Useful Macro Information
For OpenOffice
By
Andrew Pitonyak

This is not the same as my book

OpenOffice.org Macros Explained.

My book is a more complete and organized presentation. This document, is primarily a random collection of thoughts and examples (brain dump).

Last Modified
Tuesday, November 6, 2007 at 03:54:17 PM
Document Revision: 1026
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. Robert Black Eagle you were tireless and offered excellent insight, not to mention your wonderful Master Document “Howto”. 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. Finally, I should thank Jean Hollis Weber and C Robert Pearsall for agreeing to be technical editors for my book due for publication in August of 2004. Thanks to Christian Junker, for reviewing and making extensive changes and additions to the document – Christian spent numerous hours proofreading and providing updates.

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

<table>
<thead>
<tr>
<th>Contribution</th>
<th>Contributor</th>
<th>Contact</th>
<th>Copyright</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original author</td>
<td>Andrew Pitonyak</td>
<td><a href="mailto:andrew@pitonyak.org">andrew@pitonyak.org</a></td>
<td>2002-2007</td>
</tr>
<tr>
<td>French translation</td>
<td>The French Native-lang documentation project</td>
<td><a href="mailto:doc@fr.openoffice.org">doc@fr.openoffice.org</a></td>
<td>2003</td>
</tr>
<tr>
<td>German Translation</td>
<td>Hermann-Josef Beckers</td>
<td><a href="mailto:hjb-rheine@t-online.de">hjb-rheine@t-online.de</a></td>
<td>2003</td>
</tr>
<tr>
<td>Macros</td>
<td>Hermann Kienlein</td>
<td><a href="mailto:info@kienlein.com">info@kienlein.com</a></td>
<td>2002-2003</td>
</tr>
<tr>
<td>Macros</td>
<td>Sasa Kelecevic</td>
<td><a href="mailto:scat@teol.net">scat@teol.net</a></td>
<td>2002-2003</td>
</tr>
<tr>
<td>Editing and additions</td>
<td>Christian Junker</td>
<td><a href="mailto:usboy2004@web.de">usboy2004@web.de</a></td>
<td>2004</td>
</tr>
</tbody>
</table>

Table of Contents

1. Introduction ........................................................................................................................................ 1
   1.1. 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 ....................................................................................................................................... 8
   3.4. Variables, Constants, Strings and numerical Types ......................................................................... 8
   3.5. Accessing And Creating Objects In OpenOffice ............................................................................. 9
   3.6. Everyone keeps talking about UNO, what is it? .............................................................................. 10
      3.6.1. Structures .................................................................................................................................. 10
      3.6.2. Interfaces .................................................................................................................................. 11
      3.6.3. Services .................................................................................................................................... 12
      3.6.4. Interfaces and services ........................................................................................................... 12
5. Miscellaneous Examples

3.6.5. What type is this object? ................................................................. 12
3.6.6. What methods, properties, interfaces, and services are supported? ... 15
3.6.7. Languages other than Basic

3.6.7.1. CreateUnoService .................................................................. 15
3.6.7.2. ThisComponent .................................................................... 16
3.6.7.3. StarDesktop .......................................................................... 16
3.6.8. Accessing methods and properties ............................................. 17

4. Examples ......................................................................................... 19

4.1. Debugging And Inspecting Macros

4.1.1. Determine Document Type ....................................................... 19
4.1.2. Display Object Methods And Properties .................................... 20

4.2. X-Ray .......................................................................................... 21

4.3. Dispatch: Using Universal Network Objects (UNO)

4.3.1. The Dispatcher Changed In Version 1.1 ..................................... 23
4.3.2. Using the dispatcher requires a user interface .............................. 23
4.3.2.1. Modifying the menu a dispatcher example ............................. 24

4.4. Intercept menu commands using Basic ......................................... 25

5. Miscellaneous Examples

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) ............................ 35
5.8.1. Close OpenOffice And/Or Documents ...................................... 35
5.8.1.1. What if the file is modified? .................................................. 36
5.8.2. Load A Document From A URL .............................................. 37
5.8.2.1. A complete example ........................................................... 38
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 ......... 40
5.8.6. Error handling on load ............................................................ 41
5.8.7. Mail Merge example, merge all documents in a directory .......... 42

5.9. Loading/Inserting a graphic into your document ............................ 43
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9.1. Convert a linked graphic to an embedded graphic</td>
<td>45</td>
</tr>
<tr>
<td>5.9.2. Danny Brewer embeds a graphic</td>
<td>45</td>
</tr>
<tr>
<td>5.9.3. Embed a graphics using a dispatch</td>
<td>46</td>
</tr>
<tr>
<td>5.9.4. Embed a graphics directly</td>
<td>47</td>
</tr>
<tr>
<td>5.9.5. Duplicate an existing graphic</td>
<td>48</td>
</tr>
<tr>
<td>5.10. Setting Margins</td>
<td>48</td>
</tr>
<tr>
<td>5.10.1. Setting the paper size</td>
<td>49</td>
</tr>
<tr>
<td>5.11. Calling an external program (Internet Explorer) using OLE</td>
<td>49</td>
</tr>
<tr>
<td>5.12. Use the Shell command for files containing spaces</td>
<td>50</td>
</tr>
<tr>
<td>5.13. Read And Write Number In File</td>
<td>50</td>
</tr>
<tr>
<td>5.14. Create Number Format Style</td>
<td>51</td>
</tr>
<tr>
<td>5.14.1. View Supported Number Format Styles</td>
<td>52</td>
</tr>
<tr>
<td>5.15. Return the Fibonacci array</td>
<td>53</td>
</tr>
<tr>
<td>5.16. Insert Text At Bookmark</td>
<td>54</td>
</tr>
<tr>
<td>5.17. Saving And Exporting A Document</td>
<td>54</td>
</tr>
<tr>
<td>5.18. User Fields</td>
<td>55</td>
</tr>
<tr>
<td>5.18.1. Document Information</td>
<td>56</td>
</tr>
<tr>
<td>5.18.2. Text Fields</td>
<td>57</td>
</tr>
<tr>
<td>5.18.3. Master Fields</td>
<td>58</td>
</tr>
<tr>
<td>5.18.4. Removing Text Fields</td>
<td>62</td>
</tr>
<tr>
<td>5.18.5. Insert a URL into a Calc cell</td>
<td>62</td>
</tr>
<tr>
<td>5.18.6. Adding a SetExpression TextField</td>
<td>62</td>
</tr>
<tr>
<td>5.19. User Defined Data Types</td>
<td>63</td>
</tr>
<tr>
<td>5.20. Spell Check, Hyphenation, and Thesaurus</td>
<td>64</td>
</tr>
<tr>
<td>5.21. Changing The Mouse Cursor</td>
<td>65</td>
</tr>
<tr>
<td>5.22. Setting The Page Background</td>
<td>66</td>
</tr>
<tr>
<td>5.23. Manipulating the clipboard</td>
<td>66</td>
</tr>
<tr>
<td>5.23.1. Copy Spreadsheet Cells With The Clipboard</td>
<td>66</td>
</tr>
<tr>
<td>5.23.2. Copy Spreadsheet Cells Without The Clipboard</td>
<td>67</td>
</tr>
<tr>
<td>5.23.3. Getting the content-type of the Clipboard</td>
<td>68</td>
</tr>
<tr>
<td>5.23.4. Storing a string to the clipboard</td>
<td>69</td>
</tr>
<tr>
<td>5.23.5. View the clipboard as text</td>
<td>69</td>
</tr>
<tr>
<td>5.23.6. An alternative to the clipboard transferable content</td>
<td>70</td>
</tr>
<tr>
<td>5.24. Setting The Locale (Language)</td>
<td>71</td>
</tr>
<tr>
<td>5.25. Setting the locale for selected text</td>
<td>72</td>
</tr>
<tr>
<td>5.26. Auto Text</td>
<td>74</td>
</tr>
<tr>
<td>5.27. Decimal Feet To Fraction</td>
<td>76</td>
</tr>
<tr>
<td>5.27.1. Convert number to words</td>
<td>80</td>
</tr>
<tr>
<td>5.28. Sending Email</td>
<td>85</td>
</tr>
<tr>
<td>5.29. Macro libraries</td>
<td>87</td>
</tr>
<tr>
<td>5.29.1. The vocabulary</td>
<td>87</td>
</tr>
<tr>
<td>5.29.1.1. Library container</td>
<td>87</td>
</tr>
<tr>
<td>5.29.1.2. Libraries</td>
<td>87</td>
</tr>
</tbody>
</table>
6. Calc macros

6.1. Is this a spreadsheet document? ................................................................. 119
6.2. Display cell value, string, or formula ......................................................... 119
6.3. Set cell value, format, string, or formula .................................................... 120
6.3.1. Reference a cell in another document .................................................... 120
6.4. Clear a cell .................................................................................................. 120
6.5. Selected text, what is it? ............................................................................. 121
6.5.1. Simple example processing selected cells .............................................. 122
6.5.2. Get the active cell and ignore the rest ................................................... 123
6.5.3. Select a Cell .............................................................................................. 124
6.6. Human readable address of cell .................................................................. 125
6.7. Insert formatted date into cell ..................................................................... 126
6.7.1. A shorter way to do it .............................................................................. 127
6.8. Selected text, what is it? ............................................................................. 128
6.8.1. Simple example processing selected cells .............................................. 129
6.8.2. Get the active cell and ignore the rest ................................................... 130
6.8.3. Select a Cell .............................................................................................. 131
6.9. Human readable address of cell .................................................................. 132
6.10. Insert formatted date into cell .................................................................... 133
6.10.1. A shorter way to do it ............................................................................ 134
6.11. Simple example processing selected cells .................................................. 135
6.11.1. Get the active cell and ignore the rest ................................................... 136
6.11.2. Select a Cell ............................................................................................ 137
7. Writer Macros

7.1. Selected Text, What Is It?

7.2. Text Cursors, What Are They?

7.3. Andrew's Selected Text Framework

6. Table of Contents

6.1. Display selected range in message box

6.2. Fill selected range with text

6.3. Some stats on a selected range

6.4. Database range

6.5. Set selected cells to a database range

6.6. Delete database range

6.7. Table borders

6.8. Sort range

6.9. Display all data in a column

6.10. Using Outline (Grouping) Methods

6.11. Protecting your data

6.12. Setting header and footer text

6.13. Copying spreadsheet cells

6.14. Selecting a named range

6.15. Copy entire sheet to a new document

6.16. Select a named range

6.17. Select an entire column

6.18. Select an entire row

6.19. Convert data in column format into rows

6.20. Toggle Automatic Calculation

6.21. Which cells are used in a sheet?

6.22. Searching a Calc document

6.23. Printing a Calc range

6.24. Is a cell merged?

6.25. Copying spreadsheet cells

6.26. Write your own Calc functions

6.27. User defined Calc functions

6.28. Evaluating the argument

6.29. What is the return type

6.30. Do not modify other cells in the sheet

7. Writer Macros

7.1. Selected Text, What Is It?

7.1.1. Is the cursor in a text table?

7.1.2. Can I check the current selection for a TextTable or Cell?

7.2. Text Cursors, What Are They?

7.2.1. You can not move a cursor to a TextTable anchor

7.2.2. Inserting something before (or after) a text table

7.2.3. You can move a cursor to a Bookmark anchor

7.2.4. Insert Text At Bookmark

7.3. Andrew's Selected Text Framework

7.3.1. Is Text Selected?

7.3.2. How To Get A Selection

7.3.3. Selected Text, Which End Is Which

7.3.4. The Selected Text Framework Macro

7.3.4.1. The Rejected Framework
8.1. Finding text tables ................................................................. 217
  8.1.1. Where is the text table? ................................................ 217
  8.1.2. Enumerating text tables ............................................... 219
8.2. Enumerating cells in a text table .......................................... 220
  8.2.1. Simple text tables ...................................................... 221
  8.2.2. Formatting a simple text table ...................................... 221
  8.2.3. What is a complex text table ....................................... 223
  8.2.4. Enumerating cells in any text table ............................... 224
8.3. Getting data from a simple text table .................................... 225
8.4. Table cursors and cell ranges .............................................. 225
8.5. Cell ranges ........................................................................ 225
  8.5.1. Using a cell range to clear cells .................................... 226
8.6. Chart data .......................................................................... 226
8.7. Column Widths ................................................................... 226
8.8. Setting the optimal column width ........................................ 227
8.9. How wide is a text table? ................................................... 227
8.10. The cursor in a text table .................................................. 228
  8.10.1. Move the cursor after a text table ................................. 229
8.11. Creating a table ............................................................... 230
8.12. A table with no borders .................................................... 231
9. Formatting macros .................................................................. 233
  9.1. String and array utilities .................................................. 233
    9.1.1. Special characters and numbers in strings ................. 236
    9.1.2. Arrays of strings ..................................................... 240
  9.2. Utilities to find code sections ............................................ 242
  9.3. Formatting using character styles ...................................... 243
  9.4. The main module ............................................................ 246
    9.4.1. How to use the macros ............................................. 248
    9.4.2. The worker macro .................................................. 250
14. Language

14.1. Comments ................................................................. 303
14.2. Variables ................................................................. 303
   14.2.1. Names ............................................................. 303
   14.2.2. Declaration ....................................................... 303
   14.2.3. Evil Global Variables And Statics ......................... 305
   14.2.4. Types ............................................................. 305
      14.2.4.1. Boolean Variables ........................................ 307
      14.2.4.2. Integer Variables ......................................... 307
      14.2.4.3. Long Integer Variables ................................. 308
      14.2.4.4. Currency Variables ...................................... 308
      14.2.4.5. Single Variables .......................................... 308
      14.2.4.6. Double Variables ......................................... 308
      14.2.4.7. String Variables ......................................... 308
   14.2.5. Object, Variant, Empty, and Null .......................... 309
   14.2.6. Should I Use Object Or Variant ......................... 309
   14.2.7. Constants ........................................................ 310
   14.2.8. Arrays ........................................................... 310
      14.2.8.1. Option Base ............................................. 310
      14.2.8.2. LBound(arrayname[,Dimension]) ..................... 310
      14.2.8.3. UBound(arrayname[,Dimension]) ..................... 311
      14.2.8.4. Is This Array Defined ................................. 311
   14.2.9. DimArray, Changing The Dimension ....................... 311
   14.2.10. ReDim, Changing The Number Of Elements ............... 311
   14.2.11. Testing Objects .............................................. 313
   14.2.12. Comparison Operators ...................................... 313
14.3. Functions and SubProcedures .................................... 314
   14.3.1. Optional Parameters ......................................... 314
   14.3.2. Parameters By Reference Or Value ........................ 315
   14.3.3. Recursion ...................................................... 316
14.4. Flow Control ......................................................... 316
   14.4.1. If Then Else .................................................. 316
   14.4.2. IIF .............................................................. 317
   14.4.3. Choose ........................................................ 317
   14.4.4. For...Next ................................................. 317
   14.4.5. Do Loop ....................................................... 318
   14.4.6. Select Case .................................................. 319
      14.4.6.1. Case Expressions ....................................... 319
      14.4.6.2. Incorrect Simple Example ............................ 319
      14.4.6.3. Incorrect Range Example ............................. 320
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.4.6.4</td>
<td>Incorrect Range Example</td>
<td>320</td>
</tr>
<tr>
<td>14.4.6.5</td>
<td>Ranges, The Correct Way</td>
<td>320</td>
</tr>
<tr>
<td>14.4.7</td>
<td>While...Wend</td>
<td>321</td>
</tr>
<tr>
<td>14.4.8</td>
<td>GoSub</td>
<td>321</td>
</tr>
<tr>
<td>14.4.9</td>
<td>GoTo</td>
<td>322</td>
</tr>
<tr>
<td>14.4.10</td>
<td>On GoTo</td>
<td>322</td>
</tr>
<tr>
<td>14.4.11</td>
<td>Exit</td>
<td>323</td>
</tr>
<tr>
<td>14.4.12</td>
<td>Error Handling</td>
<td>323</td>
</tr>
<tr>
<td>14.4.12.1</td>
<td>Specify How To Handle The Error</td>
<td>324</td>
</tr>
<tr>
<td>14.4.12.2</td>
<td>Write The Error Handler</td>
<td>324</td>
</tr>
<tr>
<td>14.4.12.3</td>
<td>An Example</td>
<td>325</td>
</tr>
<tr>
<td>14.5</td>
<td>Miscellaneous</td>
<td>326</td>
</tr>
<tr>
<td>15</td>
<td>Compatibility With Visual BASIC</td>
<td>329</td>
</tr>
<tr>
<td>15.1</td>
<td>Data types</td>
<td>330</td>
</tr>
<tr>
<td>15.2</td>
<td>Variables</td>
<td>330</td>
</tr>
<tr>
<td>15.3</td>
<td>Arrays</td>
<td>331</td>
</tr>
<tr>
<td>15.4</td>
<td>Subroutine and Function Constructs</td>
<td>331</td>
</tr>
<tr>
<td>15.5</td>
<td>Operators</td>
<td>331</td>
</tr>
<tr>
<td>15.6</td>
<td>Subroutines and Functions</td>
<td>332</td>
</tr>
<tr>
<td>15.6.1</td>
<td>Numerical Subroutines and Functions</td>
<td>332</td>
</tr>
<tr>
<td>15.7</td>
<td>Compatibility mode and private variables</td>
<td>333</td>
</tr>
<tr>
<td>16</td>
<td>Operators and Precedence</td>
<td>335</td>
</tr>
<tr>
<td>17</td>
<td>String Manipulations</td>
<td>337</td>
</tr>
<tr>
<td>17.1</td>
<td>Remove Characters From String</td>
<td>338</td>
</tr>
<tr>
<td>17.2</td>
<td>Replace Text In String</td>
<td>338</td>
</tr>
<tr>
<td>17.3</td>
<td>Printing The ASCII Values Of A String</td>
<td>339</td>
</tr>
<tr>
<td>17.4</td>
<td>Remove All Occurrences Of A String</td>
<td>339</td>
</tr>
<tr>
<td>18</td>
<td>Numeric Manipulations</td>
<td>341</td>
</tr>
<tr>
<td>19</td>
<td>Date Manipulations</td>
<td>343</td>
</tr>
<tr>
<td>20</td>
<td>File Manipulations</td>
<td>345</td>
</tr>
<tr>
<td>21</td>
<td>Operators, Statements, and Functions</td>
<td>347</td>
</tr>
<tr>
<td>21.1</td>
<td>Subtraction operator (-)</td>
<td>347</td>
</tr>
<tr>
<td>21.2</td>
<td>Multiplication operator (*)</td>
<td>347</td>
</tr>
<tr>
<td>21.3</td>
<td>Addition operator (+)</td>
<td>347</td>
</tr>
<tr>
<td>21.4</td>
<td>Power operator (^)</td>
<td>348</td>
</tr>
<tr>
<td>21.5</td>
<td>Division operator (/)</td>
<td>348</td>
</tr>
<tr>
<td>21.6</td>
<td>AND Operator</td>
<td>349</td>
</tr>
<tr>
<td>21.7</td>
<td>Abs Function</td>
<td>349</td>
</tr>
<tr>
<td>21.8</td>
<td>Array Function</td>
<td>350</td>
</tr>
<tr>
<td>21.9</td>
<td>Asc Function</td>
<td>351</td>
</tr>
<tr>
<td>21.10</td>
<td>ATN Function</td>
<td>351</td>
</tr>
<tr>
<td>21.11</td>
<td>Beep Statement</td>
<td>352</td>
</tr>
<tr>
<td>21.12</td>
<td>Blue Function</td>
<td>352</td>
</tr>
</tbody>
</table>
21.13. ByVal Keyword .......................................................... 353
21.15. CBool Function .......................................................... 354
21.16. CByte Function .......................................................... 354
21.17. CDate Function .......................................................... 355
21.18. CDateFromIso Function ................................................ 355
21.19. CDateToIso Function .................................................. 356
21.20. CDbl Function ............................................................. 356
21.21. ChDir statement is deprecated ...................................... 357
21.22. ChDrive statement is deprecated ................................. 357
21.23. Choose Function ......................................................... 357
21.24. Chr Function .............................................................. 358
21.25. CInt Function ............................................................. 359
21.26. CLng Function ............................................................ 359
21.27. Close Statement .......................................................... 360
21.28. Const Statement .......................................................... 360
21.29. ConvertFromURL Function .......................................... 361
21.30. ConvertToURL Function ............................................... 361
21.31. Cos Function .............................................................. 362
21.32. CreateUnoDialog Function ......................................... 362
21.33. CreateUnoService Function ........................................ 363
21.34. CreateUnoStruct Function ........................................... 364
21.35. CSng Function ........................................................... 364
21.36. CStr Function ........................................................... 365
21.37. CurDir Function ........................................................... 365
21.38. Date Function ............................................................ 366
21.39. DateSerial Function .................................................... 366
21.40. DateValue Function .................................................... 367
21.41. Day Function ............................................................. 368
21.42. Declare Statement ....................................................... 368
21.43. DefBool Statement ....................................................... 369
21.44. DefDate Statement ..................................................... 370
21.45. DefDb1 Statement ....................................................... 370
21.46. DefInt Statement ........................................................ 370
21.47. DefLng Statement ....................................................... 371
21.48. DefObj Statement ....................................................... 371
21.49. DefVar Statement ....................................................... 371
21.50. Dim Statement ........................................................... 372
21.51. DimArray Function ..................................................... 373
21.52. Dir Function .............................................................. 373
21.53. Do...Loop Statement .................................................. 375
21.54. End Statement ........................................................... 375
21.55. Environ Function ....................................................... 376
21.56.  EOF Function ........................................................................................................ 377
21.57.  EqualUnoObjects Function .................................................................................... 377
21.58.  EQV Operator ........................................................................................................ 378
21.59.  Erl Function ............................................................................................................ 379
21.60.  Err Function ........................................................................................................... 379
21.61.  Error statement does not work as indicated ................................................................. 380
21.62.  Error Function ........................................................................................................ 380
21.63.  Exit Statement ........................................................................................................ 381
21.64.  Exp Function .......................................................................................................... 381
21.65.  FileAttr Function ................................................................................................... 382
21.66.  FileCopy Statement ................................................................................................ 383
21.67.  FileDateTime Function ........................................................................................... 383
21.68.  FileExists Function ............................................................................................... 384
21.69.  FileLen Function .................................................................................................... 384
21.70.  FindObject Function .............................................................................................. 385
21.71.  FindPropertyObject Function ................................................................................. 386
21.72.  Fix Function ........................................................................................................... 386
21.73.  For...Next Statement ................................................................................................ 387
21.74.  Format Function ...................................................................................................... 387
21.75.  FreeFile Function .................................................................................................... 389
21.76.  FreeLibrary Function ............................................................................................ 390
21.77.  Function Statement ................................................................................................ 390
21.78.  Get Statement ......................................................................................................... 391
21.79.  GetAttr Function .................................................................................................... 392
21.80.  GetProcessServiceManager Function ..................................................................... 393
21.81.  GetSolarVersion Function ....................................................................................... 393
21.82.  GetSystemTicks Function ........................................................................................ 394
21.83.  GlobalScope Statement .......................................................................................... 394
21.84.  GoSub Statement ..................................................................................................... 395
21.85.  GoTo Statement ....................................................................................................... 395
21.86.  Green Function ....................................................................................................... 396
21.87.  HasUnoInterfaces Function ....................................................................................... 396
21.88.  Hex Function .......................................................................................................... 397
21.89.  Hour Function ........................................................................................................ 398
21.90.  If Statement ............................................................................................................ 398
21.91.  IIF Statement ......................................................................................................... 399
21.92.  Imp Operator .......................................................................................................... 399
21.93.  Input Statement ....................................................................................................... 400
21.94.  InputBox Function .................................................................................................. 401
21.95.  InStr Function ....................................................................................................... 402
21.96.  Int Function ............................................................................................................ 403
21.97.  IsArray Function ..................................................................................................... 403
21.98.  IsDate Function ...................................................................................................... 404
21.99. IsEmpty Function ........................................................................................................... 404
21.100. IsMissing Function ....................................................................................................... 405
21.101. IsNull Function ............................................................................................................. 405
21.102. IsNumeric Function ..................................................................................................... 406
21.103. IsObject Function ....................................................................................................... 406
21.104. IsUnoStruct Function .................................................................................................. 407
21.106. LBound Function ........................................................................................................ 408
21.107. LCase Function .......................................................................................................... 408
21.108. Left Function .............................................................................................................. 408
21.109. Len Function ............................................................................................................... 409
21.110. Let Function ................................................................................................................ 409
21.111. Line Input Statement ................................................................................................... 410
21.112. Loc Function ............................................................................................................... 410
21.113. Lof Function ................................................................................................................ 411
21.114. Log Function .............................................................................................................. 412
21.115. Loop Statement .......................................................................................................... 412
21.116. LSet Statement ........................................................................................................... 413
21.117. LTrim Function ........................................................................................................... 414
21.118. Private Keyword ........................................................................................................ 414
21.119. Public Keyword .......................................................................................................... 415
21.120. Red Function ............................................................................................................... 415
21.121. RSet Statement .......................................................................................................... 416
21.122. Shell Function ............................................................................................................ 416
21.123. UBound Function ....................................................................................................... 418
21.124. UCase Function ......................................................................................................... 419
21.125. URL Notation And Filenames .................................................................................... 419
   21.125.1. URL Notation ......................................................................................................... 419
   21.125.2. Paths With Spaces And Other Special Characters .................................................... 419
   21.125.3. Anchoring To The Home Directory On Unix ............................................................ 419
22. Other languages .................................................................................................................. 421
   22.1. C# ............................................................................................................................... 421
   22.2. Visual Basic Programmers .......................................................................................... 422
       22.2.1. ActiveWorkbook .................................................................................................. 422
       22.2.2. ActiveSheet and ActiveCell ............................................................................... 423
23. Index ...................................................................................................................................... 425
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 covers significantly fewer commands, is not as accurate, or indept. In other words, consider obtaining a copy of my published book “OpenOffice.org Macros Explained” (see http://www.pitonyak.org/book/”).

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 a documenting macros requested by the user community. My quest to understand Oo 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. 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.

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>UNO</td>
<td>Universal Network Objects</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>OLE</td>
<td>Object Linking and Embedding</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>OoO</td>
<td>OpenOffice.org</td>
</tr>
</tbody>
</table>
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 (Prior to version 2.0, use Tools | Macros | Macro) to open the Macro dialog. Expand the Tools library in the OpenOffice.org library container. Inspect the Debug module – some good examples include WritedbgInfo(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/](http://www.pitonyak.org/book/)). This is a great introduction for a beginner and it is free.

2.2. On line Resources

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

- [http://www.openoffice.org](http://www.openoffice.org) (the main link)
- [http://www.oooforum.org](http://www.oooforum.org) (if you need help with your macros this is the perfect place to ask, probably one of the best supported forums on the web)
- [http://api.openoffice.org/docs/common/ref/com/sun/star/module-ix.html](http://api.openoffice.org/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)
• http://www.pitonyak.org/AndrewMacro.odt (Latest copy of this document)
• http://www.pitonyak.org/book/ (buy a 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://docs.sun.com (Search for StarOffice and find the StarBasic documentation)
• http://api.openoffice.org (This site takes some getting used to but it is very complete)
• http://documentation.openoffice.org Download the “How To” document referenced below.
  http://documentation.openoffice.org/HOW_TO/various_topics/How_to_use_basic_macros.sxw
• http://udk.openoffice.org (Here you will find advanced information about UNO)
• http://udk.openoffice.org/common/man/tutorial/office_automation.html (OLE)
• http://ooextras.sourceforge.net/ (Examples)
• http://disemia.com/software/openoffice/ (Examples, mainly for the Calculator)
• http://kienlein.com/pages/oo.html (Examples)
• http://www.darwinwars.com/lunatic/bugs/oo_macros.html (Examples)
• http://sourceforge.net/project/showfiles.php?group_id=43716 (Examples)
• http://www.kargs.net/openoffice.html (Examples)
• http://www.8daysaweek.co.uk/ (Examples and Documentation)
• http://ooo.ximian.com/lxr/ (you can dig through the source code online here)
• http://homepages.paradise.net.nz/hillview/OOo/ (numerous excellent macros here including reveal codes macros, key macros, and, information on converting from MS Office)

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.

<table>
<thead>
<tr>
<th>Link</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.pitonyak.org/AndrewMacro.odt">http://www.pitonyak.org/AndrewMacro.odt</a></td>
<td>English</td>
</tr>
<tr>
<td><a href="http://www.pitonyak.org/AndrewMacroGerman.sxw">http://www.pitonyak.org/AndrewMacroGerman.sxw</a>*</td>
<td>German</td>
</tr>
</tbody>
</table>

* The translations are not up to date, especially the German version.
3. Getting started: concepts

The first chapter of my book, which is available as a free download, is a more complete user friendly getting started. I consider it a much 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 exhausting error-prone manual labor. Currently, the automatic actions are most easily created by writing Macros in OOo Basic. The new scripting framework in OOo version 2 should ease the use of other languages, but Basic is still the most 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
• 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. Each routine provides can call other routines. The difference between a Sub and a Function, is that a Function returns a value and a Sub does not. The macro in Listing 3.2 obtains the text string from the function named HellowWorldString.
**Listing 3.2:** “Hello world” using a subroutine and a function.

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

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

A module contains a collection of procedures. A library contains modules. A document may contain a library – or more than one library. A library can also exist at the application level.

### 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, Constants, Strings and numerical Types

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 variables' type determines what it can store. Attempting to store the wrong type of data into a variable, will frequently cause an error. Technically speaking, this error is called an Exception. Before a variable can be used, it must be given a name. A simple variable declaration is as follows:

**Listing 3.3:** Declare a simple variable.

```vbnet
Dim <variablename> as <Type>
```

Variables are dynamic – and without them you can not store data! Like a box, a variable's content can change. I have a box in my room and only I can change what the box contains. Well, I am married, so my wife can also change what my box contains. Technically speaking, a variable's scope specifies where the variable is accessible. You can define a variable to be accessible only in one subroutine, for an entire module, and even globally for all libraries. The keywords “Global”, “Public”, and “Private” are used to define a variable's scope. the location in your macro where the variable is declared also affects its scope.

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.

```vbnet
Const <constantname> = <constantvalue>
```
Oo Basic supports many different 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 14 Language on page 318 and also refer to the Oo 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.

```vbscript
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
```

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.

```vbscript
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** The Developer's Guide says that the Variant type should be used rather than the object type. This is mentioned in more detail on page 309.

**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.

```vbscript
createUnoService("com.sun.star.frame.Desktop")
```
3.6. Everyone keeps talking about UNO, what is it?

UNO (Universal Network Object) was created to allow all different environments to play nice with each other. Why? Because different programming languages and environments may have a different representation for the same type of data. Even integers, as simple as they are, can be represented in a different manner on different computers and in different programming languages.

- UNO defines numerous basic types such as strings, integers, etc. (so they are the same in different environments).
- UNO objects may have methods (a method may return a value and may take arguments).
- UNO objects may have properties. A property may be another UNO object, or a simple type. Also, a property may be optional.
- UNO objects are defined using a complicated (to most people) Interface Definition Language (UNO IDL). Although you may not like looking at a UNOIDL, it defines the properties and methods that an object supports.
- Assume that I have a UNO environment in Basic, and another in Java. For an UNO environment to use an object, all it requires is the definition of the object (UNOIDL). The environments can easily pass objects back and forth.

I use the OOo API to interact with OOo. The API may return:

- UNO intrinsic type such as a floating point number, integer, etc.
- Constants are values, usually numeric, associated with a name. For example, normal text has a font weight of `com.sun.star.awt.FontWeight.NORMAL`, which is 100.0.
- Enumerations, use a name such as `com.sun.star.awt.FontSlant.ITALIC`, but they are typically associated with an integer even though the numerical value is not listed.
- Structures are objects with properties but no methods. The fact that an object has no methods does not imply that it is a structure, however. The object is defined to be a structure.
- UNO with methods and/or properties.

3.6.1. Structures

Structures that are defined by OpenOffice.org are testable using the method `IsUNOStruct()`.

Listing 3.7: Test for an UNO structure.

```vba
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

Although you can define and use your own structures, IsUNOStruct() does not recognize them as structures.

### 3.6.2. Interfaces

Almost all OOO objects support services and interfaces. When I use the word interface, as it relates to an object, the word always means a 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.*

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

The macro in Listing 3.18 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.
3.6.3. Services

When I use the word service, as it relates to an object, the word always means 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.

3.6.4. Interfaces and services

The point of all of this is that although the definition of almost every service and interface is available, it is not the perfect solution for quickly determining everything about an object. The BASIC IDE allows you to set break points in your code and to examine variables. Many people inspect objects using a freely available object inspection library called Xray. I have my own set of object inspection routines as outlined in my book OpenOffice.org Macros Explained.

If an object implements the XServiceInfo interface, then you can ask the object if it supports a specific service (see http://api.openoffice.org/docs/common/ref/com/sun/star/lang/XServiceInfo.html). This capability is commonly used to determine if a document is of a specific type.

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
```

3.6.5. 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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsArray</td>
<td>Is the parameter an array?</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>Is the parameter an uninitialized variant type?</td>
</tr>
<tr>
<td>IsNull</td>
<td>Does the parameter contain “no” data?</td>
</tr>
</tbody>
</table>
A variables type name can also provide information as to a variables properties.

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

<table>
<thead>
<tr>
<th>Type</th>
<th><code>TypeName()</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant</td>
<td>“Empty” or name of contained object</td>
</tr>
<tr>
<td>Object</td>
<td>“Object” even if it is null. Same for structures.</td>
</tr>
<tr>
<td>regular type</td>
<td>regular type name such as “String”</td>
</tr>
<tr>
<td>array</td>
<td>name followed by “()”</td>
</tr>
</tbody>
</table>

Listing 3.9 demonstrates the statements in Table 3.2, and Figure 3.1 shows a portion of the output.

**Listing 3.9: Inspect variables.**

```vbnet
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**

```vbnet
    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
```

13
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)
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
3.6.6. What methods, properties, interfaces, and services are supported?

Both a service and a structure have a type name of “Object”. For a structure, you need to
know what kind so that you can find the supported properties – use the API website to check
the IDL. UNO objects, on the other hand, usually support the ServiceInfo, which provides
information about the service. The getImplementationName() object method returns the fully
qualified name of the object. Use the fully qualified name to search Google or the
Developer's Guide for more information. The getSupportedServiceNames() object method
returns a sequence of all interfaces supported by the object. A common method finding out
what an object can do is to call the following three methods:

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 WritedbgInfo(object) macro
inserts object debug information into an open writer document.

3.6.7. Languages other than Basic

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

3.6.7.1. 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")

15
Rem Create a desktop object.
Set oDesk = oManager.createInstance("com.sun.star.frame.Desktop")

3.6.7.2. ThisComponent

In StarBasic, ThisComponent references the current the document, or the document that caused a macro to be called. ThisComponent is set once 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. Unfortunately, this will not return a document object if the Basic IDE or the help window is current, but this is less likely when a language other than Basic is used.

There are times when you might want to use StarDesktop.getCurrentComponent() rather than simply using ThisComponent, but your macro will fail when run from the Basic IDE.

3.6.7.3. StarDesktop

OOo Basic defines a few global variables and provides them as a convenience to programmers. The StarDesktop variable references the desktop object which is essentially the primary OOo 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 3.1).

Figure 3.1: The desktop contains components.

Getting back to the macro in Listing 3.18, the StarDesktop provides access to the currently open components. The getCurrentComponent() method 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


3.6.8. 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.

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

Listing 3.15: O0o Basic allows you to access properties directly.

```basic
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.

Listing 3.16: Access some properties using get methods.

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

Code that uses the get and set methods is easier to translate into other languages. StarBasic also allows certain properties to act as an array, even if the property is not an array. The Sheets property in a Calc document is a good example of this. In Calc, both of the following accomplish the same thing, but only the second example may work outside of StarBasic.

Listing 3.17: O0o Basic allows you to access some properties as an array.

```basic
oDoc.sheets(1)
oDoc.getSheets().getIndex(1)
```
3.7. Summary

Writing macros in OpenOffice.org (OOo) is a complicated task with a steep learning curve. The problem is not the base language or the environment, it is the OOo application programming interface (API). The base language refers to the syntax and the commands that are not used to interact with an OOo document.

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

```vbnet
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 and therefore require the OOo API. In this context, the term API refers to the objects and the each object's properties and methods.

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

```vbnet
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 3.18 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 OOO, 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() = "simpress"
  ElseIf oDoc.SupportsService(sBase) Then
    GetDocumentType() = "sbase"
  End If

NODOCUMENTTYPE:
  If Err <> 0 Then
    GetDocumentType() = ""
    Resume GOON
  GOON:
End If

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

4.1.2. Display Object Methods And Properties

The following Subroutine displays the names of either the supported methods or properties of an object. If the second parameter is the empty string, the method names are printed in a dialog. Any other value and the property names are printed. Because the displayed list is frequently very long, the list is displayed in manageable chunks.
Listing 4.3: Display methods, interfaces, or properties of an object.

'A subroutine to display all the methods or properties of an input
'Author: Tony Bloomfield [tonyb.lx@btinternet.com]
'Modified: hal@thresholddigital.com support services and verify obj.
'Modified: A Pitonyak to fix declaration errors
Sub DisplayMethods(oObj As Object, sWhat As String)
    Dim sMethodList As String, sMsgBox As String
    Dim i As Integer, ep As Integer
    Dim EOL As Boolean
    If IsNull(oObj) Then
        Print "Object does not exist."
    Else
        If sWhat = "m" Then
            sMethodList = oObj.DBG_Methods
        ElseIf sWhat = "s" Then
            sMethodList = oObj.DBG_SupportedInterfaces
        ElseIf sWhat = "p" Then
            sMethodList = oObj.DBG_Properties
        End If
        fs = 1
        EOL = FALSE
        While fs <= Len(sMethodList)
            sMsgBox = ""
            For i = 0 to 15
                ep = InStr(fs, sMethodList, ";")
                If ep = 0 then
                    ep = Len(sMethodList)
                End If
                sMsgBox = sMsgBox & Mid$(sMethodList, fs, ep - fs) & Chr$(13)
                fs = ep + 1
            Next i
            MsgBox sMsgBox
        Wend
    End If
End Sub

4.2. X-Ray
Bernard Marcelly wrote a tool called X-Ray, which displays object information in a dialog. X-Ray tool is available for download from [http://www.ooomacros.org/dev.php](http://www.ooomacros.org/dev.php), and is highly recommended by many people!
I do not use X-Ray because I wrote own tool before X-Ray was available. My original tool, the SimpleObjectBrowser, is available in the Pitonyak library on my web site. More recently, I created an Object inspector for my book “OpenOffice.org Macros Explained” – unfortunately, you must purchase the book to get a copy of my object inspector. The publisher's page is shown below, but the book is also available from Amazon.com.

http://www.hentzenwerke.com/catalogpricelists/oome.htm

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 using the OLD dispatch mechanism. !!!Remove the older methods!!!.

Listing 4.4: Use older outdated dispatch method to perform the undo operation.

Sub UnoUndo
    PerformDispatch(ThisComponent.CurrentController.Frame, ".uno:Undo")
End Sub

Sub PerformDispatch(oObj As Object, uno$)
    Dim oParser As Object
    Dim oUrl As New com.sun.star.util.URL
    Dim oDisp As Object
    REM The UNO service is represented as a URL
    oUrl.Complete = uno$

    REM Parse the URL as required
    oParser = createUnoService("com.sun.star.util.URLTransformer")
    oParser.parseStrict(oUrl)

    REM See if the current Frame supports this UNO command
    oDisp = oObj.queryDispatch(oUrl,"",0)
    If (Not IsNull(oDisp)) Then
        oDisp.dispatch(oUrl,noargs())
    Else
        MsgBox uno$ & " was not found"
    End If
End Sub
Starting in release 1.1, this can be written as

**Listing 4.5: Use the new dispatch method to perform the undo operation.**

```vba
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 4.6: Use the new dispatch method to export a Write document to PDF.**

```vba
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 4.7: Use the new dispatch method to go to a cell, copy, and paste.**

```vba
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 Changed In Version 1.1

Hal Vaughan asked: “Is it just me, or is there a reason the dispatcher won't work with most functions under 1.0.3?” Mathias Bauer answered.

The dispatcher uses some functionality not present in OOo1.0 (dispatch helper). The code to execute a dispatch with parameters, for example.

Will Dispatch Names Change?

This is another Hal Vaughan question and Mathias Bauer answered:

Macros using dispatch names rather than numbers will not change between OOO versions.

### 4.3.2. Using the dispatcher requires a user interface.

Yet another Hal Vaughan question and Mathias Bauer answer.

Is there any reason to use the regular API calls instead of calling the dispatcher with the function name?
Dispatch calls do not work on a document without a UI. If OOO is run in a real "server" mode (something that could happen in the OOO2.0 release) 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:

Recording macros

As a workaround when a certain task can not be done by any “real” API (because it does not exist or it is broken).

### 4.3.2.1. Modifying the menu – a dispatcher example

As of OpenOffice.org 1.1.1, it is not possible to assign a macro to a menu item using the API – this should be possible in version 2.0:

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, assigning a macro to that ID is not – still waiting for OOo 2.0! This is possible now, however, using a DispatchProviderInterceptor, which can be compared to a Listener or Handler.

With a DispatchProviderInterceptor we can listen for Dispatch commands. It is possible to intercept almost every command. Using the 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 on the oooforum, toggles the visibility of tool bars on and off.

Listing 4.8: Toggle the visibility of a tool bar:

```vba
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")
```

24
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 4.9: 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

25
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 ".uno: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 careful! 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
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 Oo
    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

5.3. Iterate Through All Open Documents

Listing 5.4: Iterate through all of the open documents.

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”.

30
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$ as String
        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"
    '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"
    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
```
5.7.2. OOo Locale

Listing 5.11: Obtain the current OpenOffice.org locale.

```vba
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 OOLangue as string
    OOLangue= oSet.getByName("ooLocale") 'en-US
    OOlang=lcase(Left(trim(OOLangue),2)) 'en
End Function
```

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 XClosable 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]

**Listing 5.12: Proper method to close an OpenOffice.org document.**

```javascript
If oDoc.supportsService("com.sun.star.frame.XModel")
  If HasUnoInterfaces(oDoc, "com.sun.star.util.XCloseable") Then
    oDoc.close(true)
  Else
    oDoc.dispose()
  End If
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 5.13: Close a document that is modified**

```javascript
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:

```java
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", 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](http://api.openoffice.org/docs/common/ref/com/sun/star/frame/FrameSearchFlag.html)

Table 5.1: Frame search flags.

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Auto</td>
<td>SELF+CHILDREN</td>
</tr>
<tr>
<td>1</td>
<td>PARENT</td>
<td>Includes the parent frame</td>
</tr>
<tr>
<td>2</td>
<td>SELF</td>
<td>Includes the start frame</td>
</tr>
<tr>
<td>4</td>
<td>CHILDREN</td>
<td>Include the child frames of the start frame</td>
</tr>
<tr>
<td>8</td>
<td>CREATE</td>
<td>Frame will be created if not found</td>
</tr>
<tr>
<td>16</td>
<td>SIBLINGS</td>
<td>Include the other child frames of the parent of the start frame</td>
</tr>
<tr>
<td>32</td>
<td>TASKS</td>
<td>Include all frames in all tasks in the current frames hierarchy</td>
</tr>
<tr>
<td>23</td>
<td>ALL</td>
<td>Include all frames not in other tasks. $23 = 1+2+4+16 = \text{PARENT} + \text{SELF} + \text{CHILDREN} + \text{SIBLINGS}$.</td>
</tr>
<tr>
<td>55</td>
<td>GLOBAL</td>
<td>Search entire hierarchy of frames. $55 = 1+2+4+16+32 = \text{PARENT} + \text{SELF} + \text{CHILDREN} + \text{SIBLINGS} + \text{TASKS}$.</td>
</tr>
<tr>
<td>63</td>
<td></td>
<td>GLOBAL + CREATE</td>
</tr>
</tbody>
</table>
**Listing 5.14: 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())

<table>
<thead>
<tr>
<th>Tip</th>
<th>In 1.1 the frame implements loadComponentFromURL so you can use:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>oDoc = oDesk.LoadComponentFromUrl(sUrl_1, &quot;_blank&quot;, 0, Noargs())</td>
</tr>
<tr>
<td></td>
<td>oFrame = oDoc.CurrentController.Frame</td>
</tr>
<tr>
<td></td>
<td>oDoc = oFrame.LoadComponentFromUrl(sUrl_2, &quot;&quot;, 2, Noargs())</td>
</tr>
<tr>
<td></td>
<td>Note the search flag arguments and the empty frame name argument.</td>
</tr>
</tbody>
</table>

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

**Listing 5.15: 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 5.16: 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 5.17: Load a document into an existing frame.**

```vba
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 5.18: Save a document using a password.

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

    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 5.19: Create a new document from a template.

```vba
Sub NewDoc
    Dim oDoc
    Dim sPath$ as string
    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 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 5.20: Examples setting media descriptor properties.

```vba
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 5.21: 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 desitnation 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())
    End If
  Loop
End Sub
5.9. Loading/Inserting a graphic 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 graphic is inserted as a link into the document. The following macro inserts a text graphic 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 graphic size, you do not need to set the position.

**Listing 5.22: Insert a GraphicsObject into a document as a link.**

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

    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
    FileNane=dir()
End Sub
```

```vba
Dim 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
Else
    oCursor.gotoEnd(false)
    oCursor.BreakType = com.sun.star.style.BreakType.PAGE_BEFORE
    oCursor.insertDocumentFromUrl(SrcFile, argsInsert())
End If
Else
    oCursor.gotoEnd(false)
    oCursor.BreakType = com.sun.star.style.BreakType.PAGE_BEFORE
    oCursor.insertDocumentFromUrl(SrcFile, argsInsert())
End If
Else
    oCursor.gotoEnd(false)
    oCursor.BreakType = com.sun.star.style.BreakType.PAGE_BEFORE
    oCursor.insertDocumentFromUrl(SrcFile, argsInsert())
End If
Else
    oCursor.gotoEnd(false)
    oCursor.BreakType = com.sun.star.style.BreakType.PAGE_BEFORE
    oCursor.insertDocumentFromUrl(SrcFile, argsInsert())
End If
Else
    If IsNull(oDoc) OR IsEmpty(oDoc) Then
        MsgBox "No documents merged!",16,"Merging Documents"
        Exit Sub
    Else
        Dim oGraph = oDoc.createInstance("com.sun.star.text.GraphicObject"
        oGraph.insertDocumentFromUrl(SrcFile, argsInsert())
    End If
    End Sub
```
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 5.23:** 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.getDrawablepage().add(oGraph)

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

**Tip** An inserted graphic can be contained in the document, or outside of the document. In either case, the GraphicURL always links to the graphic. If the object is not inserted as a link, then then URL stars with the text “vnd.sun.star.GraphObject:”. 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. Convert a linked graphic to an embedded graphic.

To insert an embedded graphic into a document, it must first be inserted as a link and then changed to an embedded object. Unfortunately, I only know how to do this using a drawing graphic, not a text graphic. This is unfortunate, because I have a strong preference for a text graphic in a Writer document so that I can anchor it as a character. The following macro was used while traversing the text content, to convert linked graphics into embedded graphics.

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

```vba
Sub EmbedLinkedGraphic(oGraph)
    REM Author: Andrew Pitonyak
    Dim sGraphURL As String
    Dim oGraph_2
    Dim oCurs
    Dim oText
    Dim oAnchor
    Dim s$
    If InStr(oGraph.GraphicURL, "vnd.sun") <> 0 Then
        REM Ignore an image that is already embedded
        Exit Sub
    End If
    s = "com.sun.star.drawing.GraphicsObjectShape"
    If oGraph.supportsService(s) Then
        REM I only know how to convert a GraphicObjectShape.
        REM I do not know how to convert a TextGraphicObject,
        REM but it is probably related to the ImageMap attribute.
        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)
    End If
End Sub
```

5.9.2. Danny Brewer embeds a graphic

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.**

```vba
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 an 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 5.22 or Listing 5.23 to insert a graphic that is stored inside of the
document. The following macro inserts the same graphic using two different methods. Either
will work, depending upon whether you desire an embedded or a linked graphic. 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"))

5.9.3. Embed a graphics using a dispatch

A dispatch can easily embed a graphic into a document.
**Listing 5.27: Embed a graphic into a document.**

```vba
Dim oFrame
Dim oDisp
Dim oProp(l) 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())
```

**5.9.4. Embed a graphics directly**

The following method, requires OOo version 2.0 or later. I do not have the time to explain how it works, but it is certainly useful.

**Listing 5.28: Embed a graphic into a document.**

```vba
Sub EmbedGraphic(oDoc, sURL$)
    REM Author: Stephan Wunderlich
    Dim oShape
    Dim oGraph 'The graphic object is text content.
    Dim oProvider 'GraphicProvider service.
    Dim oText
    Dim s$

    s = "com.sun.star.drawing.GraphicObjectShape"
oShape = oDoc.createInstance(s)
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

    oShape.Graphic = oProvider.queryGraphic(oProps())
oGraph.graphicurl = oShape.graphicurl
    oText = oDoc.getText()

    ' Insert at the current cursor location
    Dim oVC : oVC = oDoc.getCurrentController().getViewCursor()
oText.insertTextContent(oVC, oGraph, false)

    ' We no longer require the shape object.
```
5.9.5. Duplicate an existing graphic

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

**Listing 5.29: Duplicate an existing graphic.**

```vba
Sub DuplicateFirstGraphic()
    Dim oDoc, oExistingGraph, oNewGraph
    oDoc = ThisComponent
    oExistingGraph = oDoc.getGraphicObjects().getByName(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
```

5.10. Setting Margins

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

**Listing 5.30: Set the margins by modifying the text style.**

```vba
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 5.31: Set the page size using the Width and Height properties.**

```vba
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 5.32: Use the OleObjectFactory to start an application.**

```vba
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 5.33: 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 5.34: 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 5.35: 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
'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 5.36: 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$, Chaine$

52
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$(9) & CHRS(9) & chaine & CHR$(10)
oText.insertString(vTextCursor, Chaine, FALSE)
End If
Next
MsgBox "Finished"
End Sub

5.15. Return the Fibonacci array

Listing 5.37: 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, one of the greatest European 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 5.38: Insert text at a bookmark.

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

5.17. Saving And Exporting A Document

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

Listing 5.39: Save a document if it has not changed and it can be stored.

```vba
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 5.40: Save a document to a different location.

```vba
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())
```
The code shown so far, will not export the document to a different type. To do this, a particular export filter must be defined and any required properties must be set. You should know the name of the export filter and the file extension. There is a list of import and export filters at http://framework.openoffice.org/files/documents/25/897/filter_description.html and there is a lot of good information at http://oooconv.free.fr/engine/OOOconv.php as well. 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 5.41: 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 5.42: 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.getName
    args3(0).Value = "file:///c|/"&nom&".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

5.18. User Fields

I finally spent some time with user fields so although I do not really understand everything that there is to know, I can at least use them. Most people will choose to use the Master Fields. These fields allow you to set your own names and values.
5.18.1. Document Information

There are four fields with the names “Info 1”, “Info 2”, “Info 3”, and “Info 4”. I have essentially no use for them, but they exist and you can access them, so I mention them. You can give them your own name and your own values.

Listing 5.43: Display document information.

' Access the user fields for the information regarding the document.
vInfo = vDoc.getDocumentInfo()
vVal = oData.ElementNames
s = "===User Fields==="
For i = 0 to vInfo.GetUserFieldCount() - 1
    sKey = vInfo.GetUserFieldName(i)
    sVal = vinfo.GetUserFieldValue(i)
    s = s & Chr$(13) & "(" & sKey & "," & sVal & ")"
Next i
'(Info 1,)
'(Info 2,)
'(Info 3,)
'(Info 4,)
MsgBox s, 0, "User Fields"

The general document information such as author, creation date is also available from the document information object. In my macro book, I discuss how to access the properties direction using properties (see page 208 and 209).

Listing 5.44: Display more document information.

Sub InspectDocumentProperties
    REM Author: Andrew Pitonyak
    REM Certain types of properties can not be directly converted to
    REM a string so I use On Error Resume Next. As of July 27, 2005,
    REM in OOo 2.0 beta, all properties that fail are date structures
    REM so this is not really needed right now.
    On Error Resume Next

    Dim oPropValues()  ' Array of property values
    Dim i%           ' General index variable
    Dim s$           ' General string variable

    REM Get the property values from the document information object.
    REM This is an array of property values.
    oPropValues() = ThisComponent.getDocumentInfo().getPropertyValues()
    For i=LBound(oPropValues()) To UBound(oPropValues())
        s = s & oPropValues(i).Name
    Next i

    REM The property value is a struct so it is probably a Date
    REM structure.
    If IsUnoStruct(oPropValues(i).Value) Then

56
REM If the name contains "Date", then assume that it is
REM a date object.
If InStr(oPropValues(i).Name, "Date") > 0 Then
    With oPropValues(i).Value
        s = s & " : " & .Month & "/" & .Day & "/" & .Year & 
           " : " & .Hours & ":" & .Minutes & ":" & .Seconds & 
           "." & .HundredthSeconds
    End With
Else
    REM This is an unknown property value.
    s = s & ":??:"
End If
Else
    REM The property is not a struct. This line will
    REM fail if the object is not automatically converted
    REM to a string.
    s = s & ":" & oPropValues(i).Value
End If
s = s & CHR$(10)
Next
MsgBox s
End Sub

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 5.45: 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
        End Do
    End If
End Sub
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 5.46: Get a master field.

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

```Sub FieldExamples
    Dim vDoc, vInfo, vVal, vNames
    Dim iVar%, sKey$, sVal$, s$
    vDoc = ThisComponent
    Dim vTextFieldMaster
    Dim sUserType$
    sUserType = "com.sun.star.text.FieldMaster.User"

    vVal = vDoc.getTextFieldMasters()
    vNames = vVal.getElementNames()
    'You will have 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
```
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 5.48: Setting document variables.**

```vbnet
'setting document variables.

Function CountDocVars(ByVal DocVars As Variant, ByVal DocVarValue As Variant) As Integer
Dim VarCount As Integer
Dim Names As Variant
VarCount = 0
Names = thisComponent.getTextFieldMasters().getElementNames()
For iVar = LBound(Names) To UBound(Names)
    If (Len$(Names(iVar)) = 34) Then
        xMaster = ThisComponent.getTextFieldMasters.getByName(Names(iVar))
        DocVars(VarCount) = xMaster.Name
        DocVarValue(VarCount) = xMaster.Value
        VarCount = VarCount + 1
    End If
Next iVar
CountDocVars = VarCount
End Function

Function SetDocumentVariable(ByVal strVarName As String, ByVal aValue As String) As Boolean
On Error GoTo ErrorHandler
oActiveDocument = thisComponent
Dim bFound As Boolean
bFound = False
If Not IsNull(oActiveDocument) Then
    oActiveDocument = oActiveDocument
End If
bFound = True
SetDocumentVariable = bFound
ErrorHandler:
SetDocumentVariable = False
End Function
```

59
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.createElementInstance( 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 doc 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)  ' check if variable exists
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)  ' check if variable 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 As String, ByVal avalue As Variant) 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.
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 5.49: Use the code in Listing 5.48
REM Here is my code to call the routines above:
Sub Using_docVariables()
odoc = thisComponent
otext = odoc.getText()
ocursor = otext.createTextCursor()
ocursor.goToStart(false)
SetDocumentVariable("docVar1", "Value 1") 'create my DocVariable

61
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 5.50: 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. 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 5.51: Append a SetExpression text field to the end of the document.

Sub AddExpressionField
  Dim oField            ' This is the SestExpression 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 Now, 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 5.52: 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 5.53: 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", "PNEUMONOULTRAMICROSCOPICSILICOVOLCANOCONIOSIS", "Pitonyak", "misspell")

    '*********Spell Check Example!
    'http://api.openoffice.org/docs/common/ref/com/sun/star/linguistic2/XSpellChecker.html
    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!
    'http://api.openoffice.org/docs/common/ref/com/sun/star/linguistic2/XHyphenator.html
    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
            MsgBox "Hyphenating " & s(i) & " returns null"
        Else
            MsgBox "Hyphenating " & s(i) & " returns " & vReturn.getPossibleHyphens()
        End If
    Next

    '******Thesaurus Example!
    'http://api.openoffice.org/docs/common/ref/com/sun/star/linguistic2/XThesaurus.html
    Dim vThesaurus As Variant
    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
            Dim vSyns As Variant
            vSyns = vReturn(j).querySynonyms()
            For k = LBound(vSyns) To UBound(vSyns)
            Next
        End If
    Next
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 5.54: You can not change the mouse cursor as of OOo version 1.1.3.

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 5.55: Switch the mouse pointer for a control.

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 5.56: 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 5.57: 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, oct1
   Dim sUrl As String
   Dim NoArg()

   REM Set source doc/currentController/frame/sheet/range.
   oSourceDoc=ThisComponent
   oct1 = oSourceDoc.getCurrentController()
   oSourceframe = oct1.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:
for more details. The following code was posted on the devapi mailing list.

Listing 5.58: 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 rang to copy to.
oRangeOrg = oSheet1.getCellRangeByName("A1:C10").RangeAddress
oRangeCpy = oSheet2.getCellRangeByName("A1:C10").RangeAddress

REM
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 5.59: 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 "Implemantation 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 Sub
5.23.4. Storing a string to the clipboard

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

**Listing 5.60: 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.setContent(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 5.61 demonstrates how to access the clipboard as text. Too busy to explain how this code works.
Listing 5.61: 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 "Implemantation name = " & oClip.getImplementationName()
    oClipContents = oClip.getContents()
    oTypes = oClipContents.getTransferDataFlavors()

    Dim msg$, iLoc%, outS
    msg = ""
    iLoc = -1
    For i = LBound(oTypes) To UBound(oTypes)
        If oTypes(i).MimeType = "text/plain;charset=utf-16" Then
            iPlainLoc = i
            Exit For
        End If
        msg = msg & "Mime type = " & x(ii).MimeType & " normal = " & x(ii).HumanPresentableName & Chr$(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 5.62: 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())
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 5.63: 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 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 5.63 iterates through the document using a paragraph cursor. I opted to not use a paragraph cursor in Listing 5.64, 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 5.64: Set the locale for selected text (or the document).**

```vba
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
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
```

72
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()
        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 oSel.getCount() > 1 Then
    IsAnythingSelected = True
Else
    oSel = oSel.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/AutoTextContainer.html
http://api.openoffice.org/docs/common/ref/com/sun/star/text/XAutoTextContainer.html

Listing 5.65: 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.getCount())
oListGroupArrivee.removeItem(0,oListGroupArrivee.getCount())
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 oListGroup as object
    Dim oListElement as object
    Dim i as integer
    Dim aTableau() as variant

    oListGroup = oDialog.getControl(ListGroupe)
    oListElement = oDialog.getControl(ListElement)
    oGroupe = oAutoText.getByIndex(oListGroup.getSelectedItemPos())
aTableau = oGroupe.getTitles()
oListElement.removeItem(0,oListElement.getCount())
For i = LBound(aTableau) To UBound(aTableau)
oListGroup.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(oListGroupArrivee.getSelectedItemPos())
If oGroupArrivee.HasByName(aTableau(oListElement.getSelectedItemPos())) Then
  MsgBox ("Cet élément existe déjà")
  Exit Sub
End If
oGroupArrivee.insertNewByName(aTableau(oListElement.getSelectedItem()), oListElement.getSelectedItem(), oElement.Text)
ChargerListeGroupe("ListBox2","ListBox4")
End Sub

5.27. Decimal Feet To Fraction

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 5.66: 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 5.67: 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

    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
    End If
    x = y
End Sub
To test this routine, I created the following test code.

```vbscript
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 5.68:** Convert decimal feet to a string.

REM [-]feet'-inches n/d"
REM No part is returned if it is zero.
Function DecimalFeetToString64(ByVal x#) As String
    'I only use 64, because this is what it was in the original
    DecimalFeetToString64 = DecimalFeetToString(x, 64)
End Function

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, max_denominator)
    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 = 0Then
        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

DecimalFeetToString = s
End Function

Function StringToDecimalFeet(s$) As Double
REM Maximum number of tokens would include
REM <feet><'><-><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#
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

num_tokens = 0 : i = 1
Do While i <= Len(s)
  Select Case Mid(s, i, 1)
    Case "+", "-", "/" To "9"
      j = i
      If Left(s, i, 1) = "-" Then j = j + 1
      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
      tokens(num_tokens) = Mid(s, i, j-i)
      num_tokens = num_tokens + 1
      i = j
    Case "'"
      feet = CDbl(tokens(num_tokens-1))
      tokens(num_tokens) = """
      num_tokens = num_tokens + 1
      i = i + 1
      If (i <= Len(s)) Then
        If x <> 0 Then
          s = s & x
          If numerator <> 0 Then s = s & " "
        End If
        If numerator <> 0 Then s = s & numerator & "/" & denominator
    Case Else
      i = i + 1
  End Select
  j = i
  i = i + 1
Do While (j <= Len(s))
  c = Asc(Mid(s, j, 1))
  If 48 <= c AND c <= 57 Then
    j = j + 1
    If j > Len(s) Then Exit Do
  End If
  c = Asc(Mid(s, j, 1))
  Loop
  If (j <= Len(s)) Then
If Mid(s, i, 1) = "-" Then i = i + 1
End If
Case "**, "/", " "
tokens(num_tokens) = Mid(s, i, 1)
i = i + 1
Do While i < Len(s)
    If Mid(s, i, 1) <> tokens(num_tokens) Then Exit Do
    i = i + 1
Loop
If tokens(num_tokens) = "/" Then
    n = CDbl(tokens(num_tokens-1))
    num_tokens = num_tokens + 1
ElseIf tokens(num_tokens) = " " Then
    Inches = CDbl(tokens(num_tokens-1))
ElseIf tokens(num_tokens) = "***" Then
    If num_tokens = 1 Then
        Inches = CDbl(tokens(num_tokens-1))
    ElseIf num_tokens > 1 Then
        If tokens(num_tokens-2) = "/" Then
            d = CDbl(tokens(num_tokens-1))
        Else
            Inches = CDbl(tokens(num_tokens-1))
        End If
    End If
End If
End If
Case Else
    'Hmm, this is an error
    i = i + 1
    Print "In the else"
End Select
Loop
If d = 0 Then d = 1
StringToDecimalFeet = leadingNeg * (feet + (inches + n/d)/12)
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 5.69: 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
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 5.70: Convert big numbers into words**

```vbnet
Function NumberToText(ByVal n) As String
    REM by Andrew D. Pitonyak
    Dim sBigWordsUSA() = Array("zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "ten", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen", "twenty")
    Dim sBigWordsUK() = Array("zero", "Ten", "twenty", "thirty", "fourty", "fifty", "sixty", "seventy", "eighty", "ninety")

    s = ""
    If n > 99 Then
        s = sBigWordsUSA(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 & sBigWordsUSA(Fix(n / 10))
        n = n MOD 10
        If n = 0 Then
            SmallIntToText = s
            Exit Function
        End If
        s = s & " 
    End If

    SmallIntToText = s & sBigWordsUSA(n)
End Function
```
sBigWordsUSA = Array( "", _
  "thousand", "million", "billion", "trillion", "quadrillion", _
  "quintillion", "sextillion", "septillion", _
  "octillion", "nonillion", "decillion", "undecillion", "duodecillion", _
  "tredecillion", "quattuordecillion", "quindecillion", "sexdecillion", _
  "septdecillion", "octodecillion", "novemdecillion", "vigintillion", _
  "unvigintillion", "duovigintillion", "trigintillion", _
  "quattuorvigintillion", "quingintillion", "sexagintillion", _
  "septvigintillion", "octovigintillion", "novemvigintillion", _
  "trigintillion", "untrigintillion", "duotrigintillion", _
  "tretrigintillion", "quattuortrigintillion", "quintrigintillion", _
  "sextrigintillion", "septtrigintillion", "octotrigintillion", _
  "quattuortrigintillion", "quinterigintillion", "sextrigintillion", _
  "septtrigintillion", "octotrigintillion", "novemtrigintillion", _
  "quattuorvigintillion", "quintoctogintillion", "sextinvigintillion", _
  "septoctogintillion", "octoquingintillion", "unquingintillion", _
  "centillion" )

sBigWordsUK = Array( "", _
  "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", _
  "unvigintillion", "duovigintillion", "trigintillion", _
  "quattuorvigintillion", "quingintillion", "sexagintillion", _
  "septvigintillion", "octovigintillion", "novemvigintillion", _
  "trigintillion", "untrigintillion", "duotrigintillion", _
  "tretrigintillion", "quattuortrigintillion", "quintrigintillion", _
  "sextrigintillion", "septtrigintillion", "octotrigintillion", _
  "quattuorvigintillion", "quintoctogintillion", "sextinvigintillion", _
  "septoctogintillion", "octoquingintillion", "unquingintillion", _
  "centillion" )

82
"octodeciilliard", "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", _
sextrigliard", "sextrigintilliard", "septentrigintillion", _
"septentrigintilliard", "octotrigintillion", "octotrigintilliard", _
noveotrigintillion", "noveotrigintilliard", "quadragintillion", _
quadrupligintillion", "quadrupligintilliard", "unquadragintillion", _
"unquadragintilliard", "duoquadragintillion", "duoquadragintilliard", _
trequadragintillion", "trequadragintilliard", "quattuorquadragintillion", _
quattuorquadragintilliard", "quingquadragintillion", _
quingquadragintilliard", "sexquadragintillion", "sexquadragintilliard", _
"septenquadragintillion", "septenquadragintilliard", _
octoquadragintillion", "octoquadragintilliard", _
"novemquadragintillion", "novemquadragintilliard", "quinquagintillion", _
"quinquagintilliard" _

sBigWordsDE() = Array("", "Tausand", _
"Quadrillion", "Quadrilliarde", "Quintillion", "Quintillarde", _
"Oktillion", "Oktilliarde", "Nonillion", "Nonilliarde", _
"Dezillion", "Dezilliarde", "Undezillion", "Undezilliarde", _
"Duodezillion", "Doudezilliarde", "Tredeziillion", _
"Trediilliarde", "Quattuordezillion", "Quattuordezilliarde", _
"Quindezillion", "Quindezilliarde", "Sexdezillion", _
"Sexdezilliarde", "Septendesilion", "Septendesillarde", _
"Oktodezillion", "Oktodezilliarde", "Novemdesillion", _
"Novemdesilliarde", "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", _
"Tretrigintillion", "Tretrigintilliarde", "Quattuortrigintillion", _
"Quattuortrigintilliarde", "Quintrigintillion", "Quintrigintilliarde", _
"Sextrigliard", "Sextrigliarde", "Septentrigintillion", _
"Septentrigintilliarde", "Oktotrigintillion", "Oktotrigintilliarde", _

83
"Novemtrigintillion", "Novemtrigintilliarde", "Quadragintillion", __
"Quadragintilliarde", "Unquadragintillion", "Unquadragintilliarde", __
"Duooquadragintillion", "Duooquadragintilliarde", "Trequadragintillion", __
"Trequadragintilliarde", "Quattuorquadragintillion", __
"Quattuorquadragintilliarde", "Quinquadragintillion", __
"Quinquadragintilliarde", "Sexquadragintillion", __
"Sexquadragintilliarde", "Septenquadragintillion", __
"Septenquadragintilliarde", "Octoquadragintillion", __
"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 5.71: 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 for 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 5.72: 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()
'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.createsimpleEmailMessage()
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

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 5.73: Send email using a URL.

Dim noargs()
email_dispatch_url = "mailto:demo@someplace.com?subject=Test&Body=Text"
dispatcher = createUnoService( "com.sun.star.frameDispatchHelper")
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 5.74: 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")

86
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. Russ Phillips is the author of a very good "Macro Library How-To" available from [http://www.ooomacros.org](http://www.ooomacros.org). The how-to document discusses creating and using libraries from the OOO GUI.

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 if 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 cannot 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 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 5.75: Using the ApplicationScriptLibraryContainer.**

```plaintext
Sub LibContainer()
    REM Christian Junker
    Dim allLibs()
```

88
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 5.76: Deploy a macro using the ApplicationScriptLibraryContainer:

```vbnet
'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 5.77: 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/simpress"
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 5.78: 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 5.79: 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 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 5.80: 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. 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 5.81: 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

5.30.3. 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 5.82: 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
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 5.83: 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
    MyInpuStream=ThePackageStream.GetInputStream()
    'Define the outputfile
    oFile = createUnoService("com.sun.star.ucb.SimpleFileAccess")
    oFile.WriteFile(ConvertToURL(DestFile),MyInpuStream)
    'Its Done !!!
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 5.84: Extract all files in a zip file.

```vba
' 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 all files in a zip file. 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.
    ' strDestinationFolder = full path (directory and filename) where the source
    '   folder 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(strDestinationFolder) Then Exit Sub

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

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

96
Unzip the an entire .zip archive to a destination directory.

strZipArchivePath = full path (directory and filename) to the .zip archive file.
strDestinationFilePath = folder (directory only) 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 5.85: 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 mimetype file of zero length.
Sub ZipDirectory(sSrcDir As String, sZipName As String)
    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%        ' Each directory component in an array.
    Dim repZip
    Dim chaînes$  ' Each directory component in an array.
    Dim decoupe  ' Each directory component in an array.
    Dim sBaseDir$
End Sub

97
Dim RepPere
Dim RepPereZip
Dim sFileName$  ' File stream
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 RemplaceChaîne(ByVal sSearchThis$, sFindThis$, dest$, bCase As Boolean)
' Auteurs: Laurent Godard & Bernard Marcey
' fournit une sSearchThis dont toutes les occurrences 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, remove 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
RemplaceChaîne = 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 5.86: 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 5.87: Run a macro in a document based on the value in a string.

Sub RunDocumentNamedMacro
    Dim oDisp
    Dim sMacroURL As String
    Dim sModuleName As String
    Dim sModuleLocation 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.
    sModuleName = "vnd.sun.star.script:Standard.Module1.MainExternal"
    sModuleLocation = "?language=Basic&location=document"
    sMacroURL = sModuleName & sModuleLocation

    oDisp.executeDispatch(oFrame.CurrentControllerWindow, sMacroURL, ",", 0, Array())
End Sub
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())
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.oooforum.org/forum/viewtopic.php?t=6657)

The magic is performed by the SystemShellExecute service, which contains one method; execute!

Listing 5.88: 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 5.89: List fonts.
Sub ListFonts
Dim oWindow
Dim oDescript
Dim s$
Dim i%

'The container window supports the awt XDevice interface.
'Array of awt FontDescriptor structures
'Temporary string variable to hold all of the string names.
'General index variable

oWindow = ThisComponent.getCurrentController().getFrame().getContainerWindow()
oDescript = 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 5.90: 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).
102


### Table 5.2. Properties supported by the com.sun.star.util.PathSettings service.

<table>
<thead>
<tr>
<th><strong>Property</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Backup</td>
<td>Automatic backup copies of documents are stored here.</td>
</tr>
<tr>
<td>Basic</td>
<td>The Basic files, used by the AutoPilots, can be found here. The value can be more than one path separated by a semicolon.</td>
</tr>
<tr>
<td>Favorite</td>
<td>Path to save folder bookmarks</td>
</tr>
<tr>
<td>Gallery</td>
<td>Location of the Gallery database and multimedia files. The value can be more than one path separated by a semicolon.</td>
</tr>
<tr>
<td>Graphic</td>
<td>This directory is displayed when the dialog for opening a graphic or for saving a new graphic is called.</td>
</tr>
<tr>
<td>Help</td>
<td>The path to the Office help files.</td>
</tr>
<tr>
<td>Module</td>
<td>This is the path for the modules.</td>
</tr>
<tr>
<td>Storage</td>
<td>Mail, News files and other information (for example, about FTP Server) are stored here.</td>
</tr>
<tr>
<td>Temp</td>
<td>The base URL to the office temp-files</td>
</tr>
<tr>
<td>Template</td>
<td>The templates originate from these folders and sub-folders. The value can be more than one path separated by a semicolon.</td>
</tr>
<tr>
<td>UserConfig</td>
<td>Folder that contains the user settings.</td>
</tr>
<tr>
<td>Work</td>
<td>User's work folder, which can be modified – used by the Open and Save dialogs.</td>
</tr>
</tbody>
</table>

The following code clarifies how this works:

**Listing 5.91: Use the PathSettings service.**

```vba
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 com.sun.star.util.PathSubstitution service.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(inst)</td>
<td>Installation path of the Office.</td>
</tr>
<tr>
<td>$(prog)</td>
<td>Program path of the Office.</td>
</tr>
<tr>
<td>$(user)</td>
<td>The user installation directory.</td>
</tr>
<tr>
<td>$(work)</td>
<td>The user's work directory of the user. Under Windows this is the &quot;MyDocuments&quot; subdirectory. Under Unix this is the home-directory.</td>
</tr>
<tr>
<td>$(home)</td>
<td>The user's home directory of the user. Under Unix this is the home-directory. Under Windows this is the &quot;Documents and Settings &quot; subdirectory.</td>
</tr>
<tr>
<td>$(temp)</td>
<td>The current temporary directory.</td>
</tr>
<tr>
<td>$(path)</td>
<td>The value of PATH environment variable.</td>
</tr>
<tr>
<td>$(lang)</td>
<td>The country code used by the Office, like 01=english, 49=german.</td>
</tr>
<tr>
<td>$(langid)</td>
<td>The language code used by the Office, like 0x0009=english, 0x0409=english us.</td>
</tr>
<tr>
<td>$(vlang)</td>
<td>The language used by the Office as a string. Like &quot;german&quot; for a German Office.</td>
</tr>
</tbody>
</table>

Listing 5.92: Use the PathSubstitution service.

```java
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 5.93: SimpleFileAccess allows you to set the output format.

```java
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](http://www.oooforum.org/forum/viewtopic.php?t=4907)). You should read this before using the following macro:
Listing 5.94: 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 occured.'}

{' 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 As String, oAttributes 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
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
gLocatorSet = True
End Sub

DannyB recommends starting with a small file for your initial tests:

<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 5.95: Adding two days together is easy.

Function addDays(StartDate as Date, nDays As Integer) As Date
  REM To add days, simply add them in
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 Van Buggenhaut, 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 5.96: Add years, months, and days to a date.

```vba
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
End Function
```
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 5.97: 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 5.98: Determine a document's type.*

```basic
Sub DetectDocType()
    Dim oMediaDescr(30) As new com.sun.star.beans.PropertyValue
    Dim ss$ : ss$ = "com.sun.star.document.TypeDetection"
    Dim oTypeManager

    oMediaDescr(o).Name = "URL"
    oMediaDescr(o).Value = ThisComponent.getURL()

    oTypeManager = createUnoService(ss$)
    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 5.99: Determine a document's type.*

```basic
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/ooood/

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 5.100.

Listing 5.100: 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", __
)
Use a frame's LayoutManager to find the current toolbars. It may be a but the Listing 5.101 displays toolbars, menus, and status bars.

**Listing 5.101: Display toolbars in the current document.**

```vba
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.TOOBAR

    oFrame = oDoc.getCurrentController().getFrame()
    oCfgManager = oDoc.getUIConfigurationManager()
    oToolInfo = oCfgManager.getUIElementsInfo( iToolType )
    For Each x in oFrameLayoutManager.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 method checks all element types, not just toolbars.

**Listing 5.102: See if a specified toolbar is visible.**

```vba
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 = oFrameLayoutManager
    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 5.103: 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 5.104:** Create a simple toolbar item.

Rem A com.sun.star.ui.ItemDescriptor is an array of property values. This example does not set all supported values, such as "Style", which uses values from com.sun.star.ui.ItemStyle. For menu items, the "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, 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 5.105:** Add a simple toolbar to the Basic IDE.

Sub CreateBasicIDEToolbar
  Dim sToolbarURL$ as String ' URL of the custom toolbar.
  Dim sCmdID$ as String ' Command for a single toolbar button.
  Dim sCmdLabel as String ' Label for a single toolbar button.
  Dim sDocType$ as String ' Component type that will contain the toolbar.
  Dim sSupplier$ as String ' ModuleUIConfigurationManagerSupplier
  Dim oSupplier as Object
  Dim oModuleCfgMgr as Object ' Module manager.
  Dim oTBSettings as Object ' Settings that comprise the toolbar.
  Dim oToolBarItem as Object ' Single toolbar button.
  Dim nCount% as Integer

  REM Name of the custom toolbar; must start with "custom_".
sToolbarURL = "private:resource/toolbar/custom_test"

  REM Retrieve the module configuration manager from the central module configuration manager supplier
  sSupplier = "com.sun.star.ui.ModuleUIConfigurationManagerSupplier"

  CreateBasicIDEToolbar
End Sub
oSupplier = CreateUnoService(sSupplier)

REM Specify the document type associated with this toolbar.
REM sDocType = "com.sun.star.text.TextDocument"
REM 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.getService( 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
REM oTBSettings.UIName = "My little custom toolbar"
REM *** Create a button for our new custom toolbar
REM sCmdID = "macro:///PitonyakUtil.UI.TBTest()"
REM sCmdLable = "Test"
REM nCount = 0
REM 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)
REM If ( oModuleCfgMgr.hasSettings( sToolbarURL ) ) Then
REM  oModuleCfgMgr.replaceSettings( sToolbarURL, oTBSettings )
REM Else
REM  oModuleCfgMgr.insertSettings( sToolbarURL, oTBSettings )
REM End If
End Sub

Carsten Driesner provided a macro that adds a button to the standard toolbar in a Writer document.

**Listing 5.106: 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 central module configuration 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

    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.
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.

```vbnet
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.

```vbnet
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:

```vbnet
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()
```
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

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 nSheets As Long
    Dim oCellRange As Object

    oDoc = ThisComponent
    oSheets = oDoc.Sheets
    nSheets = oDoc.Sheets.Count
    REM Get the third sheet, as in 0, 1, 2
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.

```vba
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
```

```vba
End Sub
```
Print "Multiple Cell Ranges selected = " & oSels.getImplementationName()
Print "Count = " & oSels.getCount()
Else
Print "Something 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() As Double
    Dim oRow() As Double
    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.**

```vba
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.**

```vba
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

After selecting a cell, the entire cell is selected – the entire cell is black. This is easy

Listing 6.13: Select a single cell so that it is black.

Dim oCell
Dim oSheet

REM Get the first sheet.
oSheet = ThisComponent.getSheets().getIndex(0)
REM Get cell A2
oCell = oSheet.getCellByPosition(0, 1)
REM Move the selection to cell A2
ThisComponent.CurrentController.Select(oCell)

If you do want the entire cell to be dark, but rather selected with the dark outline around it, select an empty range AFTER moving the cursor to the desired cell.

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

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

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 $
'Save this so I do NOT modify the parameter.
'This was an icky bug that took me a while to find

125
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 sell 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.hhmms", 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
'******************************************************************
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 Analize
    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.getActiveCell
```


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
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()
End Sub
6.11.1. Set selected cells to a database range

Listing 6.23: Set selected cells to a database range.

'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
End Sub

6.11.2. Delete database range

Listing 6.24: Delete a database range.

'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.BorderLine
lHor.LineDistance = 10
Dim 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.**

```vba
'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.

```vba
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.AUTO
    '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
    aSortFields(1).FieldType = com.sun.star.util.SortFieldType.NUMERIC
    aSortFields(1).Field = 2
    aSortFields(1).SortAscending = TRUE

    aSortDesc(0).Name = "SortFields"
    aSortDesc(0).Value = aSortFields()      ' aSortFields(0)
```

132
To specify the first row as a header row, use another property. Be certain that you dimension enough properties.

```vba
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.

```vba
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@aurelius-mfg.com%](mailto:ryan@aurelius-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
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
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 document.

Sub CopySpreadsheet
 firstDoc = ThisComponent
 selectSheetByName(firstDoc, "Sheet2")
dispatchURL(firstDoc,".uno:SelectAll")
dispatchURL(firstDoc,".uno:Copy")
secondDoc = StarDesktop.loadComponentFromUrl("private:factory/scalc","_blank",0,dimArray())
secondDoc.getSheets().insertNewByName("inserted",0)
selectSheetByName(secondDoc,"inserted")
dispatchURL(secondDoc,".uno:Paste")
End Sub

Sub selectSheetByName(document, sheetName)
 document.getCurrentController.select(document.getSheets().getByName(sheetName))
End Sub

Sub dispatchURL(document, aURL)
 Dim noProps()
 Dim URL As New com.sun.star.util.URL
 frame = document.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.

```vba
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
identifying the upper left corner of the named range. 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.getByName(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:

<table>
<thead>
<tr>
<th>customer1</th>
<th>address1a</th>
<th>address1b</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer2</td>
<td>address2a</td>
<td>address2b</td>
</tr>
</tbody>
</table>

The macro shown below converts these records into a row format so that they can be used to
merge data into a form letter.

<table>
<thead>
<tr>
<th>Customer</th>
<th>address1</th>
<th>address2</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer1</td>
<td>address1a</td>
<td>address1b</td>
</tr>
<tr>
<td>customer2</td>
<td>address2a</td>
<td>address2b</td>
</tr>
</tbody>
</table>

[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
    int_col = 0
End Sub
int_row = 0
osheet = ThisComponent.Sheets.getByIndex(0)
ThisComponent.CurrentController.Select(osheet.GetCellByPosition(0,0))
Cellstring = ThisComponent.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
    ThisComponent.CurrentController.Select(osheet.GetCellByPosition(loop_col,loop_row))
    cellstring = ThisComponent.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:

```vba
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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculate()</td>
<td>Recalculate all dirty cells.</td>
</tr>
<tr>
<td>calculate all()</td>
<td>Recalculate all cells.</td>
</tr>
<tr>
<td>isAutomaticCalculationEnabled()</td>
<td>Return True if automatic calculation is enabled.</td>
</tr>
<tr>
<td>enableAutomaticCalculation(boolean)</td>
<td>Enable or disable automatic calculation.</td>
</tr>
</tbody>
</table>

The following code disables and then enables automatic calculation:

```vba
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. Gerrit Jasper provided the following example (which he did not write):

```vba
Sub testEndColRow
    Dim oSheet
    Dim oCell
    Dim nEndCol As Integer
    Dim nEndRow As Integer
    oSheet = ThisComponent.Sheets.getByIndex(0)
    nEndCol = getLastUsedColumn(oSheet) 'see functions below
    nEndRow = getLastUsedRow(oSheet)
    REM Then do as you please, e.g.
    oCell = oSheet.GetCellByPosition(nEndCol + 1, nEndRow + 1)
    oCell.String = "test"
    ThisComponent.CurrentController.Select(oCell)
End Sub

Function getLastUsedColumn(oSheet as Object) as Integer
    Dim oCell As Object
    Dim oCursor As Object
    Dim aAddress As Variant
    oCell = oSheet.GetCellByPosition(0, 0)
    oCursor = oSheet.createCursorByRange(oCell)
    oCursor.GotoEndOfUsedArea(True)
    aAddress = oCursor.RangeAddress
    GetLastUsedColumn = aAddress.EndColumn
End Function

Function getLastUsedRow(oSheet as Object) as Integer
    Dim oCell As Object
    Dim oCursor As Object
    Dim aAddress As Variant
    oCell = oSheet.GetCellByPosition(0, 0)
    oCursor = oSheet.createCursorByRange(oCell)
    oCursor.GotoEndOfUsedArea(True)
    aAddress = oCursor.RangeAddress
    GetLastUsedRow = aAddress.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:

```vba
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.html
    With oDescriptor
        .SearchString = sString$
        .SearchString = sString$
    End With
    REM These all default to false
REM SearchWords forces the entire cell to contain only the search string
REM SearchWords = bWholeWord
REM 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 notice 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
Dim oSheet
Dim oRange
Dim oFoundCell
oSheet = ThisComponent.getSheets().getByIndex(0)
oRange = oSheet.getCellRangeByName("F7:H11")
oFoundCell = SimpleSheetSearch("41", oRange, False)
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 the 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
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 call 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 user 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.

```vba
Function EvalArgs(Optional x As Variant) 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(1 To 3, 1 To 2)
        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 m
        Next n
    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.

```vba
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 m
    Next n
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.

<table>
<thead>
<tr>
<th></th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>6</td>
<td>9</td>
<td>12</td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>24</td>
<td>27</td>
<td>30</td>
<td>33</td>
</tr>
<tr>
<td>10</td>
<td>36</td>
<td>39</td>
<td>42</td>
<td>45</td>
<td>48</td>
</tr>
<tr>
<td>11</td>
<td>51</td>
<td>54</td>
<td>57</td>
<td>60</td>
<td>63</td>
</tr>
<tr>
<td>12</td>
<td>66</td>
<td>69</td>
<td>72</td>
<td>75</td>
<td>78</td>
</tr>
</tbody>
</table>

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.
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?**

```vba
Sub IsACellSelected
    Dim oSels As Object 'All of the selections
    Dim oSel As Object 'A single selection
    Dim i As Integer
    Dim 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:


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.

```java
oAnchor = oTable.getAnchor()
oCurs.gotoRange(oAnchor.getStart(), False) REM Error
oCurs = oAnchor.getText().createTextCursorByRange(oAnchor.getStart()) REM Error
oText.insertTextContent(oAnchor.getEnd(), oTable2, False) REM 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.

```java
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.

```vba
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 crursor 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.

```vba
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 cusor.

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.getName("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.*

```vba
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.

```vba
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 documentations states that if there is no current controller, then getCurrentSelection() will return a null rather than the selections. I have a limited understanding of this but I will take them at their word and check for this.
If the selection count is zero, then nothing is selected. I have never seen a selection count of zero, but I check for it anyway. If no text is selected, I have one selection of zero length. I have seen examples where a zero length selection is determined as follows:

**Listing 7.9:** This is a problem if more than 64K of text is selected.

If \( \text{Len}(\text{oSel}.\text{getString}()) = 0 \) Then nothing is selected

The problem with this is that it is possible that selected text will contain more than 64K characters and a string can not contain more than 64K characters. I consider this not safe. 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:** Oops, I used “oDoc.Text” rather than “oSel.Text”.

\[
\text{oCursor} = \text{oDoc.Text.CreateTextCursorByRange(oSel)}
\]
\[
\text{If oCursor.IsCollapsed() Then nothing is selected}
\]

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 the cursor in a text table?

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 certainly
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)
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.**

```vba
'******************************************************************************
'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
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.

```vba
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 Is AnythingSelected(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
```
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.

```vba
'******************************************************************
'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 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
    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(oL as Object, oR as Object)
    Dim oText
    oText = oL.getText()
    If IsNull(oL) Or IsNull(oR) Or IsNull(oText) Then Exit Sub
    If oText.compareRegionEnds(oL, oR) <= 0 Then Exit Sub
    oL.goRight(0, False)
    Do While oL.goRight(1, True) AND oText.compareRegionEnds(oL, oR) >= 0
        Print "Character = '" & oL.getString() & "'
        REM This will cause the currently selected text to become
        REM no longer selected
        oL.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 most likely fail if there are any tables and such because the
REM Sentence 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.gotNextParagraph(True)
        'At this point, you have the entire paragraph highlighted
        oSentenceCursor.gotoRange(oCursor startPos(), False)
        Do While oSentenceCursor.gotNextSentence(True) AND_
            oText.compareRegionEnds(oSentenceCursor, oCursor) >= 0
            i = i + 1
        Loop
        oCursor.goRight(0, False)
    Loop
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      'The current char is not white space, ignore it
  RankChar = 0 'so delete the current character.
ElseIf iPrevChar = 0 Then              'Start of a line and current char is white space
  RankChar = 1 'so delete the current character.
ElseIf Not IsWhiteSpace(iPrevChar) Then 'Current char is white space but previous is not
  RankChar = 0 'so delete the previous character.
ElseIf iPrevChar = 13 Then             'Previous char is highest ranked white space
  RankChar = 1 'so delete the current character.
ElseIf iCurChar = 13 Then              'Current character is highest ranked white space
  RankChar = -1 'so delete the previous character.
ElseIf iPrevChar = 10 Then             'No new Pars so see if previous char is new line
  RankChar = 1 'so delete the current character.
ElseIf iCurChar = 10 Then              'No new Pars so see if the current char is new line
  RankChar = -1 'so delete the previous character.
ElseIf iPrevChar = 9 Then              'No new Lines so see if the previous char is tab
  RankChar = 1 'so delete the current character.
ElseIf iCurChar = 9 Then               'No new Lines so see if the current char is a tab
  RankChar = -1 'so delete the previous character.
ElseIf iPrevChar = 160 Then            'No Tabs so check previous char for a hard space
  RankChar = 1 'so delete the current character.
ElseIf iCurChar = 160 Then             'No Tabs so check current char for a hard space
  RankChar = -1 'so delete the previous character.
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 it will only be removed from the selected region.
Sub RemoveEmptySpace
    Dim oCurs(), i%
    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, iTThisChar As Integer, iRank As Integer
    iThisChar = 0
    iThisChar = 0
    oLCurs.goRight(0, False)
    Do While oLCurs.goRight(0, True)
        iTThisChar = 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("")
            Exit Do
        End If
    'If went past the end then get out
    If i < 0 Then Exit Do
    iRank = RankChar(iLastChar, iTThisChar)
    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 " + iTThisChar
    End If
    iThisChar = iTThisChar + 1
    iLastChar = iThisChar
    oLCurs.goRight(1, False)
    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, iTThisChar As Integer, iRank As Integer
    iThisChar = 0
    iThisChar = 0
    oLCurs.goRight(0, False)
    Do While oLCurs.goRight(0, True)
        iTThisChar = 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("")
            Exit Do
        End If
    'If went past the end then get out
    If i < 0 Then Exit Do
    iRank = RankChar(iLastChar, iTThisChar)
    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 " + iTThisChar
    End If
    iThisChar = iTThisChar + 1
    iLastChar = iThisChar
    oLCurs.goRight(1, False)
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
End 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.goRight(0, False)
iLastChar = iThisChar
        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.

```vba
Function ADPWordCountStrings(vDoc) As String
    REM Place what ever characters you want as word separators here!
    Dim sSeps$ = Chr$(9) & Chr$(13) & Chr$(10) & " ,;."
    Dim bSeps(256) As Boolean, i As Long
    For i = LBound(bSeps()) To UBound(bSeps())
        bSeps(i) = False
    Next
    For i = 1 To Len(sSeps)
        bSeps(Asc(Mid(sSeps, i, 1))) = True
    Next
    Dim nSelChars As Long, nSelwords As Long, nSel%, nNonEmptySel%, j As Long, s$
    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
        oLCurEnd Sub
```
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) & Chr(13) & "Paragraphs-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
  End If
  lNumWords = lNumWords + 1
```

166
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
        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
        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. 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
7.3.8.5. 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.

```vba
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 cursor count routine 
  ' not unendurably slow with lots of footnotes, using cursors.
  ' acb, June 2003
  ' rewritten version of the
  ' dwwc 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, fnotes, 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 fnotes=0 To fnotes.getCount()-1
      thisnote=fnotes.getByIndex(fnotes)
      startcursor=thisnote.getStart()
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
      nSelwords+=startcursor.getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      sRes=nSelChars
      sRes+=startcursor.getCount() / 4
      sRes+=startcursor getCount() / 4
      thisrange.addCharacterRange(startcursor, startcursor.getCount())
      stopcursor=startcursor
      nSelChars+=startcursor.getCount()+1
```
stopcursor=thisnote.getEnd()
Do While thisnote.getText().compareRegionStarts(startcursor,stopcursor) AND _
startcursor.gotoNextWord(FALSE)
  fnotecount=fnotecount+1
Loop
msgbox(startcursor.getString())
  fnotecount=fnotecount+stringcount(thisnote.getString())
  fnotecount=fnotecount+CursorCount(thisnote,bigtext)
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: " +AllChars +" 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)
}169
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.

```vba
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
                    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
                    lIndex = lIndex + 1
                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.

```vba
Sub SelectNewParagraphsToNewLines
    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("")
        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 CRToNLMain
    Dim oCurs(), i%, sPrompt$
sPrompt$ = "Convert New Paragraphs to New Lines for the ENTIRE document?"
If Not CreateSelectedTextIterator(ThisComponent, sPrompt$, oCurs()) Then Exit Sub
For i% = LBOUND(oCurs()) To UBOUND(oCurs())
    CRToNLWorker(oCurs(i%, 0), oCurs(i%, 1), ThisComponent.Text)
Next i%
End Sub
Sub CRToNLWorker(oLCurs As Object, oRCurs As Object, oText As Object)
    If IsNull(oLCurs) Or IsNull(oRCurs) Or IsNull(oText) Then Exit Sub
    If oText.compareRegionEnds(oLCurs, oRCurs) <= 0 Then Exit Sub
    oLCurs.goRight(0, False)
    Do While oLCurs.gotoNextParagraph(False) AND oText.compareRegionEnds(oLCurs, oRCurs) >= 0
        oLCurs.goLeft(1, True)
        oLCurs.setString(""
        oText.insertControlCharacter(oLCurs, com.sun.star.text.ControlCharacter.LINE_BREAK, True)
    Loop
End Sub
'SpaceToTabsInWordsMain
Sub SpaceToTabsInWordsMain
    Dim oCurs(), i%, sPrompt$
    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
            bChanged = False
        ElseIf iCurrentState = InWordState Then
            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
            End Select
        ElseIf iCurrentState = BetweenWordState Then
            bChanged = True
        End If
    Lo
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
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
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()

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)
        End Select
        lCurrentState = iNextState(iCurrentState, iWhatFound, 1)
        lIndex = lIndex + 1
    End Do
End For
'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 character 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$, bSomethingChanged As Boolean
   'There may be multiple selections present!
   'There will probably be one more than expected because
   'it will count the current cursor location as one piece
   'of selected text, just so you know!
   oSels = ThisComponent.getCurrentSelection()
lSelCount = oSels.getCount()
   'Print "total selected = " + lSelCount
   For iWhichSelection = 0 To lSelCount - 1
      bSomethingChanged = False
      s = oSels.getByIndex(iWhichSelection).getString()
      'Print "Text group " + iWhichSelection + " is of length " + Len(s)
      lIndex = 1
      Do While lIndex < Len(s)
         'Print "ASCII at " + lIndex + " = " + Asc(Mid(s, lIndex, 1))
         If Asc(Mid(s, lIndex, 1)) = 9 Then
            oCursor.setString(Chr$(9) + Mid(s, lIndex, 1))
            oCursor.goRight(2, False)
            'Print "Inserted a tab"
         Else
            oCursor.setString(Mid(s, lIndex, 1))
            oCursor.goRight(1, False)
            'Print "Inserted Something"
         End If
      lIndex = lIndex + 1
   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, BOLDDASHDOT, BOLDDASHDOT, BOLDDASHDOT,
'BOLDDASHDOTDOT, BOLDDASHDOT, BOLDDASHDOT, BOLDDASHDOT
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)
    End If

182
7.7.2. Inserting a Note (Annotation)

**Listing 7.32:** Add a note at the cursor.

```vba
Sub AddNoteAtCursor
    Dim vDoc, vViewCursor, oCurs, vTextField
    Dim $s
    'Let's 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"
        'Omit 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:


which discusses two properties. The `ParagraphProperties.PageNumberOffset` states: “If a page break property is set at a paragraph, this property contains the new value for the page number.” The `ParagraphProperties.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.

```javascript
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.

```javascript
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)
        End If
    Loop
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.

```vba
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 of 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

```vba
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.getByName("PageStyles").getByName(sName)
    REM Use FooterIsOn to toggle the footer state.
    If oStyle.HEADER.isOn <> bHeaderState Then
        oStyle.HEADER.isOn = 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 = "47BB4CB-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.ParaStyleName = "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.paraStyleName = "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.

```plaintext
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](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.

```vbnet
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 Marcellly.

See Also:

**Listing 7.43: Search selected text**

```vba
'******************************************************************
'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 deleteTextBetweenDlimiters
    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
        End If
    End Do
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.

```vbnet
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())
End Sub
```
REM Now do the work!
oxDoc.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
    End Do
End Sub
7.15. Changing The Case Of Words

OOo determines the case of a word based on a character property. In theory this means that you can select the entire document and then set the character case. In practice, however, the selected portion may not support the character case property. As a compromise between speed and possible problems, I opted to use a word cursor to traverse the text setting the case of each word individually. I wrote the macro to work on entire words as an arbitrary decision. If you do not like it, change it. I used my selected text framework, so you require those macros for these to work.

If you find that you have text that does not support setting the character case, you can 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.

**Warning** In OOo version 1.0.3.1, title case is broken; “heLLo” shows as “HeLLo”.

---

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(), i%, sMapType$, iMapType$
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

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.

Dim oVCurs
Dim oTChars
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 oTChars
oVCurs = ThisComponent.getCurrentController().getViewCursor()
oTChars = oVCurs.getText().createTextCursorByRange(oVCurs)
oTChars.gotoNextParagraph(False) 'goto start of next paragraph.
oTChars.gotoEndOfParagraph(True) 'Select entire paragraph.
MsgBox "(" & oTChars.getString() & ")"
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.

<table>
<thead>
<tr>
<th>Description</th>
<th>Original Style</th>
<th>New Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macro with one line.</td>
<td>_code_one_line</td>
<td>_OooComputerCodeLastLine</td>
</tr>
<tr>
<td>First line</td>
<td>_code_first_line</td>
<td>_OooComputerCode</td>
</tr>
<tr>
<td>Last line</td>
<td>_code_last_line</td>
<td>_OooComputerCodeLastLine</td>
</tr>
<tr>
<td>Middle lines</td>
<td>_code</td>
<td>_OooComputerCode</td>
</tr>
</tbody>
</table>

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$, midStyle$, _
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
  End If

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$
  End Select

End Do
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 = firstStyle$
        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 If

REM Done with the work so advance the trailing cursor
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?**

```vba
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.**

```vba
Sub EnumerateParagraphs
    Dim oParEnum 'Enumerator used to enumerate the paragraphs
    Dim oPar 'The enumerated paragraph

    oParEnum = ThisComponent.getText().createEnumeration()
    Do While oParEnum.hasMoreElements()
        oPar = oParEnum.nextElement()
    Loop
End Sub
```
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
    End While
  End If
End While

End Sub
Loop
  MsgBox s, 0, "Paragraph " & i
End If
Loop
End Sub

![Figure 7.1: Enumerated paragraph.](image)

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!

### 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.**

```vba
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"
```
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

201
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
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

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 7.56). 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 7.56: 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 7.57: 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 7.58: 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 
   End While 
Loop
Loop

Putting it all together in a single program yields the following:

**Listing 7.59: 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"

203
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 " & _
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

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 position
        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
Sub DeleteCurrentLine
    Dim oVCurs
    oVCurs = ThisComponent.getCurrentController().getViewCursor()
    oVCurs.gotoStartOfLine(False)
    oVCurs.gotoEndOfLine(True)
    oVCurs.setString(""
End Sub

To delete the current paragraph, use 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 (oCurs.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__
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.

```vba
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@pitonyak.org"
    ' oVCursor.HyperLinkURL = "andrew@pitonyak.org"
```
7.20. Sorting Text

A text cursor can be used to sort data in a Write document.

```vba
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
```

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 7.60 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 7.60: 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 7.61 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 7.61: 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 "Other properties"
                oProp.Value = sNames(i)
        End Select
    Next j
Next i
End Sub

211
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 7.61 contains three entries, then only the first three entries in the outline numbering are modified. Use the code as a starting point. Listing 7.62 demonstrates how to use both Listing 7.60 and Listing 7.61.

Listing 7.62: 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. 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
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!
REM oIndex.update()
End Sub

### 7.23. 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 7.63: 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

7.23.1. 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 7.64: 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

214
s = "com.sun.star.text.TextSection"
oSect = ThisComponent.createInstance(s)
oSect.setName(sName)
REM Now, create the columns...
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

7.24. 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 7.65: 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.
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.getByIndex(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.**

```vba
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?**

```vba
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?**

```vba
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.

```vba
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.

```vba
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.*

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>D1</td>
</tr>
<tr>
<td>A2</td>
<td>B2</td>
<td>C2</td>
<td>D2</td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>D3</td>
</tr>
<tr>
<td>A4</td>
<td>B4</td>
<td>C4</td>
<td>D4</td>
</tr>
</tbody>
</table>

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.*

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getByIndex</td>
<td>Retrieve a specific row.</td>
</tr>
<tr>
<td>getCount</td>
<td>Number of rows in the table.</td>
</tr>
<tr>
<td>hasElements</td>
<td>Determine if there are any rows in the table.</td>
</tr>
<tr>
<td>insertByIndex</td>
<td>Add rows to the table.</td>
</tr>
<tr>
<td>removeByIndex</td>
<td>Remove rows from the table.</td>
</tr>
</tbody>
</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.

```vba
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.

```vba
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
```
Text tables

oTable = oUseTable
End If
If IsNull(oTable) OR IsEmpty(oTable) Then
    Print "FormatTable: No table specified"
    Exit Sub
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.HorizontalLine : x.OuterLineWidth = 0 : v.HorizontalLine = 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.ParagraphStyleName = 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 8.3*.

1) Right click in cell A2 and choose Cell > Split > Horizontal. Cell A2 becomes two cells, the rest of the cells are not renamed. At this point, the API indicates that there are four rows and zero columns.


<table>
<thead>
<tr>
<th></th>
<th>A1</th>
<th>B1</th>
<th>C1</th>
<th>D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2=&gt;A2.1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>B3</td>
<td></td>
<td>C3</td>
<td>D3</td>
</tr>
<tr>
<td>A4</td>
<td>B4</td>
<td></td>
<td>C4</td>
<td>D4</td>
</tr>
</tbody>
</table>

3) Select cells A3 and B3, right click and choose Cell > Merge. Notice that the two cells A3 and B3 are now called A3. The cells C3 and D3 are also renamed.

<table>
<thead>
<tr>
<th></th>
<th>A1</th>
<th>B1</th>
<th>C1</th>
<th>D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2=&gt;A2.1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3, B3=&gt;A3</td>
<td></td>
<td>C3=&gt;B3</td>
<td></td>
<td>D3=&gt;C3</td>
</tr>
<tr>
<td>A4</td>
<td>B4</td>
<td></td>
<td>C4</td>
<td>D4</td>
</tr>
</tbody>
</table>

4) Select cells C3=>B3 and C4, right click and choose Cell > Merge.
Table 8.5. Complex table after merging cells in the same column.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>D1</td>
</tr>
<tr>
<td>A2=&gt;A2.1.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3, B3=&gt;A3=&gt;A3.1.1</td>
<td>(C3=&gt;B3),C4=&gt;B3</td>
<td></td>
<td>D3=&gt;C3=&gt;C3.1.1</td>
</tr>
<tr>
<td>A4=&gt;A3.1.2</td>
<td>B4=&gt;A3.2.2</td>
<td></td>
<td>D4=&gt;C3.1.2</td>
</tr>
</tbody>
</table>

The final merge, caused all sorts of mayhem to occur. Notice that the cells A4 and B4 are now named A3.1.2 and A3.2.2. A similar thing happened for C3 and D4 from Table 8.4. After step 4, the API reports that there are now three rows and zero columns.

**TIP**

Obviously, a text table can not have zero columns, so this might be a good way to quickly determine if a text table is simple. On the other hand, this is probably not documented behavior, so it might change in the future.

### 8.2.4. Enumerating cells in any text table

In a simple text table you can use `getCellByPosition(col, row)` to obtain a cell; this is not possible in a complex text table. Every text table supports the method `getCellByName()`, which returns a cell based on its name. Use the `getCellNames()` to return an array of cell names contained in the table.

**Listing 8.9: Enumerate the cells in any text table.**

```vba
Dim sNames() As String
Dim i As Long
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 obtain a list of all cell names as follows:
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.

```vba
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.

```vba
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 cannot 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.**

```plaintext
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 8.6.

**Table 8.6: Text table properties related to the text table width.**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeftMargin</td>
<td>Left margin of the table.</td>
</tr>
<tr>
<td>RightMargin</td>
<td>Right margin of the table.</td>
</tr>
</tbody>
</table>
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.**

```plaintext
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 $      '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!
    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.**

```vba
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.**

```vba
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.**

```vba
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).
  '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.
```
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.**

```vba
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.**

```vba
'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
```

```vba
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. Formatting macros

This section contains the macros I use to format macros in text documents so that they look like macros viewed in the IDE.

9.1. String and array utilities

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.

REM  *****  BASIC  *****
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
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.**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsWhiteSpace(Integer)</td>
<td>True if the character is a space or tab.</td>
</tr>
<tr>
<td>FMT_IsSpecialChar(Integer)</td>
<td>True if the character is special such as a *,;,&amp;.</td>
</tr>
<tr>
<td>FMT_IsWordSep(Integer)</td>
<td>True if special character or white space.</td>
</tr>
<tr>
<td>FMT_IsDigit(Integer)</td>
<td>True if a numeric digit [0 – 9].</td>
</tr>
<tr>
<td>FMT_IsHexDigit(Integer)</td>
<td>True if a numeric digit [0 – 9], [A-F], or [a-f].</td>
</tr>
<tr>
<td>FMT_IsOctDigit(Integer)</td>
<td>True if a numeric digit [0 – 7].</td>
</tr>
<tr>
<td>FMT_StrIsDigit(String)</td>
<td>True if the first character of the string is a digit.</td>
</tr>
</tbody>
</table>

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 OOo 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 is 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 is 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

235
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.**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMT_FindNextNonSpace</td>
<td>Used to quickly skip white space starting from iPos.</td>
</tr>
<tr>
<td>FMT_FindEndQuote</td>
<td>Search for the next quote character.</td>
</tr>
<tr>
<td>FMT_FindEndQuoteEscape</td>
<td>Search for the next quote character, ignoring any character preceded by a backslash () character.</td>
</tr>
<tr>
<td>FMT_FindEndQuoteDouble</td>
<td>Search for the next quote character, ignoring any character followed by a quote character. For example, the text &quot;&quot;&quot;&quot; finds all quote characters.</td>
</tr>
<tr>
<td>FMT_FindNumberEnd</td>
<td>Identify a number, including 0xFF</td>
</tr>
</tbody>
</table>

**Listing 9.4: Functions to find the next relevant special character.**

```vbnet
'******************************************************************************
'** 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

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

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

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

237
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
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.

```vbscript
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(CDb1(nIts * nMinIts) * 1000 / CDb1(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 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 = iUB + 1
            'Exit Do
        ElseIf sItems(i) > s Then
            iUB = i - 1
        Else
            iLB = i + 1
        End If
    Loop While iUB > iLB
End Function
9.2. Utilities to find code sections

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.

```vbnet
'******************************************************************************
'** 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())
    '      oCurs.gotoPreviousParagraph(False) Then Exit Do Loop
    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())
        oCurs.gotoNextParagraph(False) Then Exit Do Loop
    If NOT FMT_ArrayHasString(oCurs.ParaStyleName, sStyles()) Then
        oCurs.gotoPreviousParagraph(False)
    End If
End Sub
```

242
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.

9.3. Formatting using 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.

Listing 9.8: 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("CharHeight", 12), _
                        CreateProperty("CharColor", RGB(0, 0, 0)), _
                        CreateProperty("CharNoHyphenation", True))
    CreateCharacterStyle("OOoComputerCode", oProps())
    REM Base style for normal listings. Reduces the character height
    REM but still ignores color
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("CharHeight", 9))
    CreateCharacterStyle("_OOoComputerBase", oProps())

    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("ParentStyle", "_OOoComputerBase"), _
                      CreateProperty("CharColor", RGB(0, 128, 0)))
    CreateCharacterStyle("_OOoComputerIdent", oProps())
End Sub

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 = ""

    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("ParentStyle", "_OOoComputerBase"), _
                      CreateProperty("CharColor", RGB(0, 128, 0)))
    CreateCharacterStyle("_OOoComputerIdent", oProps())
End Sub
noLocale.Language = "zxx"

CreateBaseCharStyles()

oProps() = Array(CreateProperty("ParentStyle", "_OoComputerBase"), CreateProperty("CharColor", RGB(153, 204, 255)))
CreateCharacterStyle("_JavaComment", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OoComputerBase"), CreateProperty("CharColor", RGB(0, 0, 255)))
CreateCharacterStyle("_JavaLiteral", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OoComputerBase"), CreateProperty("CharLocale", noLocale), CreateProperty("CharColor", RGB(153, 40, 76)))
CreateCharacterStyle("_JavaKeyword", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OoComputerBase"), CreateProperty("ParentStyle", "_OoComputerBase"), CreateProperty("CharColor", RGB(0, 0, 0)))
CreateCharacterStyle("_JavaIdent", oProps())

End Sub

****************************************************************
** Create a character style if it does not exist.  
****************************************************************

Sub CreateCharacterStyle(sStyleName$, oProps())
  Dim sMsg As String, n%, 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
      oStyle.ParentStyle = oProps(i).Value
    Else
      oStyle.setPropertyValue(oProps(i).Name, oProps(i).Value)
    End If
  Next
  oStyles.insertByName(sStyleName, oStyle)
End Sub
9.4. The main module

Character styles are used to format the macro. The character style names are obtained from a function. The order is important.

Listing 9.9: Styles for Basic

```vba
'*** 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 9.10: Initialize keywords and paragraph style arrays.

```vba
'*** 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.hxx. '*** 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", _,
        "defsg", "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", "implementations", "in", "input", _,
        "integer", "is", _,
        "let", "lib", "line", "line input", "local", _,
        "lock", "long", _,
        "loop", "lprint", "lset", _,
        "mod", _,
        "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", _,
        "_OOoComputerCodeLastLine", _,
        "_code", _
        247
9.4.1. How to 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 9.14).

The hard work is in the main worker macro.

Listing 9.11: Format the current macro.

```vbscript
'****************************************************************
'** 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
    Print "The text cursor must be in a code segment"
    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 9.12: Highlight all macros in the document.

```vbscript
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)
                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 paragraphs. The main worker moves one paragraph at a time so the last paragraph may not be processed if it is not entirely selected.

**Listing 9.13: Highlight selected macros**

REM Format just the selected text
Sub HighlightSel()
    Dim oSels
    Dim oSel
    Dim i%

249
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 9.14: 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%    'Temporary integer variable.
    Dim iLen%    'Current character
    Dim sChar$   '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.
    oCurs = oText.createTextCursorByRange(oSel.getStart())
    oCurs.gotoEndOfParagraph(True)

    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.

250
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)

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 oCurs.gotoEndOfParagraph(True) Then Exit Do
Loop
End Sub

9.5. Formatting Java

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.

Listing 9.15: 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
  Print "The text cursor must be in a code segment"
  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())

'****************************************************************
'** 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 = oText.createTextCursorByRange(oSel.getStart())
oCurs.gotoEndOfParagraph(True)

  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
      End If
      REM Check for "**" characters
      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
            Exit Do
         End If
      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) = "/

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 = iPos + 1

Else     sChar = """"

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)

255
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 oCurs.gotoEndOfParagraph(True) Then Exit Do
Loop
End Sub

Sub FMT_InitTokensJava(sTokens())
    sTokens() = Array( "abstract", "assert", _

256
"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 9.16: Example highlighted Java code**

```java
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);
}
```

257
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. Formatting Cpp

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 9.17: C++ character styles.

'****************************************************************
'** 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

The keywords are as follows:

Listing 9.18: C++ Keywords.

'****************************************************************
'** Tokens recognized by c++
'****************************************************************
Sub FMT_InitTokensCpp(sTokens())
    sTokens() = Array("asm", "auto", "bool", "break", _
    "case", "catch", "char", "class", "const", "const_cast", "continue", _
    "default", "define", "defined", "delete", _
    "do", "double", "dynamic_cast", _
    "else", "endif", "enum", "explicit", "export", "extern", _
    "false", "float", "for", "friend", "goto", _
    "if", "ifdef", "include", "inline", "int", "long", _
Main program to format C++

Listing 9.19: C++ Highlight code.

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
        Print "The text cursor must be in a code segment"
        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

'****************************************************************
'** Very simple parsing of Java code.
'****************************************************************
Sub FMT_ColorCodeOneRangeStringCpp(oSel, sTokens(), sCharStyles())
    FMT_ColorCodeOneRangeStringJava(oSel, sTokens(), sCharStyles())
End Sub
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](mailto: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/XFormsSupplier.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**

```vba
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:

```vba
Dim oDlg As Object
Sub StartDialog
    oDlg = CreateUnoDialog(DialogLibraries.Standard.Dialog1)
    oDlg.execute()
End Sub
```

261
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.

```vba
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.

<table>
<thead>
<tr>
<th>Tip</th>
<th>You must have a variable referencing the dialog that has scope outside the subroutine that creates it so that other subroutines can access it.</th>
</tr>
</thead>
</table>
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.
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 addItems(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.addItems(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/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/XCheckBox.html
10.2.8. Option/Radio Button

The purpose of an option 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.**

```vba
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.**

```vba
    x = oDlg.getControls()
For ii=LB bound(x) To UB bound(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.**

```vba
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
  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**

```vba
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
```
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 a “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.**

```vba
Sub ButtonCall(x)
    Dim oButton As Button ' Button that was used to call the handler.
    Dim oModel As Object ' The model for the button.
    Dim oShape As Shape ' 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.

```vba
Sub ExampleGetAFileName
    Dim filterNames(1) As String
    filterNames(0) = "*.txt"
    filterNames(1) = "*.sxw"
    Print GetAFileName(filterNames)
End Sub

Function GetAFileName(FilterNames() As String) 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)
    '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")
    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
End Function
```

268
'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.setPosition(iXPos, iYPos, DialogWidth, _
                        DialogHeight, com.sun.star.awt.PosSize.POS)
    oDialog.execute()
End Sub

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 10.9: 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.getCurrentController
    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"

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.

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.

### 10.7.1. Inserting a formula

![Insert OLE Object dialog](image)

*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.

<table>
<thead>
<tr>
<th>Child</th>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Create new</td>
<td>RADIO_BUTTON</td>
</tr>
<tr>
<td>1</td>
<td>Create from file</td>
<td>RADIO_BUTTON</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>PANEL</td>
</tr>
<tr>
<td>3</td>
<td>Object type</td>
<td>SEPARATOR</td>
</tr>
<tr>
<td>4</td>
<td>OK</td>
<td>PUSH_BUTTON</td>
</tr>
<tr>
<td>5</td>
<td>Cancel</td>
<td>PUSH_BUTTON</td>
</tr>
<tr>
<td>6</td>
<td>Help</td>
<td>PUSH_BUTTON</td>
</tr>
</tbody>
</table>
The macro in Listing 10.10 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 10.10: Insert a formula into writer.**

```vba
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
```

```vba
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

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 10.11. 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 10.11: 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", "GLASSPane", _
"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.getAccessibleChild(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
End Function

The Options dialog is very complicated, which makes it an excellent item to inspect. The
code shown in Listing 10.12 is not complete. The code is modified from Listing 10.10; 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 10.12: How to inspect the accessible content.

Option Explicit
Private sss$

Sub ManipulateOptions()
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 10.13: Search for the specified child.**

```vba
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  
    End If  
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 10.14: Collapse all of the trees in the options dialog.

```vba
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.AccessibleChildCount-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
End If
Next k
End Function
```
Finally, after opening the Options dialog, you need to manipulate it.

**Listing 10.15: 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 10.16 requires the macro shown in Listing 10.13.

**Listing 10.16: 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()
' arPrinter = GetAllPrinters()
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
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 10.17:** Find an open dialog based on its title.

```vba
'------------------- 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 10.11 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 10.17. 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 10.18:** 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

280
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
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
End Function
sText = oAC.getAccessibleName()
On Error Goto 0
If sText ="" Then
    sText = "--"
EndIf
s = s + sText

'AccessibleObject name
sProps = Split(oAC.Dbq_Properties,""")
s = s + " " + sProps(1)

'AccessibleRole
lRole=-1
On Error Resume Next
lRole = oAC.getAccessibleRole()
On Error Goto 0
If lRole <>-1 Then
EndIf

'AccessibleDescription
On Error Resume Next
s = s + " " + oAC.getAccessibleDescription()
On Error Goto 0

If HasUnoInterfaces(oAC, "com.sun.star.accessibility.XAccessibleAction") Then
    lActionCount = oAC.getAccessibleActionCount
For k=0 To lActionCount-1
    s = s + " " + k + ": " + oAC.getAccessibleDescription(k)
Next k
EndIf

AccessibleObjectDescriptionString = s
End Function
11. 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
12. Investment example

12.1. 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 $rP$ 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 12.1.

Equation 12.1 \[ FV = P(1+r)^n \]

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 12.2).

Equation 12.2 \[ FV = P\left(1+r \frac{m}{365}\right) \]

Assume that you periodically place money into an account of some kind. Later, you want to know how well the account has done.

**Table 12.1. Terms for calculating interest.**

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_i$</td>
<td>Amount of the $i^{th}$ deposit ($P$ stands for principle). For example, $P_0$ is the amount of the first deposit or payment.</td>
</tr>
<tr>
<td>$t_i$</td>
<td>Date ($t$ stands for time) of the $i^{th}$ deposit.</td>
</tr>
<tr>
<td>$FV$</td>
<td>Final (or future) value.</td>
</tr>
<tr>
<td>$r$</td>
<td>Annual rate of return (how much did I earn each year).</td>
</tr>
<tr>
<td>$C_i$</td>
<td>How much money I added as of the $i^{th}$ deposit.</td>
</tr>
</tbody>
</table>

Adding all of the deposits produces the amount invested.

Equation 12.3 \[ C_n = \sum_{i=0}^{n} P_i \]

12.1.1. Using only simple interest

An ultra simplified example of Equation 12.2 extended to handle multiple deposits over time (even if it is NEVER used in this way) (see Equation 12.4).

Equation 12.4 \[ FV = \sum_{i=0}^{n} P_i \left(1+r(t-t_i)/365\right) \]

Now expand the terms.
Equation 12.5  \[ FV = \sum_{i=0}^{n} P_i + \sum_{i=0}^{n} P_i r(t - t_i)/365 = \sum_{i=0}^{n} P_i + \frac{r}{365} \sum_{i=0}^{n} P_i(t - t_i) \]

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.

Equation 12.6  \[ FV - \sum_{i=0}^{n} P_i = \frac{r}{365} \sum_{i=0}^{n} P_i(t - t_i) \]

Rearrange a little.

Equation 12.7  \[ 365 \frac{FV - \sum_{i=0}^{n} P_i}{\sum_{i=0}^{n} P_i(t - t_i)} = r \]

Listing 12.1, usable as a Calc function, calculates the rate of return shown in Equation 12.7.

**Listing 12.1:** Calculate the yearly rate of return.

```vba
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
    YearlyRateOfReturn = dSum / dTotalDeposit/365
End Function
```
If dSum = 0 Then
    dRate = 0.0
Else
    dRate = 365 * (FinalValue - dTotalDeposit) / dSum
End If
'Print "Final rate = " & dRate
YearlyRateOfReturn = dRate
End Function

12.1.2. Compound the interest

Assume that my money compounds \( k \) times per year and I let it compound \( n \) times at a rate \( r \).

Equation 12.8 \[ FV = P\left(1 + \frac{r}{k}\right)^n \]

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 12.9.

Equation 12.9 \[ FV = \sum_{i=0}^{n} P_i\left(1 + \frac{r}{k}\right)^{(T-t_i)} \]

All variables in Equation 12.9 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 12.1 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 ??
13. 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.

13.1. 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 variables 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 13.1: 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 13.2: Translate a two character string to Esperanto.

```vbc
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
```

289
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 13.3: Insert the Esperanto character into the document.**

```vba
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 13.4**, every key that you type will go to this key listener.

**Listing 13.4: Register the translator.**

```vba
Sub Compose
    oComposerDocView = ThisComponent.getCurrentController
    oComposerKeyHandler = createUnoListener( "Composer_", _
        "com.sun.star.awt.XKeyHandler" )
    oComposerDocView.addKeyHandler( oComposerKeyHandler )
    oComposerInputString = ""
End Sub
```

Removing the key listener is easy.

**Listing 13.5: Remove the translator.**

```vba
Sub ExitCompose
    oComposerDocView.removeKeyHandler( oComposerKeyHandler )
    oComposerInputString = ""
End Sub
```
The “handler” methods that are called automatically are prefaced with “Composer_” as dictated in Listing 13.4. 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 13.6: The keyReleased method.**

```basic
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 13.2 is used to convert the string into an appropriate character. The converted character is inserted into the document using Listing 13.3.

**Listing 13.7: The primary worker method.**

```basic
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
```

13.2. **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](http://www.openoffice.org/licenses/pdl.pdf)

13.2.1. **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.

```basic
oListener = CreateUnoListener( sPrefix , sInterfaceName )
```
This function is described very well in the OOo Basic help.

```vbnet
sListenerName = "com.sun.star.lang.XEventListener"
oListener = CreateUnoListener("prefix_", sListenerName)
MsgBox oListener.Dbg_supportedInterfaces
MsgBox oListener.Dbg_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.

### 13.2.2. 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 13.8: Simple event listener**

```vbnet
Sub Example_Listener
    sListenerName = "com.sun.star.lang.XEventListener"
    oListener = CreateUnoListener("prefix_", sListenerName)
    Dim oArg As New com.sun.star.lang.EventObject
    oListener.disposing( 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.

### 13.2.3. 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:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addSelectionChangeListener</code></td>
<td>Registers an event listener, which is called when the selection changes.</td>
</tr>
<tr>
<td><code>removeSelectionChangeListener</code></td>
<td>Unregisters an event listener which was registered with <code>addSelectionChangeListener</code>.</td>
</tr>
</tbody>
</table>

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.

### 13.2.4. 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 13.9: 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
```

293
'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/XSelectionSupplier.html
http://api.openoffice.org/docs/common/ref/com/sun/star/view/XSelectionChangeListener.html

13.2.5. 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 13.10: 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

End Select

295
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/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

13.2.6. 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 13.11: 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

296
Sub MyApp_disposing(oEvt)
    'nothing to do here
End Sub

Function MyApp_KeyPressied(oEvt) As Boolean
    select case oEvt.KeyChar
        case "t", "a", "b", "u"
            MyApp_KeyPressied = True
            MsgBox "key " " & oEvt.KeyChar & " not allowed!"
        case else
            MyApp_KeyPressied = False
    end select
End Function

Function MyApp_KeyPressReleased(oEvt) As Boolean
    MyApp_KeyPressReleased = 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/InputEvent.html

13.2.6.1. 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 13.12: Trapping special characters in a key hander.

    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_KeyPressied = 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_KeyPressied = True
    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 13.13: Trapping special characters in a key hander.**

```vcl
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!"
    MyAppKeyPressed = True
End If
```

### 13.2.6.2. 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.

### 13.2.7. Example 4: com.sun.star.awt.XMouseClickHandler

This handler allows to intercept mouse clicks in a document.

**Listing 13.14: Complete mouse click handler.**

```vcl
Option Explicit

Global oDocView As Object
Global oMouseClickHandler As Object

Sub RegisterMouseClickHandler
    oDocView = ThisComponent.currentController
    oMouseClickHandler = _
    createUnoListener("MyApp_", "com.sun.star.awt.XMouseClickHandler")
    ' writedbginfo oMouseClickHandler
End Sub
```
```
  oDocView.addMouseClickHandler(oMouseClickHandler)
End Sub

Sub UnregisterMouseClickHandler
  on error resume next
  oDocView.removeMouseClickHandler(oMouseClickHandler)
  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)
    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/XMouseClickHandler.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

**13.2.8. 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 13.15: 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

**13.3. 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.
14. 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!

14.1. 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.

```plaintext
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 index variable in loops
Print i REM This will print the value of i
```

14.2. Variables

14.2.1. 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.

14.2.2. 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:

```vba
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= 1231231231231231234 ' 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”.

```vba
Sub MultipleDeclaration
    Dim a, b As Integer
    Dim c As Long, d As String
    Dim e As Variant, f As Float
    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(f) REM Object, Float is unknown type
    Print TypeName(g) REM Object, because NOT initialized
End Sub
```
14.2.3. 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.

```vba
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
```

14.2.4. 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.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Variable Type</th>
<th>VarType</th>
<th>Auto Type</th>
<th>Defxxx</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Boolean</td>
<td>11</td>
<td></td>
<td>DefBool</td>
</tr>
<tr>
<td>Currency</td>
<td>Currency with 4 Decimal places</td>
<td>6</td>
<td>@</td>
<td>DefCur</td>
</tr>
<tr>
<td>Date</td>
<td>Date</td>
<td>7</td>
<td>#</td>
<td>DefDat</td>
</tr>
<tr>
<td>Double</td>
<td>Double Floating Point</td>
<td>5</td>
<td>#</td>
<td>DefDbl</td>
</tr>
<tr>
<td>Integer</td>
<td>Integer</td>
<td>2</td>
<td>%</td>
<td>DefInt</td>
</tr>
<tr>
<td>Long</td>
<td>Long</td>
<td>3</td>
<td>&amp;</td>
<td>DefLng</td>
</tr>
<tr>
<td>Object</td>
<td>Object</td>
<td>9</td>
<td></td>
<td>DefObj</td>
</tr>
<tr>
<td>Single</td>
<td>Single Floating Point</td>
<td>4</td>
<td>!</td>
<td>DefSin</td>
</tr>
<tr>
<td>String</td>
<td>String</td>
<td>8</td>
<td>$</td>
<td>DefStr</td>
</tr>
<tr>
<td>Variant</td>
<td>Can contain all types specified by the definition</td>
<td>12</td>
<td></td>
<td>DefVar</td>
</tr>
<tr>
<td>Empty</td>
<td>Variable is not initialized</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Null</td>
<td>No valid data</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The ExampleTypes macro demonstrates the behavior.

**Listing 14.1: Example variable types.**

```vbscript
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")
```

306
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

14.2.4.1. 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

14.2.4.2. 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
14.2.4.3. 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.

```vbnet
Dim Age&
Dim Dogs As Long
```

14.2.4.4. 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.

```vbnet
Dim Income@
Dim Cost As Currency
```

14.2.4.5. Single Variables
Single variables are 32-bit numbers. The greatest magnitude is 3.402823 x 10E38. The smallest non-zero magnitude is 1.401298 x 10E-45. Postfixing a variable with the “!” character causes it to become a Single Variable.

```vbnet
Dim Weight!
Dim Height As Single
```

14.2.4.6. Double Variables
Double variables are 64-bit numbers. The greatest magnitude for a double variable is 1.79769313486232 x 10E308. The smallest non-zero magnitude for a double variable is 4.94065645841247 x 10E-324. Postfixing a variable with the “#” character causes it to become a Double Variable.

```vbnet
Dim Weight#
Dim Height As Double
```

14.2.4.7. String Variables
String variables use a 1-byte ASCII character for each character and are limited in length to 64Kbytes. Postfixing a variable with the “$” character causes it to become a String variable.

```vbnet
Dim FirstName$
Dim LastName As String
```
14.2.5. 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 IsNull(var) REM True
End Sub
```

14.2.6. 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.

### 14.2.7. 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.

```basic
Const Gravity = 9.81
```

### 14.2.8. 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

```basic
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.

```basic
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
```

#### 14.2.8.1. 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 }

#### 14.2.8.2. 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).

```basic
LBound(a()) REM 0
LBound(b$()) REM 5
```
14.2.8.3. 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

14.2.8.4. 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.

14.2.9. 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

14.2.10. 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-dimensionalized.

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
REM 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
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

14.2.11. 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.

14.2.12. Comparison Operators

Conditionals generally work as expected but they do not perform short circuit evaluation. The following standard conditional operators are used

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal To</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less Than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater Than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less Than or Equal To</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater Than or Equal To</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Not Equal To</td>
</tr>
<tr>
<td>Is</td>
<td>Are these the same Object</td>
</tr>
</tbody>
</table>

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 ")
    msg = msg + "have greater precedence than XOR" & Chr(13)
    msg = msg & 6 "XOR does "
    msg = msg + IIF(True OR True XOR True, ",", "not ")
    msg = msg + "have greater precedence than OR"
    MsgBox msg
End Sub

14.3. 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.

14.3.1. 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 TestOptI()    REM MMM
    Print TestOptI(,,)    REM 488MM (Error)
    Print TestOptI(1)    REM 488M
    Print TestOptI(1,1)    REM 1MM
    Print TestOptI(1,,)  REM 1MM
    Print TestOptI(1,1,)  REM 1MM
    Print TestOptI(1,2)  REM 12M
    Print TestOptI(1,2,3)  REM 123
    Print TestOptI(1,,3)  REM 14883 (Error)
End Sub
```
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)
End Sub
```

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.

### 14.3.2. 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).
Do While n > 1
    ' This is fun when n is a constant.
    Print n
    n = n - 1
Loop
End Sub

14.3.3. 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

Function RecursiveFactorial(n As Long) As Long
    If n > 1 Then
        RecursiveFactorial = n * RecursiveFactorial(n-1)
    Else
        RecursiveFactorial = 1
    End If
End Function

14.4. Flow Control

14.4.1. 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

### 14.4.2. 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:

```vba
If (Condition) Then
    object = TrueExpression
Else
    object = FalseExpression
End If
max_age = IIf(johns_age > bills_age, johns_age, bills_age)
```

### 14.4.3. 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!

### 14.4.4. For....Next

The on-line help contains an excellent complete description, read it.

Repeat a block of statements a specified number of times.

**Syntax:**

```vba
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.

```vba
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
```

14.4.5. 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.

```vba
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.

```vba
Do Until condition
    Block
Loop
```
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:

```vba
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:

```vba
Do
    Block
Loop Until condition
```

In a “Do Loop”, you can force an immediate exit from the loop with the “Exit Do” statement.

### 14.4.6. 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:

```vba
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.

#### 14.4.6.1. 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.

---

#### 14.4.6.2. 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:
14.4.6.3. 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

14.4.6.4. 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)

14.4.6.5. 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 Marcellly:

    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.

```vbscript
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
```

### 14.4.7. 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.

### 14.4.8. 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.

```vbscript
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
```

321
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.

14.4.9. GoTo

The GoTo statement causes execution to jump to a defined label in the current subroutine. You cannot 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

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.

14.4.10. 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 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

14.4.11. 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

14.4.12. 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.

323
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).

### 14.4.12.1. 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 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).

### 14.4.12.2. 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.

```vbnet
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

14.4.12.3. An Example
The following example demonstrates many excellent error handling examples.

'******************************************************************
'Author: Bernard Marcelly
'email: marcelly@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

325
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

14.5. 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!

**********

******
15. 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 15.1.

Table 1. OOo BASIC Keywords deprecated when moving to VB to VB.NET.

<table>
<thead>
<tr>
<th>Word</th>
<th>Word</th>
<th>Word</th>
<th>Word</th>
<th>Word</th>
<th>Word</th>
<th>Word</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atn</td>
<td>Currency</td>
<td>DefBool</td>
<td>DefDate</td>
<td>DefDb1</td>
<td>DefInt</td>
<td>DefLng</td>
<td></td>
</tr>
<tr>
<td>DefObj</td>
<td>DefVar</td>
<td>Empty</td>
<td>Eqv</td>
<td>GoSub</td>
<td>Imp</td>
<td>IsEmpty</td>
<td></td>
</tr>
<tr>
<td>IsMissing</td>
<td>IsNull</td>
<td>IsObject</td>
<td>Let</td>
<td>Line</td>
<td>LSet</td>
<td>MsgBox</td>
<td></td>
</tr>
<tr>
<td>Now</td>
<td>Null</td>
<td>On?GoSub</td>
<td>On?GoTo</td>
<td>Option Base</td>
<td>Private</td>
<td>Rnd</td>
<td></td>
</tr>
<tr>
<td>RSet</td>
<td>Set</td>
<td>Sgn</td>
<td>Sqr</td>
<td>Wend</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
15.1. Data types

Table 15.2: VB.NET uses different names for some numerical functions.

<table>
<thead>
<tr>
<th></th>
<th>OOo BASIC</th>
<th>VB</th>
<th>VB.NET</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>Byte</td>
<td>Byte</td>
<td>0 through 255, OOo BASIC uses the CByte function to create one.</td>
<td></td>
</tr>
<tr>
<td>Integer</td>
<td>Integer</td>
<td>Short</td>
<td>–32,768 through 32,767</td>
<td></td>
</tr>
<tr>
<td>Long</td>
<td>Long</td>
<td>Integer</td>
<td>–2,147,483,648 through 2,147,483,647</td>
<td></td>
</tr>
<tr>
<td>Currency</td>
<td>Currency</td>
<td>Currency</td>
<td>+/-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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Decimal</td>
<td>Decimal</td>
<td>+/-79,228,162,514,264,337,593,543,950,335 for numbers with no decimal places. The smallest possible non-zero number is 0.00000000000000000000000000000000000000001.</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>String</td>
<td>String</td>
<td>OOo BASIC has a limit of 65,535 Unicode characters, VB may contain approximately 2 billion Unicode characters.</td>
<td></td>
</tr>
</tbody>
</table>

15.2. 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 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.
15.3. **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.

15.4. **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.

15.5. **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 higer 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.

15.6. Subroutines and Functions

15.6.1. 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 15.3: VB.NET uses different names for some numerical functions.

<table>
<thead>
<tr>
<th>OOo BASIC</th>
<th>VB</th>
<th>VB.NET</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>ABS</td>
<td>Math.Abs</td>
<td>The absolute value of a specified number.</td>
</tr>
<tr>
<td>ATN</td>
<td>ATN</td>
<td>Math.Atan</td>
<td>The angle whose tangent is the specified number.</td>
</tr>
<tr>
<td>COS</td>
<td>COS</td>
<td>Math.Cos</td>
<td>The cosine of the specified angle.</td>
</tr>
<tr>
<td>Exp</td>
<td>Exp</td>
<td>Math.Exp</td>
<td>The base of natural logarithms raised to a power.</td>
</tr>
<tr>
<td>Log</td>
<td>Log</td>
<td>Math.Log</td>
<td>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.</td>
</tr>
<tr>
<td>not supported</td>
<td>Round</td>
<td>Math.Round</td>
<td>Value containing the number nearest the specified value.</td>
</tr>
<tr>
<td>Sgn</td>
<td>Sgn</td>
<td>Math.Sign</td>
<td>Integer value indicating the sign of a number.</td>
</tr>
<tr>
<td>SIN</td>
<td>SIN</td>
<td>Math.Sin</td>
<td>The sine of an angle.</td>
</tr>
<tr>
<td>Sqr</td>
<td>Sqr</td>
<td>Math.Sqrt</td>
<td>The square root of a number.</td>
</tr>
<tr>
<td>TAN</td>
<td>TAN</td>
<td>Math.Tan</td>
<td>The tangent of an angle.</td>
</tr>
</tbody>
</table>

• 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 15.4: OOo BASIC Keywords deprecated when moving to VB to VB.NET.

<table>
<thead>
<tr>
<th>Word</th>
<th>Word</th>
<th>Word</th>
<th>Word</th>
<th>Word</th>
<th>Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsEmpty</td>
<td>IsNull</td>
<td>IsObject</td>
<td>Line</td>
<td>LSet</td>
<td>MsgBox</td>
</tr>
</tbody>
</table>

15.7. 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:

```vbnet
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.
16. 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.

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>AND</td>
<td>Bitwise on numerics and logical on Boolean</td>
</tr>
<tr>
<td>0</td>
<td>OR</td>
<td>Bitwise on numerics and logical on Boolean</td>
</tr>
<tr>
<td>0</td>
<td>XOR</td>
<td>Bitwise on numerics and logical on Boolean</td>
</tr>
<tr>
<td>0</td>
<td>EQV</td>
<td>Logical and/or Bitwise equivalence</td>
</tr>
<tr>
<td>0</td>
<td>IMP</td>
<td>Logical Implication (buggy as of 1.0.3.1)</td>
</tr>
<tr>
<td>1</td>
<td>=</td>
<td>Logical</td>
</tr>
<tr>
<td>1</td>
<td>&lt;</td>
<td>Logical</td>
</tr>
<tr>
<td>1</td>
<td>&gt;</td>
<td>Logical</td>
</tr>
<tr>
<td>1</td>
<td>&lt;=</td>
<td>Logical</td>
</tr>
<tr>
<td>1</td>
<td>&gt;=</td>
<td>Logical</td>
</tr>
<tr>
<td>1</td>
<td>&lt;&gt;</td>
<td>Logical</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>Numerical Subtraction</td>
</tr>
<tr>
<td>2</td>
<td>+</td>
<td>Numerical Addition and String Concatenation</td>
</tr>
<tr>
<td>2</td>
<td>&amp;</td>
<td>String Concatenation</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td>Numerical Multiplication</td>
</tr>
<tr>
<td>3</td>
<td>/</td>
<td>Numerical Division</td>
</tr>
<tr>
<td>3</td>
<td>MOD</td>
<td>Numerical remainder after division</td>
</tr>
<tr>
<td>4</td>
<td>^</td>
<td>Numerical Exponentiation</td>
</tr>
</tbody>
</table>

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.
17. String Manipulations

offers a few methods for the manipulation of strings.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asc(s$)</td>
<td>ASCII value of the first character in the string.</td>
</tr>
<tr>
<td>Chr$(i)</td>
<td>Return the character corresponding to the ASCII code.</td>
</tr>
<tr>
<td>CStr(Expression)</td>
<td>Convert the numeric expression to a string.</td>
</tr>
<tr>
<td>Format(number [, f])</td>
<td>Format the number based on the format string.</td>
</tr>
<tr>
<td>Hex(Number)</td>
<td>String that represents the hexadecimal value of a number.</td>
</tr>
<tr>
<td>InStr([i,] s$, f$[, c])</td>
<td>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.</td>
</tr>
<tr>
<td>LCase(s$)</td>
<td>Returns string as all lower case.</td>
</tr>
<tr>
<td>Left(s$, n)</td>
<td>Return the leftmost n characters from s. n is an integer but the string may be 64K in size.</td>
</tr>
<tr>
<td>Len(s$)</td>
<td>Returns the length of the string s.</td>
</tr>
<tr>
<td>LSet s$ = Text</td>
<td>Left align a string. Broken in 1.0.3.1, fixed in 1.1.</td>
</tr>
<tr>
<td>LTrim(s$)</td>
<td>Return a string with no leading spaces. Does not modify the string.</td>
</tr>
<tr>
<td>Mid(s$, i, n])</td>
<td>Substring from location i of length n.</td>
</tr>
<tr>
<td>Mid(s$, i, n, r$)</td>
<td>Replace the substring with r with limitations. I use to delete portions.</td>
</tr>
<tr>
<td>Oct(Number)</td>
<td>String that represents the Octal value of a number.</td>
</tr>
<tr>
<td>Right(s$, n)</td>
<td>Return the rightmost n characters from s. n is an integer but the string may be 64K in size.</td>
</tr>
<tr>
<td>RSet s$ = Text</td>
<td>Right align a string.</td>
</tr>
<tr>
<td>RTrim(s$)</td>
<td>Return string with no trailing spaces.</td>
</tr>
<tr>
<td>Space(n)</td>
<td>Returns a string that consists of a specified amount of spaces.</td>
</tr>
<tr>
<td>Str(Expression)</td>
<td>Convert the numeric expression to a string.</td>
</tr>
<tr>
<td>StrComp(x$, y$[, c])</td>
<td>Return -1 if x&gt;y, 0 if x=y, and 1 if x&lt;y. If c=1 then case-insensitive.</td>
</tr>
<tr>
<td>String(n, {i</td>
<td>s$})</td>
</tr>
<tr>
<td>Trim(s$)</td>
<td>Return a string with no leading or trailing space from the string.</td>
</tr>
<tr>
<td>UCase(s$)</td>
<td>Returns string as all upper case.</td>
</tr>
<tr>
<td>Val(s$)</td>
<td>Convert the string to a number.</td>
</tr>
</tbody>
</table>

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
17.1. 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

17.2. 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 string. (1 based)
REM num is a long indicating how many characters to replace.
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
17.3. 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

17.4. 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
# 18. Numeric Manipulations

<table>
<thead>
<tr>
<th><strong>Function</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Abs(Number)</td>
<td>Return the absolute value of the number as a Double.</td>
</tr>
<tr>
<td>Asc(s$)</td>
<td>Return the ASCII Integer of the first character in the string.</td>
</tr>
<tr>
<td>Atn(x)</td>
<td>Return the angle, in radians, whose tangent is x</td>
</tr>
<tr>
<td>Blue(color)</td>
<td>Returns the Blue component of the given color code.</td>
</tr>
<tr>
<td>CByte(Expression)</td>
<td>Convert a string or number to a byte.</td>
</tr>
<tr>
<td>CDbl(Expression)</td>
<td>Convert a string or number to a Double.</td>
</tr>
<tr>
<td>CInt(Expression)</td>
<td>Convert a string or number to an Integer.</td>
</tr>
<tr>
<td>CLng(Expression)</td>
<td>Convert a string or number to a Long.</td>
</tr>
<tr>
<td>Cos(x)</td>
<td>Calculates the cosine of an angle specified in radians.</td>
</tr>
<tr>
<td>CSng(Expression)</td>
<td>Convert a string or number to a single precision number.</td>
</tr>
<tr>
<td>CStr(Expression)</td>
<td>Convert a string or number to a String.</td>
</tr>
<tr>
<td>Exp(Expression)</td>
<td>Base of the natural logarithm (e = 2.718282) raised to a power.</td>
</tr>
<tr>
<td>Fix(Expression)</td>
<td>Return the integer portion of a number after truncation.</td>
</tr>
<tr>
<td>Format(number [, f$])</td>
<td>Format the number based on the format string.</td>
</tr>
<tr>
<td>Green(color)</td>
<td>Returns the Green component of the given color code.</td>
</tr>
<tr>
<td>Hex(Number)</td>
<td>String that represents the hexadecimal value of a number.</td>
</tr>
<tr>
<td>Int(Number)</td>
<td>Rounds the integer toward negative infinity. See Also: Fix().</td>
</tr>
<tr>
<td>IsNumeric (Var)</td>
<td>Tests whether the given expression is a number.</td>
</tr>
<tr>
<td>Log(Number)</td>
<td>Natural logarithm of a number.</td>
</tr>
<tr>
<td>Oct(Number)</td>
<td>String that represents the Octal value of a number.</td>
</tr>
<tr>
<td>Randomize [Number]</td>
<td>Initializes the random-number generator.</td>
</tr>
<tr>
<td>Red(color)</td>
<td>Returns the Red component of the given color code.</td>
</tr>
<tr>
<td>RGB (Red, Green, Blue)</td>
<td>Long color value consisting of red, green, and blue components.</td>
</tr>
<tr>
<td>Rnd [(Expression)]</td>
<td>Return a random number between 0 and 1.</td>
</tr>
<tr>
<td>Sgn (Number)</td>
<td>Returns 1, -1, or 0 if the number is positive, negative, or zero.</td>
</tr>
<tr>
<td>Sin(x)</td>
<td>Calculates the sine of an angle specified in radians.</td>
</tr>
<tr>
<td>Sqr(Number)</td>
<td>Square root of a numeric expression.</td>
</tr>
<tr>
<td>Tan(x)</td>
<td>Calculates the tangent of an angle specified in radians.</td>
</tr>
<tr>
<td>n = TwipsPerPixelX</td>
<td>Returns the number of twips representing the width of a pixel.</td>
</tr>
<tr>
<td>n = TwipsPerPixelY</td>
<td>Returns the number of twips representing the height of a pixel.</td>
</tr>
<tr>
<td>Val(s$)</td>
<td>Convert a string to a number.</td>
</tr>
</tbody>
</table>
## 19. Date Manipulations

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDate(Expression)</td>
<td>Convert a string or number to a date.</td>
</tr>
<tr>
<td>CDateFromIso(String)</td>
<td>Return the internal date number from a string containing a date in ISO format.</td>
</tr>
<tr>
<td>CDateToIso(Number)</td>
<td>Returns the date in ISO format from a serial date number that was generated with DateSerial or DateValue.</td>
</tr>
<tr>
<td>Date</td>
<td>Return the current system date.</td>
</tr>
<tr>
<td>Date = s$</td>
<td>Set the current system date.</td>
</tr>
<tr>
<td>DateSerial(y%, m%, d%)</td>
<td>Return a date from the year, month, and day.</td>
</tr>
<tr>
<td>DateValue([date])</td>
<td>Long from a date usable to determine the difference between dates.</td>
</tr>
<tr>
<td>Month(Number)</td>
<td>Day of month from a time generated by DateSerial or DateValue.</td>
</tr>
<tr>
<td>GetSystemTicks()</td>
<td>Returns the system ticks provided by the operating system.</td>
</tr>
<tr>
<td>Hour(Number)</td>
<td>Hour from a time generated by TimeSerial or TimeValue.</td>
</tr>
<tr>
<td>Minute(Number)</td>
<td>Minute from a time generated by TimeSerial or TimeValue.</td>
</tr>
<tr>
<td>Month(Number)</td>
<td>Month from a time generated by DateSerial or DateValue.</td>
</tr>
<tr>
<td>Now</td>
<td>Current system date and time as a Date value.</td>
</tr>
<tr>
<td>Second(Number)</td>
<td>Second from a time generated by TimeSerial or TimeValue.</td>
</tr>
<tr>
<td>Time</td>
<td>Current system time</td>
</tr>
<tr>
<td>Timer</td>
<td>Number of seconds that have elapsed since midnight.</td>
</tr>
<tr>
<td>TimeSerial (h, m, s)</td>
<td>Serial time value from the specified hour, minute, and second.</td>
</tr>
<tr>
<td>TimeValue (s$)</td>
<td>Serial time value from a formatted string.</td>
</tr>
<tr>
<td>Wait millisec</td>
<td>Pause for the given number of milliseconds</td>
</tr>
<tr>
<td>WeekDay(Number)</td>
<td>Day of week from a time generated by DateSerial or DateValue.</td>
</tr>
<tr>
<td>Year(Number)</td>
<td>Year from a time generated by DateSerial or DateValue.</td>
</tr>
</tbody>
</table>
## 20. File Manipulations

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChDir (s$)</td>
<td>Changes the current directory or drive.</td>
</tr>
<tr>
<td>ChDrive(s$)</td>
<td>Changes the current drive.</td>
</tr>
<tr>
<td>Close #n% [, #n2%[,...]]</td>
<td>Close files opened with the Open statement.</td>
</tr>
<tr>
<td>ConvertFromURL(s$)</td>
<td>Converts a file URL to a system file name.</td>
</tr>
<tr>
<td>ConvertToURL(s$)</td>
<td>Converts a system file name to a file URL.</td>
</tr>
<tr>
<td>CurDir([s$])</td>
<td>Returns the current directory of the specified drive.</td>
</tr>
<tr>
<td>Dir [s$ [, Attrib%]]</td>
<td>Perform a directory listing.</td>
</tr>
<tr>
<td>EOF(n%)</td>
<td>Has the file pointer reached the end of the file?</td>
</tr>
<tr>
<td>FileAttr (n%, Attribut%)</td>
<td>Return the file attribute of an open file.</td>
</tr>
<tr>
<td>FileCopy from$, to$</td>
<td>Copy a file.</td>
</tr>
<tr>
<td>FileDateTime(s$)</td>
<td>Return a string of the file date and time.</td>
</tr>
<tr>
<td>FileExists(s$)</td>
<td>Determine if a file or directory exists.</td>
</tr>
<tr>
<td>FileLen (s$)</td>
<td>Length of file in bytes.</td>
</tr>
<tr>
<td>FreeFile</td>
<td>Next available file number. Prevents simultaneous use.</td>
</tr>
<tr>
<td>Get [#]n%, [Pos], v</td>
<td>Read a record or bytes from a file.</td>
</tr>
<tr>
<td>GetAttr(s$)</td>
<td>Returns a bit pattern which identifies the file type.</td>
</tr>
<tr>
<td>Input #n% v1[, v2[, [...]]</td>
<td>Read data from an open sequential file.</td>
</tr>
<tr>
<td>Kill f$</td>
<td>Delete a file form a disk.</td>
</tr>
<tr>
<td>Line Input #n%, v$</td>
<td>Read a string from a sequential file into a variable.</td>
</tr>
<tr>
<td>Loc (FileNumber)</td>
<td>Returns the current position in an open file.</td>
</tr>
<tr>
<td>Lof (FileNumber)</td>
<td>Returns the current size of an open file.</td>
</tr>
<tr>
<td>MkDir s$</td>
<td>Create a new directory.</td>
</tr>
<tr>
<td>Name old$, new$</td>
<td>Rename an existing file or directory.</td>
</tr>
<tr>
<td>Open s$ [#]n%</td>
<td>Open a file. Most parameters not listed, this is very flexible.</td>
</tr>
<tr>
<td>Put [#] n%, [pos], v</td>
<td>Writes a record a sequence of bytes to a file.</td>
</tr>
<tr>
<td>Reset</td>
<td>Closes all open files and flushes all buffers to disk.</td>
</tr>
<tr>
<td>RmDir f$</td>
<td>Remove a directory.</td>
</tr>
<tr>
<td>Seek[#]n%, Pos</td>
<td>Move the file pointer.</td>
</tr>
<tr>
<td>SetAttr f$, Attribute%</td>
<td>Set the file attributes.</td>
</tr>
<tr>
<td>Write [#]n%, [Exprs]</td>
<td>Write data to a sequential file.</td>
</tr>
</tbody>
</table>
21. Operators, Statements, and Functions

21.1. Subtraction operator (-)

Summary:
Subtract two numerical values. The standard mathematical precedence is used as shown on page 335.

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

21.2. Multiplication operator (*)

Summary:
Multiply two numerical values. The standard mathematical precedence is used as shown on page 335.

Syntax:
Result = Expression1 * Expression2

Parameter:
Result : Result of the multiplication.
Expression1, Expression2 : Any numerical expressions.

Example:
Sub MultiplictionExample
    Print 4 * 3        '12
    Print 1.23e2 * 23  '2829
End Sub

21.3. 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 335.
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

21.4. 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 335.

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

21.5. 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 335.

Syntax:
Result = Expression1 / Expression2

Parameter:
Result: Result of the division.
Expression1, Expression2: Any numerical expressions.

**Example:**

```vbscript
Sub DivisionExample
    Print 4 / 2 ' 2
    Print 11 / 2 ' 5.5
End Sub
```

### 21.6. **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 335 and the operation table is shown below.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>x AND y</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Syntax:**
Result = Expression1 AND Expression2

**Parameter:**
Result: Result of the operation.
Expression1, Expression2: Numerical or Boolean expression.

**Example:**

```vbscript
Sub AndExample
    Print (3 AND 1) 'Prints 1
    Print (True AND True) 'Prints -1
    Print (True AND False) 'Prints 0
End Sub
```

### 21.7. **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 \textit{Val} function. If the number is non-negative, then it is returned, otherwise, the negative of the number is returned.

\textbf{Syntax:}
Abs (Number)

\textbf{Return value:}
Double

\textbf{Parameter:}
Number: Any numeric expression.

\textbf{Example:}
\begin{verbatim}
Sub AbsExample
    Print Abs(3) '3
    Print Abs(-4) '4
    Print Abs("-123") '123
End Sub
\end{verbatim}

21.8. \textbf{Array Function}

\textbf{Summary:}
Create a Variant array from the parameter list. This is the quickest method to create an array of constants.

\begin{center}
\hline
\textbf{Warning} \hspace{1cm} 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.
\end{center}

See also the DimArray Function.

\textbf{Syntax:}
Array (Argument list)

\textbf{Return value:}
Variant array containing the argument list.

\textbf{Parameter:}
Argument list: List of values separated by commas from which to create a list.

\textbf{Example:}
\begin{verbatim}
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
\end{verbatim}
21.9. 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

21.10. ATN Function

Summary:
Arctangent of a numeric expression with a return value in the range -π/2 to π/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

21.11. Beep Statement

Summary:
Generate a system beep (sound).

Syntax:
Beep

Example:
Sub ExampleBeep
    Beep
    Beep
End Sub

21.12. 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

352
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

### 21.13. 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

### 21.14. 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

21.15. **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

21.16. **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(1.44532e2)) '145
    Print CByte(64.9) 'A
    Print CByte(65) 'A
    Print Int(CByte("12")) '12
End Sub

21.17. 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

21.18. 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:

```vba
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
```

21.19. **CDateTolso Function**

Summary:
Return the date in ISO format from a serial date number generated with DateSerial or DateValue.

Syntax:
CDateTolso(Number)

Return value:
String

Parameter:
Number : Integer that contains the serial date number.

Example:
```vba
Sub ExampleCDateTolso
    MsgBox "" & CDateTolso(Now) ,64,"ISO Date"
End Sub
```

21.20. **CDbl 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:
```vba
Sub ExampleCDbl
    MsgBox CDbl(1234.5678)
    MsgBox CDbl("1234.5678")
End Sub
```
21.21. **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:**

```vbscript
Sub ExampleChDir
    Dim sDir as String
    sDir = CurDir
    ChDir( "C:\temp" )
    msgbox CurDir
    ChDir( sDir )
    msgbox CurDir
End Sub
```

21.22. **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:**

```vbscript
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
```

21.23. **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.

```vba
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
```

**21.24. 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:

```vbscript
sub ExampleChr
    REM Show "Line 1" and "Line 2" on separate lines.
    MsgBox "Line 1" + Chr$(13) + "Line 2"
End Sub
```

## 21.25. **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:**
```vbscript
Sub ExampleCInt
    MsgBox CInt(1234.5678)
    MsgBox CInt("1234.5678")
End Sub
```

## 21.26. **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:**
```vbscript
Sub ExampleCLng
    MsgBox CLng(1234.5678)
    MsgBox CLng("1234.5678")
End Sub
```
21.27. 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

21.28. 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

21.29. 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$ as string
    s = "file:///c:/temp/file.txt"
    MsgBox s & " => " & ConvertFromURL(s)
    s = "file:///temp/file.txt"
    MsgBox s & " => " & ConvertFromURL(s)
End Sub

21.30. 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$ as string
    s = "c:\temp\file.txt"
21.31. 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

21.32. 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

21.33. 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 figure.

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@qwest.com")
    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"
Message.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.
Mail.sendSimpleMailMessage(vMessage, com.sun.star.system.SimpleMailClientFlags.NO_USER_INTERFACE)
End Sub

21.34. 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")

21.35. 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

21.36. CStr Function

Summary:
Convert any expression to a string expression. This is usually used to convert a number to a string.

<table>
<thead>
<tr>
<th>Input Type</th>
<th>Output Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>“True” or “False”</td>
</tr>
<tr>
<td>Date</td>
<td>String with the date and time</td>
</tr>
<tr>
<td>Null</td>
<td>run-time error</td>
</tr>
<tr>
<td>Empty</td>
<td>empty string “”</td>
</tr>
<tr>
<td>Any Number</td>
<td>The number as a string. Trailing zeros to the right of the decimal are dropped.</td>
</tr>
</tbody>
</table>

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 CDb1(1234.5678)
    MsgBox CInt(1234.5678)
    MsgBox CLng(1234.5678)
    MsgBox CSng(1234.5678)
    sVar = CStr(1234.5678)
    MsgBox sVar
End Sub

21.37. 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

21.38. 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

21.39. 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

21.40. 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

21.41. 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

21.42. 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.

368
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
```

21.43. 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
```
21.44. 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

21.45. 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

21.46. 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 “ii”, 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

### 21.47. 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

### 21.48. 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

### 21.49. 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

### 21.50. **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.

```vba
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

21.51. 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 )

21.52. 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.

373
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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal files.</td>
</tr>
<tr>
<td>2</td>
<td>Hidden files.</td>
</tr>
<tr>
<td>4</td>
<td>System files.</td>
</tr>
<tr>
<td>8</td>
<td>Returns the name of the volume (Exclusive).</td>
</tr>
<tr>
<td>16</td>
<td>Return directories (Exclusive).</td>
</tr>
</tbody>
</table>

**Example:**

```vba
Sub ExampleDir
    REM Displays all files and directories
    Dim sFile As String, sPath As String
    Dim sDir As String, sValue As String
    Dim iFile As Integer
    sFile = "Files: "; sValue = Dir$(sPath, 0 + 2 + 4 + 16)
    Do
        If sValue <> "." and sValue <> ".." Then
            If (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

REM Remove the 16 and you will receive the files
sValue = Dir$(sPath, 0 + 2 + 4)
```

374
End If
End If
sValue = Dir$
Loop Until sValue = ""
MsgBox sDir,0,sPath
MsgBox "" & iFile & " " & sFile,0,sPath
End Sub

<table>
<thead>
<tr>
<th>Tip</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Warning</td>
<td>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.</td>
</tr>
<tr>
<td>Warning</td>
<td>When you obtain a directory listing, files are not returned, even if the online help example seems to indicate that it does.</td>
</tr>
</tbody>
</table>

21.53. Do...Loop Statement

Summary:
Construct to repeat statements.

See Also: Loop Control Page 318.

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]

21.54. End Statement

Summary:
Mark the end of a procedure, block or even a subroutine or function.

See Also: Exit

Syntax:

<table>
<thead>
<tr>
<th>Form</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>End</td>
<td>End by itself ends program execution. It may be entered anywhere. This is optional.</td>
</tr>
<tr>
<td>End Function</td>
<td>Mark the end a function</td>
</tr>
<tr>
<td>End If</td>
<td>Mark the end of an If...Then...Else block.</td>
</tr>
<tr>
<td>End Select</td>
<td>Mark the end of a Select Case block.</td>
</tr>
<tr>
<td>End Sub</td>
<td>Mark the end a subroutine.</td>
</tr>
</tbody>
</table>

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

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

21.55. 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

21.56. **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

21.57. **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

21.58. 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 335 and the operation table is shown below.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x EQV y</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

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

21.59. 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 occured"
End Sub

21.60. 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 occured"
End Sub

21.61. Error statement does not work as indicated.

Summary:
Simulates the occurrence of an error during program execution. This does NOT work!

Syntax:
Error errormumber As Integer

Return value:
Integer

Parameter:
Errormumber: Integer expression that specifies the number of the error to be simulated.

Example:

21.62. 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 occured"
End Sub

21.63. 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:

<table>
<thead>
<tr>
<th>Form</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exit Do</td>
<td>Exit the inner-most Do...Loop.</td>
</tr>
<tr>
<td>Exit For</td>
<td>Exit the inner-most For...Next loop.</td>
</tr>
<tr>
<td>Exit Function</td>
<td>Exit a function and continue execution following the function call.</td>
</tr>
<tr>
<td>Exit Sub</td>
<td>Exit a subroutine and continue execution following the subroutine call.</td>
</tr>
</tbody>
</table>

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
    Dim iSearch 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

21.64. 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:**

```vbnet
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) * 3.0)
end sub
```

### 21.65. 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.

<table>
<thead>
<tr>
<th>Value</th>
<th>Access Mode Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INPUT (file open for input)</td>
</tr>
<tr>
<td>2</td>
<td>OUTPUT (file open for output)</td>
</tr>
<tr>
<td>4</td>
<td>RANDOM (file open for random access)</td>
</tr>
<tr>
<td>8</td>
<td>APPEND (file open for appending)</td>
</tr>
<tr>
<td>32</td>
<td>BINARY (file open in binary mode)</td>
</tr>
</tbody>
</table>

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

### 21.66. **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

### 21.67. **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

### 21.68. **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

### 21.69. **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

21.70. 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"
    REM yes
    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
    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"
        RunSimpleObjectBrowser(oTst)
End Sub
21.71. 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
```

21.72. 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

21.73. For...Next Statement

Summary:
Construct to repeat statements with an auto-incrementing counter.

See Also: For....Next on Page 317.

Syntax:
For counter=start To end [Step step]
  statement block
  [Exit For]
  statement block
Next [counter]

21.74. 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.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>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.</td>
</tr>
<tr>
<td>#</td>
<td>This works like the 0, but leading and trailing zeros are not displayed.</td>
</tr>
<tr>
<td>.</td>
<td>The decimal placeholder determines the number of decimal places to the left and right</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>%</td>
<td>Multiply the number by 100 and insert the percent sign (%) where it appears in the format code.</td>
</tr>
<tr>
<td>E-</td>
<td>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+.</td>
</tr>
<tr>
<td>E+</td>
<td></td>
</tr>
<tr>
<td>e-</td>
<td></td>
</tr>
<tr>
<td>e+</td>
<td></td>
</tr>
<tr>
<td>,</td>
<td>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 #).</td>
</tr>
<tr>
<td>-</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td></td>
</tr>
<tr>
<td>)</td>
<td></td>
</tr>
<tr>
<td>space</td>
<td>(+), minus (-), dollar ($), space, or brackets entered directly in the format code are displayed as the literal character.</td>
</tr>
<tr>
<td>\</td>
<td>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 (@, &amp;, &lt;, &gt;, !). You may also enclose characters in double quotes.</td>
</tr>
<tr>
<td>General Number</td>
<td>Numbers are displayed as entered.</td>
</tr>
<tr>
<td>Currency</td>
<td>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)</td>
</tr>
<tr>
<td>Fixed</td>
<td>At least one digit is displayed in front of the decimal separator. Two decimals are displayed.</td>
</tr>
<tr>
<td>Standard</td>
<td>Displays numbers with a locale specific thousands separator. Two decimals are displayed.</td>
</tr>
<tr>
<td>Scientific</td>
<td>Displays numbers in scientific notation. Two decimals are displayed.</td>
</tr>
</tbody>
</table>

**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:**

```vba
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(0.125555, ",.##")               REM .1256
    MsgBox Format(123.45678, ",.##")              REM 123.46
    MsgBox Format(.0012345678, "0.##")           REM 0.012
    MsgBox Format(123.45678, ".##")               REM 123.46
    MsgBox Format(123.45678, ".## is ###")        REM 123.46 is 679 (strange)
    MsgBox Format(8123.45678, "General Number")   REM 8,123.456789
    MsgBox Format(8123.45678, "Fixed")            REM 8,123.46
    MsgBox Format(8123.45678, "Currency")         REM 8,123.46$ (broken)
    MsgBox Format(8123.45678, "Standard")         REM 8,123.46
    MsgBox Format(8123.45678, "Scientific")       REM 8.123E03
    MsgBox Format(0.00123456789, "Scientific")    REM 1.23E03 (broken)
End Sub
```

**21.75. 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.

### 21.76. 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:**
```vba
Declare Sub MyMessageBeep Lib "user32.dll" Alias "MessageBeep" ( long )
Sub ExampleDeclare
  Dim lValue As Long
  lValue = 5000
  MyMessageBeep( lValue )
  FreeLibrary("user32.dll" )
End Sub
```

### 21.77. 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:**
```vba
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
```
21.78. 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,, "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  Put #iNumber,, "abcdefghijklmnopqrstuvwxyz"
  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   ..ABCDEFGHIJKLMNOPQRSTUVWXYZ..
  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 "another get" & sText
  Get #iNumber,1,sText
  Put #iNumber,20,"This is the text in record 20"
  Print Lof(#iNumber)
  Close #iNumber
End Sub
21.79. GetAttr Function

Summary:
Return a bit pattern identifying the file type. The attributes are a superset of those used in the Dir function.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal file.</td>
</tr>
<tr>
<td>1</td>
<td>Read-Only file</td>
</tr>
<tr>
<td>2</td>
<td>Hidden files.</td>
</tr>
<tr>
<td>4</td>
<td>System file.</td>
</tr>
<tr>
<td>8</td>
<td>Volume name.</td>
</tr>
<tr>
<td>16</td>
<td>Directory.</td>
</tr>
<tr>
<td>32</td>
<td>Archive bit (file changed since last backed up)</td>
</tr>
</tbody>
</table>

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 = 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"
   End If
End Function
If (x AND 32) <> 0 Then s = s & " Archive"
End If
FileAttributeString = s
End Function

21.80. 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")

21.81. 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 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())

OOOVersion=oSet.getbyname("ooSetupVersion")
End Function

Syntax:
s = GetSolarVersion()

**Return value:**
String

**Example:**

```basic
Sub ExampleGetSolarVersion
    REM as of 1.0.3.1, this is "641"
    Print GetSolarVersion()
End Sub
```

### 21.82. 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.

```basic
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
```

### 21.83. 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:**

```basic
GlobalScope
```
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

21.84. 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

21.85. 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

21.86. **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

21.87. **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

21.88. 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

21.89. 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
    Dim i%
    i% = 4
    Print "The current hour is " & Hour( Now )
    Print Hour(TimeSerial(14,08,12))
    Print Hour(TimeValue("14:08:12"))
End Sub

21.90. 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
    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

21.91. **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

21.92. **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 (\( x=True, y=False \)) then this statement is false. If you are large and not mugged (\( x=True, y=True \)) then this statement is true. If you are not large (\( x=False \)) then nothing can invalidate this statement.
<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x IMP y</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Warning**  As of 1.0.3.1, this is incorrect for (0 Imp 1 = 0)

**Warning**  Imp does not seem to work with constants

\[ vC = -1 : vD = 0 \]
Print -1 Imp 0 REM -1 (incorrect)
Print vC Imp vD REM 0 (correct)

**Tip**  Use (Not x Or y) rather than (x Imp y) because it is logically equivalent and returns the correct values. (2 Imp 1) should return -3!

**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

**21.93. 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 #FileNumber var1[, var2[, var3[,...]]]
```

**Parameter:**
- FileNumber: Number of the file from which data is to be read.
- var: Numeric or String variables into which the read data will be placed.

**21.94. 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
```
21.95. 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.

```vba
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”
21.96. **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.

21.97. **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
```
**21.98. 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

**21.99. 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 309.

**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
21.100. 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 314.

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 eLocale 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

21.101. 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"
End Sub
21.102. 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.

Example:
Sub ExampleIsNumeric
  Dim v1, v2, v3
  v1 = "abc" : v2 = "123" : v3 = 4
  Print IsNumeric(v1) ' False
  Print IsNumeric(v2) ' True
  Print IsNumeric(v3) ' True
  Print IsNumeric("123x")' False
End Sub

21.103. 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

21.104. **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

21.105. **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
21.106. **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 ExampleLbound
    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

21.107. **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: 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

21.108. **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

21.109. **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

21.110. **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

21.111. 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 #FileNumber As Integer, Var As String

Return value:
None

Parameter:
FileNumber: Number of the open file from which data is to be read.

var: The name of the variable used to store the result.

Example:

21.112. 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(FileNumber)
Return value:
Long

Parameter:
FileNumber: Numeric expression containing the file number of an open file.

Example:
??

21.113. 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(FileNumber)

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
21.114. 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: \( \log_n(x) = \frac{\log(x)}{\log(n)} \)

Syntax:
\( \text{Log}(\text{Number}) \)

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

21.115. 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 318.

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

21.116. 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"<" REM ">123456789"
    LSet sVar = sExpr
    Print "">"; sVar; "<" REM ">12345<"
End Sub
21.117. 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 + "" + Chr(13)
    MsgBox sOut
End Sub

21.118. 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
### 21.119. 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:**
```vbnet
Public iPub As Integer
Sub ExamplePublic
    iPub = 1
    Call CalledSub
End Sub
Sub CalledSub
    Print iPub  REM 1
End Sub
```

### 21.120. 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:**
```vbnet
Sub ExampleColor
    Dim iColor As Long
    iColor = RGB(255,10,128)
    MsgBox "The color " & iColor & " consists of:" & Chr(13) &_  
        "Red = " & Red(iColor) & Chr(13)&  
        "Green= " & Green(iColor) & Chr(13)&  
        "Blue= " & Blue(iColor) & Chr(13) , 64,"Colors"
End Sub
```
21.121. 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

21.122. Shell Function

Summary:
Start an external application. The window style of the started application may be optionally included with the following values:

<table>
<thead>
<tr>
<th>Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Focus is on the hidden program window.</td>
</tr>
<tr>
<td>1</td>
<td>Focus is on the program window in standard size.</td>
</tr>
<tr>
<td>2</td>
<td>Focus is on the minimized program window.</td>
</tr>
<tr>
<td>3</td>
<td>Focus is on the maximized program window.</td>
</tr>
<tr>
<td>4</td>
<td>Standard size program window, without focus.</td>
</tr>
<tr>
<td>6</td>
<td>Minimized program window, but focus remains on the active window.</td>
</tr>
<tr>
<td>10</td>
<td>Full-Screen display.</td>
</tr>
</tbody>
</table>

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\TSEFroWin\g32.exe", 2, "c:\Macro.txt")
    Print "I just returned and the returned 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 you need to 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

    ' loking 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

21.123. **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()) & ")"    ' (10, 20)
    Print "(" & LBound(a2()) & ")"    ' (10, 20)
    Print "(" & LBound(a2(),1) & ")"    ' (10, 20)
End Sub
21.124. **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

21.125. **URL Notation And Filenames**

21.125.1. **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:///”, change “:” to “|”, and replace “\” with “/”. If you want to insert the computer name or IP address, insert it between the second and third “/” characters as in “file://localhost/c\autoexec.bat/”.

21.125.2. **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”.

21.125.3. **Anchoring To The Home Directory On Unix**

Thanks to Andrew McMillan ([andrew@catalyst.net.nz](mailto:andrew@catalyst.net.nz)), who mentions that on a Unix/Linux system a person may anchor a pathname to a home directory as follows:

22. Other languages

22.1. 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 22.1: Simple C# example to access bookmarks.**

```csharp
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);

22.2. 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

22.2.1. 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.

### 22.2.2. **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](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 123.
23. Index

Abs 350
And 313p., 335, 349
ApplicationScriptLibraryContainer 89p.
Array 310, 312
Asc 351, 358
ATN 352
AVERAGE 128
Beep 352
Blue 352
Boolean 303
BottomLine 131
ByVal 315, 353
Call 353
case 322
CBool 354
Cbyte 354
CDate 355
CDateFromIso 355
CDateToIso 356
CDbl 356
Cell
  CellAddress 125
  CellAddressConversion 124p.
  CellBackColor 120
getFormula 119
getSpreadSheet 125
getString 119
getValue 119
  IsCellBackgroundTransparent 120
NumberFormat 120, 128
setFormula 120, 128
setString 120, 128
setValue 120
CellAddress
  Column 125
  Row 125
CharacterProperties
  CharFontName 180
  CharHeight 180, 206
  CharLocale 71, 181
  CharPosture 180p.
    FontSlant 181

CharUnderline 180p.
CharWeight 180p., 187, 191
  FontWeight 180p., 187, 191
FontSlant
  DONTKNOW 181
  ITALIC 181
  NONE 181
  OBLIQUE 181
  REVERSE_ITALIC 181
  REVERSE_OBLIQUE 181
FontUnderline
  BOLD 181
  BOLDDASH 181
  BOLDDASHDOT 181
  BOLDDASHDOTDOT 181
  BOLDDOTTED 181
  BOLDLONGDASH 181
  BOLDWAVE 181
  DASH 181
  DASHDOT 181
  DASHDOTDOT 181
  DONTKNOW 181
  DOTTED 181
  DOUBLE 181
  DOUBLEWAVE 181
  LONGDASH 181
  NONE 181
  SINGLE 181
  SMALLWAVE 181
  WAVE 181
FontWeight
  BLACK 181
  BOLD 181
  DONTKNOW 181
  LIGHT 181
  NORMAL 181
  SEMIBOLD 181
  SEMILIGHT 181
  THIN 181
  ULTRABOLD 181
  ULTRALIGHT 181
ChDir 357
ChDrive 357
Choose 358
Chr 351, 358
<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency</td>
<td>303, 308</td>
</tr>
<tr>
<td>CurrentController</td>
<td>29</td>
</tr>
<tr>
<td>Cursor</td>
<td></td>
</tr>
<tr>
<td>compareRegionEnds</td>
<td>156, 160, 162p., 166p., 170, 173pp., 189, 194</td>
</tr>
<tr>
<td>compareRegionStarts</td>
<td>156p., 169p., 176, 186, 194</td>
</tr>
<tr>
<td>CreateTextCursor</td>
<td>43, 60p., 71, 135, 158pp., 175, 179, 184p., 195p., 230</td>
</tr>
<tr>
<td>createTextCursorByRange</td>
<td>38, 53, 155pp., 170, 172, 175pp., 180pp., 186, 290</td>
</tr>
<tr>
<td>getStart()</td>
<td>180p.</td>
</tr>
<tr>
<td>goLeft</td>
<td>151, 157, 163p., 173p., 176</td>
</tr>
<tr>
<td>goRight</td>
<td>71, 151, 157, 160, 162p., 166p., 172pp., 189p., 194</td>
</tr>
<tr>
<td>gotoEnd</td>
<td>43, 60, 151, 158p., 175, 185</td>
</tr>
<tr>
<td>gotoEndOfParagraph</td>
<td>180, 184, 195p., 198</td>
</tr>
<tr>
<td>gotoEndOfParagraph()</td>
<td>180</td>
</tr>
<tr>
<td>gotoNextParagraph</td>
<td>71, 160, 163, 173, 185p., 195p., 198</td>
</tr>
<tr>
<td>gotoRange</td>
<td>160, 190, 206</td>
</tr>
<tr>
<td>GoToStart</td>
<td>71, 151, 158pp., 175, 179, 184p., 195p.</td>
</tr>
<tr>
<td>gotoStartOfParagraph()</td>
<td>180</td>
</tr>
<tr>
<td>Date</td>
<td>303, 313, 366</td>
</tr>
<tr>
<td>DateSerial</td>
<td>356, 367</td>
</tr>
<tr>
<td>DateValue</td>
<td>355p., 367</td>
</tr>
<tr>
<td>Day</td>
<td>368</td>
</tr>
<tr>
<td>DBG_Methods</td>
<td>15, 21</td>
</tr>
<tr>
<td>DBG_Properties</td>
<td>15, 21</td>
</tr>
<tr>
<td>DBG_SupportedInterfaces</td>
<td>15, 21</td>
</tr>
<tr>
<td>Debug</td>
<td></td>
</tr>
<tr>
<td>PrintdbgInfo</td>
<td>15</td>
</tr>
<tr>
<td>ShowArray</td>
<td>15</td>
</tr>
<tr>
<td>WritedbgInfo</td>
<td>15</td>
</tr>
<tr>
<td>Declare</td>
<td>368</td>
</tr>
<tr>
<td>DefBool</td>
<td>303, 369</td>
</tr>
<tr>
<td>DefDate</td>
<td>303, 370</td>
</tr>
<tr>
<td>DefDb1</td>
<td>303, 370</td>
</tr>
<tr>
<td>DefInt</td>
<td>303, 370</td>
</tr>
<tr>
<td>DefLng</td>
<td>303, 371</td>
</tr>
<tr>
<td>DefObj</td>
<td>303, 371</td>
</tr>
<tr>
<td>DefVar</td>
<td>303, 372</td>
</tr>
</tbody>
</table>
GetSolarVersion() 394
GetSystemTicks() 394
GlobalScope 394p.
GoSub 316, 321, 395, 398
GoTo 50, 316, 321p., 396, 398
gotoEndOfParagraph 194p.
gotoNextParagraph 194, 198
GraphicExportFilter 55, 94
GraphicObjectFillBitmap 45
GraphicObjectShape 45, 200
Green 396
GUI 1
hasLocation 36, 39, 54, 102
HasUnoInterfaces 397
Header
  HeaderOn 134
  HeaderShared 134
Hex 397
Hour 398
IDE 1
IDL 1
If 316, 398
IIf 314, 317, 399
Imp 335, 400
Input 401
Input# 401
InputBox 401
InStr 402
Int 51, 403
Integer 303, 307
Is 313
IsArray 12, 313, 403
IsCursorInLastPar 198
IsDate 313, 404
IsEmpty 12, 309, 313, 404
IsMissing 52, 313pp., 405
isModified 36, 39, 54
IsNull 12, 309, 313, 405
IsNumeric 313, 406
IsObject 13, 313, 406
IsPlugged 109
isReadOnly 36, 39, 54
IsUnoStruct 13, 313, 407
Kill 360, 377, 389, 407
LBound 310p., 408
LCase 408
Left 409
Len 29, 51, 409
Let 410
library 8
Line Input 410
LineDistance 131
loadComponentFromURL 37
Loc 410
Locale 71
  Country 181
  Language 181
Lof 411
Log 412
Long 303, 308
Loop 318
  Do 319
  Do Until 318
  Do While 318
  Loop Until 319
  Loop While 319
LSet 413
LTrim 414
Macro Author
  ADPSetWordCase 193
  Andrew Pitonyak
AccessModes 383
ADPWordCountCharCursor 166
ADPWordCountStrings 164
ADPWordCountWordCursor 167
CalcGroupingExample 133
ClearDefinedRange 120
CloseOpenDocument 397
ColumnNumberOfString 125
CreateSelectedTextIterator 159
DecimalFeetToChar 78
DisplayAllStyles 29
ExampleNewPage 184
ExampleShell 50
FileAttributeString 392
FindCreateNumberFormatStyle 51
ForNextExampleSort 318
GetLeftMostCursor 157
GetRightMostCursor 157
GetSomeObjInfo 13
Null 309, 313
NumberFormat 52
    addNew 52
    DATE 52
    FindCreateNumberFormatStyle 126, 182
    queryKey 52
Object 303p., 309
OLE 1
OleObjectFactory 50, 86
On Error 50
    Local 323
    On Error 323p.
    On Error GoTo 0 324
    On Error GoTo Label 324
    On Error Resume Next 324
    Resume 324
    Resume Label: 325
    Resume Next 324
On Local Error Goto 0 51
On N GoSub 322
On N GoTo 322
OOo 1
Open 51, 360, 377, 389
Option Base 310
Option Explicit 303
Optional 52, 313pp.
OR 313p., 335
outline numbering 210
Outlook 86
Outlook.Application 86
PageDescName 183
PageNumberOffset 183
ParaStyleName 195
PI 310
PRINT 51
printdbgInfo 3
Private 305, 414
PropertyValue 132
    Name 132
    Value 132
Public 305, 415
Red 415
ReDim 303, 305, 311, 313, 372
ReDim Preserve 311p.
ReDimExample 311
REM 303
Return 321
RSet 416
SDK 1
Select 66
    Case 319
    Case Else 319
    Is 319
    Select Case 319
    To 319
Selection
    Columns 129
    getRangeAddress 128
    Rows 129
    ServiceInfo 15
    SetExpression 63
    setModified 36, 39
Shell 416p.
SimpleCommandMail 85
SimpleSystemMail 85
Single 303
Sort 132
SortField 132
    Field 132
    SortAscending 132
SpreadsheetDocument
    BottomLine 131
    Column
        Columns 127p.
        getByIndex 127, 129
        getCount 128p.
        getName 127, 133
    Columns 127, 133
    Count 120
    CurrentSelection 127p.
    getByName 119p., 132
    getCellByPosition 119p., 128
    getCellRangeByName 132
    getCellRangeByPosition 131
    getCount 127
    getName 125
    getRangeAddress 127
    Row
createTextCursorByRange 181
createTextCursorByRange() 155pp., 180
CurrentController 180p.
findFirst 187
findNext 189
gGetCurrentSelection 156
gGetCurrentSelection() 154pp.
getByIndex() 155p.
getCount() 155
insertControlCharacter 172
insertString 181
Text 180
TextField
Annotation 183
DateTime 182
FieldMaster
attachTextFieldMaster 60
Drawing 58
IlIustration 58
Table 58
Text 58
User 58pp.
SheetName 135
URL 62
TextField.SetExpression 63
TextGraphicObject 93, 200
TextRange 151
TextSection 17, 213
Thanks
Alain Viret 20
Andreas Bregas 2, 310, 315
Andrew Brown 164, 168
Andrew McMillan 419
Antal Attila 417
Antoine Jarrige 108
Berend Cornelius 65, 269
Bernard Marcelly 133, 189, 320, 325
Birgit Kellner 188
C Robert Pearsall 2
Chris Clementson 139
Christian Erpelding 29
Christian Junker 2p., 36, 61
Dan Juliano 96

EndRow 127, 129
getCount 127pp.
Rows 128
StartRow 127, 129
Rows 127
setString 128
SpreadsheetDocument 119
SupportsService 119
TableBorder 131
StarDesktop 9
Static 305
StatusIndicator
start 29
Str 51
String 303p., 308
StyleFamilies 134
Styles
  CharacterStyles 29
  FrameStyles 29
  NumberingStyles 29
  PageStyles 29
  ParagraphStyles 29
Sub 7, 314
switch 319, 322
SystemShellExecute 101
table of contents 212
TableBorder 131
Tan 351
terminate 36
TextContent 200
TextCursor 151
goDown() 151
goLeft() 151
goRight() 151
gotoEnd() 151
gotoStart() 151
goUp() 151
IsCollapsed 151
IsCollapsed() 155p.
TextCursor 151
TextDocument
compareRegionEnds() 156
compareRegionStarts() 156p.
createReplaceDescriptor 188

431
<table>
<thead>
<tr>
<th>Name</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Daniel Juliano</td>
<td>86</td>
</tr>
<tr>
<td>Daniel Vogelheim</td>
<td>168</td>
</tr>
<tr>
<td>David Woody</td>
<td>131</td>
</tr>
<tr>
<td>Eric VanBuggenhaut</td>
<td>108</td>
</tr>
<tr>
<td>Erik Anderson</td>
<td>101</td>
</tr>
<tr>
<td>Frank Schönheit</td>
<td>261</td>
</tr>
<tr>
<td>Gerrit Jasper</td>
<td>140, 143</td>
</tr>
<tr>
<td>Giuseppe Castagno</td>
<td>205</td>
</tr>
<tr>
<td>Hal Vaughan</td>
<td>23</td>
</tr>
<tr>
<td>Hermann Kienlein</td>
<td>2p.</td>
</tr>
<tr>
<td>Hermann-Josef Beckers</td>
<td>2p.</td>
</tr>
<tr>
<td>Jean Hollis Weber</td>
<td>2</td>
</tr>
<tr>
<td>Jim Thompson</td>
<td>300</td>
</tr>
<tr>
<td>Jürgen Schmidt</td>
<td>89</td>
</tr>
<tr>
<td>Kelvin Eldridge</td>
<td>2</td>
</tr>
<tr>
<td>Laurent Godard</td>
<td>2, 91, 95, 97</td>
</tr>
<tr>
<td>Leston Buell</td>
<td>289</td>
</tr>
<tr>
<td>Marc Messeant</td>
<td>74, 186</td>
</tr>
<tr>
<td>Mathias Bauer</td>
<td>2, 23, 65, 301</td>
</tr>
<tr>
<td>Michelle Pitonyak</td>
<td>2</td>
</tr>
<tr>
<td>Mikhail Voitenko</td>
<td>40</td>
</tr>
<tr>
<td>Niklas Nebel</td>
<td>131</td>
</tr>
<tr>
<td>Oliver Brinzinger</td>
<td>134, 269</td>
</tr>
<tr>
<td>Oliver Brinzinger</td>
<td>67</td>
</tr>
<tr>
<td>Olivier Bietzer</td>
<td>76</td>
</tr>
<tr>
<td>Paolo Mantovani</td>
<td>100, 103, 124, 266, 291, 423</td>
</tr>
<tr>
<td>Paul Sobolik</td>
<td>30</td>
</tr>
<tr>
<td>Peter Biela</td>
<td>24</td>
</tr>
<tr>
<td>Rob Gray</td>
<td>136</td>
</tr>
<tr>
<td>Robert Black Eagle</td>
<td>2</td>
</tr>
<tr>
<td>Rodrigo V Nunes</td>
<td>59</td>
</tr>
<tr>
<td>Russ Phillips</td>
<td>87, 101</td>
</tr>
<tr>
<td>Ryan Nelson</td>
<td>66, 133</td>
</tr>
<tr>
<td>Sasa Kelecevic</td>
<td>2p., 29, 119p., 127p., 130, 132</td>
</tr>
<tr>
<td>Solveig Haugland</td>
<td>2</td>
</tr>
<tr>
<td>Stephan Wunderlich</td>
<td>135</td>
</tr>
<tr>
<td>Sunil Menon</td>
<td>90</td>
</tr>
<tr>
<td>Sven Jacobi</td>
<td>93</td>
</tr>
<tr>
<td>Tony Bloomfield</td>
<td>21</td>
</tr>
<tr>
<td>Vance Lankhaar</td>
<td>91</td>
</tr>
<tr>
<td>ThisComponent</td>
<td>9</td>
</tr>
<tr>
<td>TimeValue</td>
<td>355</td>
</tr>
<tr>
<td>TOC</td>
<td>212</td>
</tr>
</tbody>
</table>