsNull(v3) ' False
    v3 = Null
    Print IsNull(v3) ' True
    Print IsNull(o)  ' True
End Sub
```

23.102. IsNumeric Function

Summary:

Tests if the given expression is a number or may be converted into one.

Syntax:

IsNumeric(Var)

Return value:

boolean

Parameter:

Var: Any expression to be tested. This is converted to a string and the string value is tested. If Var cannot be converted to a string, there may be an error.

Example:

```
Sub ExampleIsNumeric
    Dim v1, v2, v3
    v1 = "abc" : v2 = "123" : v3 = 4
    Print IsNumeric(v1) ' False
    Print IsNumeric(v2) ' True
End Sub
```

```
Print IsNumeric(v3)      ' True
Print IsNumeric("123x") ' False
End Sub
```

23.103. *IsObject Function*

Summary:

According to the on-line documentation, this tests whether the given object variable is an OLE object. I looked at the source code and ran a test, and this also returns true for a regular object. IsObject also returns true if the argument is a variant that contains an object.

See also: include Macro GetSomeObjInfo

Syntax:

IsObject(ObjectVar)

Return value:

boolean

Parameter:

ObjectVar: Any variable to be tested.

Example:

```
Sub ExampleIsObject
    Dim o As Object, s As String
    Print IsObject(o)      ' True
    Print IsObject(s)      ' False
End Sub
```

23.104. *IsUnoStruct Function*

Summary:

Returns true, if the given object is a UNO struct. The on-line help incorrectly states that the parameter is a name rather than an object.

See also: include Macro GetSomeObjInfo

Syntax:

IsUnoStruct(var)

Return value:

boolean

Parameter:

var: object to test

Example:

```
Sub ExampleIsUnoStruct
    Dim o As Object, s As String
```

```

Dim aProperty As New com.sun.star.beans.Property
Print IsUnoStruct(o) ' False
Print IsUnoStruct("com.sun.star.beans.Property") ' False
Print IsUnoStruct(aProperty) ' True
End Sub

```

23.105. Kill Function

Summary:

Deletes a file from disk. Any file notation may be used but wild cards are not supported.

Syntax:

Kill(file_name)

Return value:

None

Parameter:

file_name : Name of the file to kill

Example:

```

Sub ExampleKill
Kill "C:\datafile.dat"
End Sub

```

23.106. LBound Function

Summary:

Returns the lower bound of an array. An array index does not have to start at 0.

Syntax:

LBound(ArrayName [, Dimension])

Return value:

Integer

Parameter:

ArrayName: Name of the array for which to return the lower limit of the array dimension.

[Dimension] : Integer that specifies which dimension is desired. If no value is specified, the first dimension is assumed.

Example:

```

Sub ExampleUboundLbound
Dim a1(10 to 20) As String, a2 (10 to 20,5 To 70) As String
Print "(" & LBound(a1()) & ", " & UBound(a1()) & ")" ' (10, 20)
Print "(" & LBound(a2()) & ", " & UBound(a2()) & ")" ' (10, 20)
Print "(" & LBound(a2(),1) & ", " & UBound(a2(),1) & ")" ' (10, 20)

```

```
Print "(" & LBound(a2(),2) & ", " & UBound(a2(),2) & ")" ' (5, 70)
End Sub
```

23.107. LCase Function

Summary:

Return a lower case copy of the string. This does not modify the string.

Syntax:

LCase (String)

Return value:

String

Parameter:

String: string to be returned as lower case.

Example:

```
Sub ExampleLCase
    Dim s$
    s = "Las Vegas"
    Print LCase(s) REM Returns "las vegas"
    Print UCase(s) REM Returns "LAS VEGAS"
end Sub
```

23.108. Left Function

Summary:

Returns the leftmost n characters of a string.

Warning	As of 1.1RC2, the parameter to Left is an integer but the string may be 64K long.

Syntax:

Left(String, Integer)

Return value:

String

Parameter:

String: Any string expression

Integer: Number of characters to return. If 0, a zero-length string is returned.

Example:

```
Print Left("123456789", 2) 'Prints 12
```


23.109. Len Function

Summary:

Returns the number of characters in a string, or the number of bytes required to store a variable.

Syntax:

Len(Text As String)

Return value:

Long

Parameter:

Text: Any string expression or a variable of another type.

Example:

```
Sub ExampleLen
    Dim s$, i%
    s = "123456"
    i = 7
    Print Len(s)      '6
    Print Len(i)      '1
    Print Len(1134)   '4
    Print Len(1.0/3)  '17
End Sub
```

23.110. Let Function

Summary:

Optional keyword indicating that a value is to be assigned to a variable. This is rarely used.

Syntax:

[Let] VarName=Expression

Return value:

None

Parameter:

VarName: Variable to which a value will be assigned.

Example:

```
Sub ExampleLet
    Dim s$
    Let s = "Las Vegas"
End Sub
```

23.111. Line Input Statement

Summary:

Reads strings from a sequential file to a variable. First, you must open the file with the Open statement. String variables are read line-by-line up to the first carriage return (Asc=13) or linefeed (Asc=10). Line end marks are not included in the resulting string.

Syntax:

Line Input #FileName As Integer, Var As String

Return value:

None

Parameter:

FileName: Number of the open file from which data is to be read.

var: The name of the variable used to store the result.

Example:

23.112. Loc Function

Summary:

The Loc function returns the current position in an open file. If the Loc function is used for an open random access file, it returns the number of the last read or written record. For a sequential file, the Loc function returns the position in a file divided by 128. For binary files, the position of the last read or written byte is returned. ?? Verify this!

Syntax:

Loc(FileName)

Return value:

Long

Parameter:

FileName: Numeric expression containing the file number of an open file.

23.113. Lof Function

Summary:

Lof returns the size of an open file in bytes. To obtain the length of a file that is not open, use the FileLen function.

Syntax:

Lof(FileName)

Return value:

Long

Parameter:

FileNumber: Numeric expression containing the file number of an open file.

Example:

?? Verify this

```
Sub ExampleRandomAccess
    Dim iNumber As Integer
    Dim sText As Variant REM must be a Variant
    Dim aFile As String
    aFile = "c:\data.txt"
    iNumber = Freefile
    Open aFile For Random As #iNumber Len=32
    Seek #iNumber,1 REM Position at start
    Put #iNumber,, "This is the first line of text" REM Fill with text
    Put #iNumber,, "This is the second line of text"
    Put #iNumber,, "This is the third line of text"
    Seek #iNumber,2
    Get #iNumber,,sText
    Print sText
    Close #iNumber
    iNumber = Freefile
    Open aFile For Random As #iNumber Len=32
    Get #iNumber,2,sText
    Put #iNumber,,"This is a new line of text"
    Get #iNumber,1,sText
    Get #iNumber,2,sText
    Put #iNumber,20,"This is the text in record 20"
    Print Lof(#iNumber)
    Close #iNumber
End Sub
```

23.114. Log Function

Summary:

Returns the natural logarithm of a number. The natural logarithm is the logarithm to the base e. Base e is a constant with the approximate value 2.718282... You can calculate logarithms to any base (n) for any number (x) by dividing the natural logarithm of x by the natural logarithm of n, as follows: $\text{Log}_n(x) = \text{Log}(x) / \text{Log}(n)$

Syntax:

Log(Number)

Return value:

Double

Parameter:

Number: Numeric expression for which to calculate the natural logarithm.

Example:

```
Sub ExampleLogExp
    Dim a As Double
    Dim const b1=12.345e12
    Dim const b2=1.345e34
    a=Exp( Log(b1)+Log(b2) )
    MsgBox "" & a & chr(13) & (b1*b2) ,0,"Multiplication by logarithm function"
End Sub
```

23.115. Loop Statement

Summary:

The Loop statement is used to repeat statements while a condition is true, or until a condition becomes true. See the treatment on do loops on page 403.

Syntax:

Do [{While | Until} condition = True]

statement block

[Exit Do]

statement block

Loop

Syntax:

Do

statement block

[Exit Do]

statement block

Loop [{While | Until} condition = True]

Example:

```
Sub ExampleDoLoop
    Dim sFile As String, sPath As String
    sPath = "c:\\" : sFile = Dir$( sPath ,22)
    If sFile <> "" Then
        Do
            MsgBox sFile
            sFile = Dir$
        Loop Until sFile = ""
    End If
End Sub
```

```
End If
End Sub
```

23.116. LSet Statement

Summary:

LSet allows you to left justify a string within the space taken used by another. Any leftover positions are filled with spaces. If any text in the new string can not fit into the old string, it is truncated. This is broken in versions prior to 1.1.x.

LSet also allows you to overlay data from one user-defined type with data from another. This takes all the bytes from one data structure and overlays them on top of another, ignoring the underlying structure. I have not tried this with a user defined type in version 1.1.1.

Syntax:

LSet Var As String = Text

LSet Var1 = Var2

Parameter:

Var: Any String variable, in which the string to be aligned to the left.

Text: String to be aligned to the left of the string variable.

Var1: Name of the user-defined type variable being copied to.

Var2: Name of the user-defined type variable being copied from.

Example:

```
Sub ExampleLSet
    Dim sVar As String, sExpr As String
    sVar = String(40, "*")
    sExpr = "SBX"
    REM Left-align "SBX" within the 40-character reference string
    LSet sVar = sExpr
    Print ">"; sVar; "<"      REM ">SBX"                                <"
    sVar = String(5, "*")
    sExpr = "123456789"
    LSet sVar = sExpr
    Print ">"; sVar; "<"      REM ">12345<"
End Sub
```

23.117. LTrim Function

Summary:

Removes all leading spaces of a string expression.

Syntax:

LTrim(Text)

Return value:

String

Parameter:

Text: Any string expression.

Example:

```
Sub ExampleSpaces
    Dim sText2 As String, sText As String, sOut As String
    sText2 = " <*Las Vegas*> "
    sOut = "'" + sText2 + "'" + Chr(13)
    sText = Ltrim(sText2)      REM sText = <*Las Vegas*> "
    sOut = sOut + "'" + sText + "'" + Chr(13)
    sText = Rtrim(sText2)     REM sText = " <*Las Vegas*> "
    sOut = sOut + "'" + sText + "'" + Chr(13)
    sText = Trim(sText2)      REM sText = " <*Las Vegas*> "
    sOut = sOut + "'" + sText + "'"
    MsgBox sOut
End Sub
```

23.118. Private Keyword

Summary:

The Private keyword is used to declare a variable outside of a subroutine as private. If a variable is declared using the Dim keyword, it is considered private. See the description on Dim for syntax descriptions.

See also: Dim, Public

Syntax:

Private Name_1 [(start To end)] [As VarType][, Name_2 [(start To end)] [As VarType][,...]]

Example:

```
Private iPriv As Integer
Sub ExamplePublic
    iPriv = 1
    Call CalledSub
End Sub
Sub CalledSub
    Print iPriv REM 1
End Sub
```

23.119. Public Keyword

Summary:

The Public keyword is used to declare a variable outside of a subroutine as Public to all modules. If a variable is declared using the Dim keyword, it is considered private. See the description on Dim for syntax descriptions.

See also: Dim, Private

Syntax:

Public Name_1 [(start To end)] [As VarType][, Name_2 [(start To end)] [As VarType][,...]]

Example:

```
Public iPub As Integer
Sub ExamplePublic
    iPub = 1
    Call CalledSub
End Sub
Sub CalledSub
    Print iPub REM 1
End Sub
```

23.120. Red Function

Summary:

Colors are represented by a long integer. Return the red component of the specified color code. See also RGB, Blue, and Green.

Syntax:

Red(Color As Long)

Return value:

Integer in the range of 0 to 255.

Parameter:

Color value: Long integer expression representing a color.

Example:

```
Sub ExampleColor
    Dim lColor As Long
    lColor = RGB(255,10,128)
    MsgBox "The color " & lColor & " consists of:" & Chr(13) & _
        "Red = " & Red(lColor) & Chr(13) & _
        "Green= " & Green(lColor) & Chr(13) & _
        "Blue= " & Blue(lColor) & Chr(13) , 64,"Colors"
End Sub
```

23.121. RSet Statement

Summary:

RSet allows you to right justify a string within the space taken used by another. Any leftover positions are filled with spaces. If any text in the new string can not fit into the old string, it is truncated. Although RSet was broken in OOo version 1.0.3.1, it works in OOo version 1.1.1.

Syntax:

RSet Var As String = Text

Parameter:

Var: Any String variable, in which the string to be aligned to the left.

Text: String to be aligned to the left of the string variable.

Example:

```
Sub ExampleRSet
    Dim sVar As String, sExpr As String
    sVar = String(40, "*")
    sExpr = "SBX"
    RSet sVar = sExpr
    Print ">"; sVar; "<"      REM ">          SBX<"
    sVar = String(5, "*")
    sExpr = "123457896"
    RSet sVar = sExpr
    Print ">"; sVar; "<"      REM ">12345<"
End Sub
```

23.122. Shell Function

Summary:

Start an external application. The window style of the started application may be optionally included with the following values:

Style	Description
0	Focus is on the hidden program window.
1	Focus is on the program window in standard size.
2	Focus is on the minimized program window.
3	Focus is on the maximized program window.
4	Standard size program window, without focus.
6	Minimized program window, but focus remains on the active window.
10	Full-Screen display.

The program is assumed to start and continue running in the background unless the last parameter (bsync) is set to True. This means that control is returned immediately from the Shell command.

The return type is not specified in the on-line help. Experimentally, I have determined this type to be a LONG. The return value has always been zero when I have bothered to check it. If the program does not exist, then an error is generated and the macro halts.

Syntax:

Shell (Pathname As String[, Windowstyle As Integer][, Param As String][, bSync])

Return value:

Long

Parameter:

Pathname: Complete path and program name of the program to start.

Windowstyle: Specifies the style of the window in which the program is executed.

Param: Any string expression that specifies the command line to be passed.

bSync: If False (the default), an immediate return occurs. If True, then the Shell statement does not return until after the program is finished running.

Example:

```
Sub ExampleShell
    Dim vRC As Variant
    REM A window type of 2 displays the window normally on top
    REM
    vRC = Shell("C:\andy\TSEProWin\g32.exe", 2, "c:\Macro.txt")
    Print "I just returned and the returnec code is " & vRC
    REM These two have spaces in their names
    Shell("file:///C:/Andy/My%20Documents/oo/tmp/h.bat",2)
    Shell("C:\Andy\My%20Documents\oo\tmp\h.bat",2)
End Sub
```

Antal Attila <atech@nolimits.ro> provided the following example of the use of the bsync argument.

```
Sub Main()
    ' First, create on your disk a file with the following content:
    ' on Windows (with name C:\tmp\Test.bat):
    '     echo %1
    '     pause
    ' on Linux (with name /home/guest/Test.sh):
    '     echo $1
    '     sleep 100000

    ' ----- Sync example -----
    ' calling my shell runner method with bSync=TRUE
    ' the basic execution will hanging up while the terminal
    ' ( or msdos prompt) will be closed (any key or Ctrl+C)
    ' on Windows
```

```

shellRunner("file://C:/tmp/", "Test", "Helo World", TRUE)
' or on Linux
shellRunner("file:///home/guest/", "Test", "Helo World", TRUE)
' signaling the end of execution
Print "The End"

' ----- Async example -----
' calling my shell runner method with bSync=FALSE
' the basic execution will be continued
' on Windows
shellRunner("file://C:/tmp/", "Test", "Helo World", FALSE)
' or on Linux
shellRunner("file:///home/guest/", "Test", "Helo World", FALSE)
' signaling the end of execution
Print "The End"
End Sub

Sub shellRunner(dirPath$, script$, prms$, sync as Boolean)
    Dim filePath$, ef$, ed$, isWindows as Boolean

    ' looking for OS type
    If instr(mid(dirPath,8),":/")>0 or instr(dirPath,8),"\")>0 Then
        isWindows=TRUE
    Else
        isWindows=FALSE
    End If

    ' converting the url to file path
    filePath = convertFromURL(dirPath)

    ' creating the execution string
    If isWindows Then
        ef = "command.com /C "+filePath+script+".bat"
    Else
        ef = "xvt -e sh "+filePath+script+".sh"
    End If

    ' running the shell command
    Shell(ef, 1, prms, sync)
End Sub

```

23.123. UBound Function

Summary:

Returns the upper bound of an array.

Syntax:

UBound(ArrayName [, Dimension])

Return value:

Integer

Parameter:

ArrayName: Name of the array for which to return the lower upper of the array dimension.

[Dimension] : Integer that specifies which dimension is desired. If no value is specified, the first dimension is assumed.

Example:

```
Sub ExampleUBoundLbound
    Dim a1(10 to 20) As String, a2 (10 to 20,5 To 70) As String
    Print "(" & LBound(a1()) & ", " & UBound(a1()) & ")" ' (10, 20)
    Print "(" & LBound(a2()) & ", " & UBound(a2()) & ")" ' (10, 20)
    Print "(" & LBound(a2(),1) & ", " & UBound(a2(),1) & ")" ' (10, 20)
    Print "(" & LBound(a2(),2) & ", " & UBound(a2(),2) & ")" ' (5, 70)
End Sub
```

23.124. UCase Function

Summary:

Return an upper case copy of the string. This does not modify the string.

Syntax:

UCase (String)

Return value:

String

Parameter:

String: string to be returned as upper case.

Example:

```
Sub ExampleUCase
    Dim s$
    s = "Las Vegas"
    Print LCase(s) REM Returns "las vegas"
    Print UCase(s) REM Returns "LAS VEGAS"
End Sub
```

23.125. URL Notation And Filenames

23.125.1. URL Notation

On a windows computer, “c:\autoexec.bat” is a typical method to reference a file. This may also be referenced in URL notation as “file:///c:/autoexec.bat”. A general idea when

performing such conversions is to start the URL with “file:///”, and replace “\” with “/”. If you want to insert the computer name or IP address, try this: third “/” characters as in “file://localhost/c:/autoexec.bat/”; if this fails, try replacing “:” with “|”.

23.125.2. Paths With Spaces And Other Special Characters

Spaces and special characters can be embedded into URLs are standard file notation using the standard URL escape sequence. Take the ASCII value you intend to embed, convert it to hex, precede it with a “%”, and then place it where you want the character. Consider embedding a space in a path. “c:\My%20Documents\info.sxw” and “file:///c:/My%20Documents/info.sxw”.

23.125.3. Anchoring To The Home Directory On Unix

Thanks to Andrew McMillan (andrew@catalyst.net.nz), who mentions that on a Unix/Linux system a person may anchor a pathname to a home directory as follows:

```
file://~/document.sxw - document.sxw in my home directory.  
file://~person/document.sxw - document.sxw in "person" home directory.
```

24. Other languages

24.1. C#

In C#, you must use COM objects to access OOo objects. Everything is returned as an object, including integers and such. The following simple C# example should demonstrate how to call a variety of different methods with different return values. This is not meant to be a complete example. If you have better and different examples, let me know.

Listing 24.1: Simple C# example to access bookmarks.

```
using System;
using System.Reflection;

namespace ootest
{
    class Class1
    {
        // obj - Invoke a method on this object.
        // method - name of the method to invoke.
        // par - Array of object arguments
        static object Invoke(object obj, string method, params object[] par)
        {
            return
obj.GetType().InvokeMember(method, BindingFlags.InvokeMethod, null, obj, par);
        }

        static void PrintImpName(object obj)
        {
            object x = Invoke(obj, "getImplementationName", new object[0]);
            System.Console.WriteLine(x.ToString());
        }

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        public static void Main(string[] args)
        {
            // Get a copy of the service manager.
            object usm = Activator.CreateInstance(
                Type.GetTypeFromProgID("com.sun.star.ServiceManager"));

            // Create a copy of the desktop. Remember that there is ONLY one.
            object desk=Invoke(usm, "createInstance", "com.sun.star.frame.Desktop");
            PrintImpName(desk);

            // The typical example loads a new Calc document
            //object calcDoc = Invoke(desk, "loadComponentFromURL",
            //    "private:factory/scalc", "_blank", 0, new object[0]);
        }
    }
}
```

```

// How about getting the current component? this takes no arguments

object oDoc = Invoke(desk, "getCurrentComponent", new object[0]);
PrintImpName(oDoc);
//System.Threading.Thread.Sleep(1000);

object x = Invoke(oDoc, "supportsService",
                  "com.sun.star.text.TextDocument");
if (x is bool && (bool) x)
{
    System.Console.WriteLine("The current component is a text document");
}
else
{
    System.Console.WriteLine(
        "The current component is NOT a text document");
    System.Threading.Thread.Sleep(10000);
    return;
}
object oBookmarks = Invoke(oDoc, "getBookmarks", new object[0]);
x = Invoke(oBookmarks, "getCount", new object[0]);
int nCount = (int) x;
for (int n = 0; n < nCount; ++n)
{
    object oMark = Invoke(oBookmarks, "getByIndex", n);
    x = Invoke(oMark, "getName", new object[0]);
    System.Console.WriteLine((string) x);
}

System.Threading.Thread.Sleep(5000);
}
}
}

```

24.2. Visual Basic Programmers

My book contains numerous notes on differences between Visual Basic and StarBasic; I will not repeat that information here – if you want to see that, buy the book.

The enterprise version of StarOffice (commercial version of OOo, see <http://www.staroffice.com>) is able to run office macros in StarOffice.

Novell is developing a free method of running Excel macros natively in OOo. Some of the functionality is already available in "OpenOffice Novell Edition", which is part of SUSE Linux.

<https://reverendted.wordpress.com/2006/07/03/openofficeorg-and-excel-vba-macros/>

There are many references available from other sources.

<http://www.oooforum.org/forum/viewforum.phtml?f=9>

<http://www.oooforum.org/forum/viewtopic.phtml?t=8833&sid=0d667c7f6452b204aef49b352cee2007>

24.2.1. ActiveWorkbook

To obtain the “active workbook” from OOo Basic, use the variable `ThisComponent`. If you are not using OOo Basic, then you must obtain the current component from the desktop object. Unfortunately, the Basic IDE and the OOo help window is also a component of the desktop, so you need to verify that the component supports the `XModel` interface. Some of the examples in this document use code such as `StarDesktop.getCurrentComponent()` or `StarDesktop.CurrentComponent` to obtain the current document. This type of code usually fails when it is run from the IDE because the IDE is the current component, but the last document that was current is referenced by the variable `ThisComponent`.

OpenOffice.org documents support a model, that contains the data, a view to display the data, and a controller that interacts with the user. It is the controller that knows what the user is doing so in general, if you want to know the current state of things, you should obtain the document's current controller and ask it. It is the current controller, for example, that knows about the currently selected text, the active sheet and the active cell.

24.2.2. ActiveSheet and ActiveCell

To find the active sheet, you need to call `getActiveSheet()` on an object that supports the `SpreadSheetView` service. Well, really, it must support the `XSpreadsheetView` interface, but the `SpreadsheetView` service supports the interface:

<http://api.openoffice.org/docs/common/ref/com/sun/star/sheet/SpreadsheetView.html>

Obtain the obtain that spreadsheet view by calling `ThisComponent.getCurrentController()`. The current controller is what interacts with the user and so it is what knows what is currently selected. The current controller for a Writer document will not support the `SpreadSheetView` service.

The problem with the current cell is that you may not have a current cell, at least not one that is obviously available. Typically, you obtain the current selection, which may be a cell, a range of cells, or multiple disjoint selections. Section 6.5 deals with selected text in a Calc document.

If you have multiple things selected and you need to know which cell contains the cursor, then you have to perform a little trickery because this is not directly available. It is possible to simply move the cell one position left and then back again, but this can fail and you lose the current selection. A more elegant solution that works was created by Paolo Mantovani and demonstrated in the `RetrieveTheActiveCell` method that is shown in the section `Get the active cell and ignore the rest` on page 146.

25. Index

Abs 434
 And 397, 419, 433
 ApplicationScriptLibraryContainer 101p.
 Array 394, 396
 Asc 435, 443
 ATN 436
 AVERAGE 151
 Beep 436
 Blue 437
 Boolean 387
 BottomLine 154
 ByVal 399p., 437
 Call 438
 case 407
 CBool 439
 Cbyte 439
 CDate 440
 CDateFromIso 440
 CDateToIso 441
 CDbI 441
 Cell
 CellAddress 148
 CellAddressConversion 146, 148
 CellBackColor 142
 getFormula 142
 getSpreadSheet 148
 getString 141
 getValue 141
 IsCellBackgroundTransparent 142
 NumberFormat 142, 151p.
 setFormula 142, 151p.
 setString 142, 151
 setValue 142
 CellAddress
 Column 148
 Row 148
 CellRangeAddressConversion 148
 CharacterProperties
 CharFontName 215
 CharHeight 215, 244
 CharLocale 80, 216
 CharPosture 215
 FontSlant 215
 CharUnderline 215p.

CharWeight 215, 222, 226
 FontWeight 215, 222, 226
 FontSlant
 DONTKNOW 215
 ITALIC 215
 NONE 215
 OBLIQUE 215
 REVERSE_ITALIC 215
 REVERSE_OBLIQUE 215
 FontUnderline
 BOLD 215
 BOLDDASH 216
 BOLDDASHDOT 216
 BOLDDASHDOTDOT 216
 BOLDDOTTED 216
 BOLDLONGDASH 216
 BOLDWAVE 216
 DASH 215
 DASHDOT 215
 DASHDOTDOT 215
 DONTKNOW 215
 DOTTED 215
 DOUBLE 215
 DOUBLEWAVE 215
 LONGDASH 215
 NONE 215
 SINGLE 215
 SMALLWAVE 215
 WAVE 215
 FontWeight
 BLACK 215
 BOLD 215
 DONTKNOW 215
 LIGHT 215
 NORMAL 215
 SEMIBOLD 215
 SEMILIGHT 215
 THIN 215
 ULTRABOLD 215
 ULTRALIGHT 215
 ChDir 442
 ChDrive 442
 Choose 443
 Chr 435, 443
 Christian Anderson 399
 CInt 444

clipboard	74		
CLong	444		
Close	36p., 43, 56, 445, 464, 478		
COM	1		
Component			
CurrentController	152		
CurrentSelection	152		
DatabaseRanges	152		
getCurrentController	155		
getText	9		
removeByName	152		
computed goto	407		
ConfigurationAccess	35		
ConfigurationProvider	32, 35		
ConfigurationUpdateAccess		32	
Const	446		
Control			
button	339		
check box	340		
combo box	340		
Controls	338		
group box	340		
label	339		
list box	339		
progress bar	340		
radio button	340		
text box	339		
Controller			
ActiveSheet	152		
getViewCursor()	215p.		
select	155		
StatusIndicator	29		
ConvertFromURL	446		
ConvertToURL	447		
Cos	447		
createInstanceWithArguments		35	
createTextCursorByRange		234	
CreateUnoDialog	337, 448		
CreateUnoService	10, 449		
CreateUnoStruct	450		
CSng	450		
CStr	451		
CurDir	452		
Currency	387, 392		
CurrentController	29		
Cursor			
			compareRegionEnds 185, 188p., 191pp., 196pp., 201, 205, 207, 224, 229
			compareRegionStarts 185, 199, 201, 209, 221, 229
			CreateTextCursor 43, 66, 68, 80, 158, 186p., 189, 208, 213, 219p., 231p., 283
			createTextCursorByRange 39, 58, 183pp., 187p., 201, 203, 208pp., 212, 215pp., 221, 372
			getStart() 215p.
			goLeft 179, 185, 192p., 205pp., 209
			goRight 80, 179, 185, 188p., 191pp., 196pp., 204pp., 209, 211, 213, 224p., 229p.
			gotoEnd 43, 66, 179, 186p., 208, 220
			gotoEndOfParagraph 215, 219, 231p., 234
			gotoEndOfParagraph() 215
			gotoNextParagraph 80, 189, 193, 205, 220p., 231p., 234
			gotoRange 189, 225, 245
			GoToStart 80, 179, 186p., 189, 208, 213, 219p., 231p.
			gotoStartOfParagraph() 215
			IsCollapsed 179, 183p., 187p., 208p.
			Date 387, 397, 452
			DateSerial 441, 453
			DateValue 440p., 454
			Day 454
			dbg_methods 16
			dbg_properties 16
			dbg_supportedInterfaces 16
			Debug
			PrintdbgInfo 16
			ShowArray 16
			WritdbgInfo 16
			Declare 455
			DefBool 387, 455
			DefDate 387, 456
			DefDbl 387, 457
			DefInt 387, 457
			DefLng 387, 457
			DefObj 387, 458
			DefVar 387, 458
			Desktop
			CurrentComponent 152
			LoadComponentFromURL() 37

Dialog			
endExecute	338		
execute	337		
Dim	387, 389, 394p., 459		
DimArray	395, 434, 460		
Dir	460, 481		
dispose	36, 43, 67p., 253, 344, 487		
Do...Loop	468		
Double	387, 392		
Else	401, 488		
ElseIf	401, 488		
email	97		
sending email		97	
Empty	392		
End	463		
End Function		463	
End If	463		
End Select	463		
End Sub	463		
Environ	463		
EOF	445, 464, 478		
EqualUnoObjects		234, 465	
EQV	419, 465p.		
Erl	409, 466		
Err	409, 467		
Error	409, 467		
ExampleShell		508	
Exit	398, 408		
Exit DO	408		
Exit For	408		
Exit Function		408	
Exit Sub	408		
Exit Do	404, 468		
Exit For	402, 468		
Exit Function		406, 469	
Exit Sub	406, 469		
Exp	469		
False	397		
Fibonacci	58		
FieldMaster	63pp., 262		
File			
CLOSE	56		
FileExists	56		
FreeFile	56		
LINE INPUT		56	
Loc	501		
Lof	502		
OPEN	56		
FileAttr	470		
FileCopy	471		
FileDateTime		472	
FileExists	472		
FileLen	473, 502		
Find & Replace			
createReplaceDescriptor	223, 226		
createSearchDescriptor	222, 224p.		
findFirst	222pp.		
findNext	222pp.		
replaceAll	223, 226		
ReplaceDescriptor			
ReplaceString		223, 226	
SearchDescriptor			
searchAll	226		
SearchCaseSensitive		222pp.	
SearchRegularExpression		226	
SearchString		222pp.	
searchStyles		226	
SearchWords		222	
SetReplaceAttributes	226		
SetSearchAttributes	226		
Fix	475		
For	402		
For...Next	468		
Format	477		
FreeFile	445, 464, 478		
Function	7, 398		
GCD	85		
Get	480		
GetAttr	460, 481		
getImplementationName	21		
getNumberFormats()	57		
GetProcessServiceManager()		482	
GetSolarVersion	34		
GetSolarVersion()	483		
GetSystemTicks()	483		
GlobalScope	484		
GoSub	401, 406, 484, 488		
GoTo	56, 401, 406p., 485, 488		
gotoEndOfParagraph	230		
gotoNextParagraph	230, 234		
GraphicExportFilter	60, 109		

GraphicObjectShape			
Enumeration			
GraphicObjectShape	237		
Green	486		
GUI	1		
hasLocation	37, 39, 59, 119		
HasUnoInterfaces	486		
Header			
HeaderOn	158		
HeaderShared	158		
Hex	487		
Hour	488		
IDE	1		
IDL	1		
If	401, 488		
IIf	397p., 401, 489		
Imp	419, 490		
Input	491		
Input#	491		
InputBox	491		
InStr	492		
Int	56, 493		
Integer	387, 391		
Is	397		
IsArray	14, 397, 493		
IsCursorInlastPar	234		
IsDate	397, 494		
IsEmpty	14, 392, 397, 494		
IsMissing	57, 397pp., 495		
isModified	37, 39, 59		
IsNull	14, 392, 397, 496		
IsNumeric	397, 496		
IsObject	14, 397, 497		
isPlugged	127		
isReadOnly	37, 39, 59		
IsUnoStruct	14, 397, 497		
Kill	445, 464, 478, 498		
LBound	394, 498		
LCase	499		
Left	499		
Len	29, 56, 500		
Let	500		
library	8		
Line Input	501		
LineDistance	154		
loadComponentFromURL	37		
Loc	501		
Locale	79		
Country	216		
Language	216		
Lof	502		
Log	503		
Long	387, 391		
Loop	403		
Do	403		
Do Until	403		
Do While	403		
Loop Until	403		
Loop While	403		
LSet	504		
LTrim	505		
Macro Author			
ADPSetWordCase	228		
Andrew Pitonyak			
AccessModes	471		
ADPWordCountCharCursor	196		
ADPWordCountStrings	194		
ADPWordCountWordCursor	197		
CalcGroupingExample	157		
ClearDefinedRange	142		
CloseOpenDocument	486		
ColumnNumberToString	148		
CreateSelectedTextIterator	187		
DecimalFeetToString	87		
DisplayAllStyles	29		
ExampleNewPage	218		
ExampleShell	55		
FileAttributeString	481		
FindCreateNumberFormatStyle	57		
ForNextExampleSort	402		
GetLeftMostCursor	185		
GetRightMostCursor	185		
GetSomeObjInfo	15		
InsertDateField	217		
InsertDateIntoCell	149		
InsertSimpleText	216		
IsAnythingSelected	183		
IsSpreadsheetDoc	141		
IsWhiteSpace	190, 479		

IterateOverSelectedTextFramework		UnzipAFile	110
186		Load library	
MultipleTextSelectionExample		Sunil Menon	102
184		Marc Messeant	
PrintableAddressOfCell	148	AppliquerStyle	221
PrintAscii	423	Niklas Nebel	
PrintDataInColumn	156	setting_borders_in_calc	154
PrintEachCharacterWorker		Oliver Brinzing	
188		CopySpreadsheetRange	76
ProtectSpreadsheet	157	Olivier Bietzer	
RankChar	190	GCD	85
Read_Write_Number_In_File	55	OpenOffice	
RemoveEmptyParsWorker		GetDocumentType	21
192		Paolo Mantovani	
RemoveFromString	422	RetrieveTheActiveCell	146
ReplaceInString	422	Paul Sobolik	31
SearchSelectedText	224	ListFonts	31
SelectedNewLinesToSpaces		Peter Biela	24
202		Ryan Nelson	
SelectedNewParagraphsToNewLines		CopyPasteRange	75
203		Sasa Kelecvic	
SetDocumentLocale	79	ActiveSheet	152
SetTextAttributes	215	Analize	151
testOptionalParameters	398	CellPos	152
ToFraction	86	DefineDbRange	153
Bernard Marcelly		DeleteDbRange	152p.
ErrorHandlingExample	411	ExampleGetValue	141
Birgit Kellner		ExampleSetValue	142
AtToUnicode	223	FillCells	151
Christian Junker	32, 74, 283	GetAddress	152
Database Columns		ProgressBar	29
David Kwok	163	SelectedCells	150
David Woody		SortRange	155
DrawLineInCalcDocument		StatusText	29
110		SetBitMapSize	
InsertAndPositionGraphic		Vance Lankhaar	102
104		Stephan Wunderlich	
edA-qa mort-ora-y		CopySpreadsheet	159
Fibonacci	58	MAX	151
GetGlobalRangeByName		MIN	151
Rob Gray	160	MOD	419
Hermann Kienlein	283	module	8
Laurent Godard		Modules	8
OOoLang	35	NOT	397
OOOVersion	34, 482	Null	392, 397
SendSimpleMail	97, 449	NumberFormat	57

addNew	57		Return	406
DATE	57		RSet	507
FindCreateNumberFormatStyle	149,		SDK	1
217			Select	74
queryKey	57		Case	404
Object	387p., 392		Case Else	404
OLE	1		Is	404
OleObjectFactory	55, 98		Select Case	404
On Error	56		To	404
Local	409		Selection	
On Error	409		Columns	152
On Error GoTo 0	409		getRangeAddress	152
On Error GoTo Label	409		Rows	152
On Error Resume Next	409		ServiceInfo	16
Resume	410		SetExpression	70
Resume Label:	410		setModified	37, 39
Resume Next	410		Shell	508
On Local Error Goto 0	56		SimpleCommandMail	97
On N GoSub	407		SimpleSystemMail	97
On N GoTo	407		Single	387
OOo	1		Sort	155
Open	56, 445, 464, 478		SortField	155
Option Base	394		Field	155
Option Explicit	387		SortAscending	155
Optional	57, 397pp.		SpreadsheetDocument	
OR	397p., 419		BottomLine	154
outline numbering	249		Column	
Outlook	98		Columns	150p.
Outlook.Application	98		getByIndex	150, 152
PageDescName	218		getCount	151p.
PageNumberOffset	218		getName	150, 156
ParaStyleName	231		Columns	150, 156
PI	393		Count	143
PRINT	56		CurrentSelection	150p.
printdbgInfo	3		getByName	141p., 155
Private	389, 505		getCellByPosition	141p., 151
PropertyValue	155		getCellRangeByName	155
Name	155		getCellRangeByPosition	154
Value	155		getCount	150
Public	389, 506		getName	148
Red	506		getRangeAddress	150
ReDim	387, 389, 395p., 459		Row	
Preserve	395p.		EndRow	150, 152
ReDim	395p.		getCount	150pp.
ReDimExample	395		Rows	151
REM	387		StartRow	150, 152

Rows	150		findFirst	222
setString	151		findNext	224
Sheets	141pp., 154pp.		getCurrentSelection	
SpreadsheetDocument	141		getCount()	184
SupportsService	141		getCurrentSelection()	183p.
TableBorder	154		getByIndex()	184
StarDesktop	9p., 17		getCount()	183
StarOffice	17		insertControlCharacter	204
Static	389		insertString	216
StatusIndicator			Text	215
start	29		TextField	
Str	56		Annotation	218
String	387p., 392		DateTime	217
StyleFamilies	158		DocInfo	
Styles			Revision	262
CharacterStyles	29		Subject	262
FrameStyles	29		FieldMaster	
NumberingStyles	29		attachTextFieldMaster	66
PageStyles	29		Drawing	262
ParagraphStyles	29		Illustration	262
Sub	7, 398		Table	262
switch	404, 407		Text	262
SystemShellExecute	118		User	64pp., 262
table of contents	253		SheetName	158
TableBorder	154		URL	69
Tan	436		TextField.SetExpression	70
terminate	36		TextGraphicObject	108
TextContent	237		Duplicate	
TextCursor	179		TextGraphicObject	50
goDown()	179		Embed	
goLeft()	179		TextGraphicObject	51p.
goRight()	179		Enumeration	
gotoEnd()	179		TextGraphicObject	237
gotoStart()	179		Export	
goUp()	179		TextGraphicObject	108
IsCollapsed	179		TextRange	179
IsCollapsed()	183p.		TextSection	18, 254
TextCursor	179		Thanks	
TextDocument			Alain Viret	22
compareRegionEnds()	185		Andreas Bregas	2, 393, 399
compareRegionStarts()	185		Andrew Brown	193, 198
createReplaceDescriptor	223		Andrew McMillan	511
createTextCursorByRange	216		Antal Attila	508
createTextCursorByRange()	183pp., 215		Antoine Jarrige	126
CurrentController	215p.		Berend Cornelius	73, 344
			Bernard Marcellly	156, 223, 405, 410

Birgit Kellner	223	ThisDatabaseDocument	10
Chris Clementson	164	TimeValue	440
Christian Erpelding	29	TOC	253
Christian Junker	iii, 36, 68	Trim	56
Dan Juliano	111	True	397
Daniel Juliano	98	TypeName	14, 389
Daniel Vogelheim	199	UBound	394p., 510
David Woody	154	UCase	510
Eric VanBuggenhaut	126	UNO	1
Erik Anderson	118	Until	503
Frank Schönheit	337	Val	434
Gerrit Jasper	165, 168	Variant	387p., 392, 397, 494
Giuseppe Castagno	242	VarType	389
Hal Vaughan	24	Visual Basic	1
Hermann Kienlein	2p.	While	503
Hermann-Josef Beckers	2p.	While...Wend	406
Jean Hollis Weber	2	WritedbgiInfo	3
Jim Thompson	383	XOR	397p., 419
Jürgen Schmidt	101	XSelectionSupplier	74
Kelvin Eldridge	2	-	419, 431
Laurent Godard	2, 103, 110, 113	*	419, 431
Leston Buell	371	/	419, 433
Marc Messeant	83, 221	&	419
Mathias Bauer	2, 24, 73, 384	^	419, 432
Michelle Pitonyak	2	+	419, 432
Mikhail Voitenko	41	<	397, 419
Niklas Nebel	154	<=	397, 419
Oliver Brinzing	158, 346	<>	397, 419
Oliver Brinzing	76	=	397, 419
Olivier Bietzer	85	>	397, 419
Paolo Mantovani	116, 120, 146, 341, 374, 515	>=	397, 419
Paul Sobolik	30		
Peter Biela	24		
Rob Gray	160		
Rodrigo V Nunes	65		
Russ Phillips	118		
Ryan Nelson	74, 157		
Sasa Kelecevic	2p., 29, 141p., 150p., 153, 155		
Solveig Haugland	2		
Stephan Wunderlich	159		
Sunil Menon	102		
Sven Jacobi	108		
Vance Lankhaar	102p.		
ThisComponent	9p., 17		