Useful Font Information

For OpenOffice

By

Andrew Pitonyak

Last Modified

Wednesday, July 17, 2013 at 09:13:19 PM

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/AndrewFontMacro.odt

Contribution	Contributor	Contact	Copyright
Original author	Andrew Pitonyak	andrew@pitonyak.org	2005

# Table of Contents

Index of Listings

1 Font descriptors

While writing, it is common to use special formatting such as bold, and italic. Bold text is darker, and italic font is typically slanted roughly 12 degrees to the right. Although a program, such as OpenOffice.org, can automatically apply these changes to a font, it is typical for a font designer to provide specialized versions of the font that are optimized for specific formatting. A font that is simply skewed to the right is called oblique. A specially designed italic font typically changes the serrifs, especially on lower case letters such as i, and the font generally has a freer look.

On my Linux system at home, OpenOffice.org has six version of the "Bitstream Vera Sans Mono"; the variations include normal, italic, oblique, bold, bold-italic, and bold-oblique. In other words, there is a special version specially for each of the six combinations, even for bold. My Windows computer supports more fonts, but only has four versions of "Bitstream Vera Sans Mone"; the two italic versions are missing.

I wrote a macro that enumerates the fonts and writes the font descriptor into a Calc document (see Listing 1). The "Font Details" button will create the Calc document with a description of all of the fonts.

I print the first and last printable character, and the font slant. On my Windows computer using the OOo version 2.0 beta, the values are 32, 65533, an 0 for every font.

Font Details
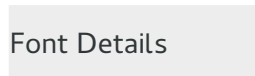
Listing 1. Write the font descriptors into a Calc document.

```
Sub FontInfoToCalc()
  Dim sNewUrl$        'URL required to create anew Calc document.
  Dim oDoc            'Newly created Calc document.
  Dim oFonts()        'Array of font descriptors supported by OOo.
  Dim oFontFamily()   'com.sun.star.awt.FontFamily constant group strings.
  Dim oCharSet()      'com.sun.star.awt.CharSet constant group strings.
  Dim oFontPitch()    'com.sun.star.awt.FontPitch constant group strings.
  Dim oFontSlant()    'com.sun.star.awt.FontSlant constant group strings.
  Dim oFontUnderline() 'com.sun.star.awt.FontUnderline constant group strings.
  Dim oFontStrikeout() 'com.sun.star.awt.FontStrikeout constant group strings.
  Dim i As Long       'General index variable.
  Dim oData()         'Data to store in the Calc document using setDataArray().
  Dim x               'Single font descriptor.
  Dim oMetric         'Individual font metric.
  Dim oWindow         'Container window AWT XDevice interface.

  REM Create a new Calc document and obtain the supported font descriptors.
  sNewUrl  = "private:factory/scalc"
  oDoc     = StarDesktop.loadComponentFromURL(sNewUrl,"_blank", 0, Array())
  oFonts() = SortedFontDescriptors(oDoc, True)
  oWindow  = oDoc.getCurrentController().getFrame().getContainerWindow()
```

```
REM Initialize the oData(0) array with appropriate header values.
oData()  = DimArray(UBound(oFonts()) + 1)
oData(0) = Array("Name", "StyleName", "Family", "CharSet", "Type", _
     "CharacterWidth", "Width", "Pitch", "Height", "Kerning", _
     "Orientation", "Slant", "Strikeout", "Weight", "Underline", _
     "WordLineMode", "First Char", "Last Char", "Slant Degrees")
Inspect(oWindow.getFont(oFonts(0)).getFontMetric())

REM Get data for each font descriptor.
REM by converting the data to a string using CStr, a new string instance
REM is returned. If I do not do this, then there are errors in SafeArray().
REM Also, False is not valid for use in setDataArray(), but a string is.
For i = 0 To UBound(oFonts())
  x = oFonts(i)
  oMetric = oWindow.getFont(x).getFontMetric()
  oData(i+1) = Array(      CStr(x.Name),        CStr(x.StyleName), _
    CStr(x.Family),        CStr(x.CharSet),   CStr(x.Type), _
    CStr(x.CharacterWidth), CStr(x.Width),     CStr(x.Pitch), _
    CStr(x.Height),        CStr(x.Kerning),   CStr(x.Orientation), _
    CStr(x.Slant),         CStr(x.Strikeout), CStr(x.Weight), _
    CStr(x.Underline),     CStr(x.WordLineMode), _
    ASC(CStr(oMetric.FirstChar)),  ASC(CStr(oMetric.LastChar)), _
    CStr(oMetric.Slant) )
Next

REM Initialize arrays with a text representation of the contant groups
REM used in the font descriptors.
oFontFamily() = Array("DONTKNOW", "DECORATIVE", "MODERN", _
                    "ROMAN", "SCRIPT", "SWISS", "SYSTEM")

oCharSet() = Array("DONTKNOW", "ANSI", "MAC", "IBMPC_437", _
            "IBMPC_850", "IBMPC_860", "IBMPC_861", "IBMPC_863", _
            "IBMPC_865", "SYSTEM", "SYMBOL")

oFontPitch() = Array("DONTKNOW", "FIXED", "VARIABLE")

oFontSlant() = Array("NONE", "OBLIQUE", "ITALIC", "DONTKNOW", _
                    "REVERSE_OBLIQUE", "REVERSE_ITALIC")

oFontUnderline() = Array("NONE", "SINGLE", "DOUBLE", "DOTTED", _
                    "DONTKNOW", "DASH", "LONGDASH", "DASHDOT", "DASHDOTDOT", _
                    "SMALLWAVE", "WAVE", "DOUBLEWAVE", "BOLD", "BOLDDOTTED", _
                    "BOLDDASH", "BOLDLONGDASH", "BOLDDASHDOT", _
                    "BOLDDASHDOTDOT", "BOLDWAVE")
```

```
oFontStrikeout() = Array("NONE", "SINGLE", "DOUBLE", "DONTKNOW", _
                         "BOLD", "SLASH", "X")

REM Convert the columns that use integers based on a constant group
REM to a string.
SafeArrayColumns(2,  oData(), oFontFamily(),    True)
SafeArrayColumns(3,  oData(), oCharSet(),       True)
SafeArrayColumns(7,  oData(), oFontPitch(),     True)
SafeArrayColumns(11, oData(), oFontSlant(),     True)
SafeArrayColumns(12, oData(), oFontStrikeout(), True)
SafeArrayColumns(14, oData(), oFontUnderline(), True)

REM Store the data into the Calc document.
AppendDataToCalcDoc(oDoc, oData())
End Sub
```

2 Formatted font summary

To help in viewing the fonts supported by the current system, I wrote a macro that creates a summary sheet of the supported fonts (see Listing 2). Use this to obtain a summary of the supported fonts with an example of how the fonts are displayed.

Font Summary

Listing 2. Write fonts, with examples, into a Write document.

```
Sub FontSummary(oDoc)
  Dim oFonts()      'Array of awt FontDescriptor structures.
  Dim sFontName$    'Name of one font.
  Dim sLastFont$    'Last used font.
  Dim oText         'Document's primary text object.
  Dim oCurs         'Text cursor used to insert the text.
  Dim nSaveHeight   'Initial character height.
  Dim sSaveFont$    'Initial font in effect.
  Dim i%            'General index variable.
  Dim s$            'Contains the sample text that is displayed using the font.

  s = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
  oDoc.lockControllers()
  On Error Goto EndLabel
  oText   = oDoc.getText()
  oCurs   = oText.createTextCursorByRange(oText.getEnd())
  sSaveFont= oCurs.charFontName
  oFonts() = SortedFontDescriptors(oDoc, False)
  oText.insertControlCharacter(oCurs, _
       com.sun.star.text.ControlCharacter.LINE_BREAK, False)

  nSaveHeight = oCurs.CharHeight
```

```
        oCurs.CharHeight = 10

    REM for each font descriptor.
    For i = LBound(oFonts()) to UBound(oFonts())
      sFontName = oFonts(i).Name
      If sFontName <> sLastFont Then
        sLastFont = sFontName

        oText.insertString(oCurs, sFontName & ": ", FALSE)
        oCurs.charFontName = sFontName
        oText.insertString(oCurs, s, FALSE)
        oCurs.charFontName = sSaveFont
        oText.insertControlCharacter(oCurs, _
            com.sun.star.text.ControlCharacter.LINE_BREAK, False)
        oText.insertControlCharacter(oCurs, _
            com.sun.star.text.ControlCharacter.LINE_BREAK, False)
      End If
    Next
    oCurs.CharHeight = nSaveHeight
    EndLabel:
    oDoc.unlockControllers()
End Sub
```

3 Font characters

I wanted to view the characters supported by a specific font. The macro in Listing 3 generates a table of the characters 32 through 255 for every font supported by the system. I could not figure out how to find the actual printable characters to limit the size of the tables and to see all of the data. This macro takes a while to run. On one Windows computer that I tried, the macro generated 67 pages of output.

The macro creates a paragraph style that keeps the table headings together with the tables below.

Font Chars

Listing 3. *Write fonts into tables.*
```
    Sub FontsInTable(oDoc)
    Dim oFonts()      'Array of awt FontDescriptor structures.
    Dim i%            'General index variable.
    Dim nRow%         'General index variable.
    Dim nCol%         'General index variable.
    Dim sFontName$    'Name of one font.
    Dim oText         'Document text object, used to insert text objects.
    Dim oCurs         'Cursor used to insert text content.
    Dim oTableCurs    'Used to format cells in the inserted tables.
    Dim sTData(16)    'Data used to set populate the inserted tables.
```

```
Dim sChars(16)      'Temporary used for individual rows added to sTData().
Dim x               'Temporary utility variable.
Dim oTable          'Table inserted into the document.
Dim oStyles         'All of the paragraph styles.
Dim oStyle          'The created paragraph style.

REM The visible characters are generally from 32 to 65533.
REM I do not know how to easily figure out which characters are supported,
REM so I will simply display the charcters from 32 to 255.
REM Start with the heading data.
sChars() = Array("",  "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", _
                 "A", "B", "C", "D", "E", "F")
sTData(0) = sChars()

REM I want the table listings to ALWAYS stay with the table below it.
REM Create a style for this. Also, set some space between the paragraph
REM and the table.
oStyles = oDoc.getStyleFamilies().getByName("ParagraphStyles")
If NOT oStyles.hasByName("_clisting") Then
  oStyle = oDoc.createInstance("com.sun.star.style.ParagraphStyle")
  oStyle.setParentStyle("Default")
  oStyle.ParaKeepTogether = True
  oStyle.ParaBottomMargin = 200
  oStyles.insertByName("_clisting", oStyle)
End If

REM i contains the ASCII value for the character.
i = 0
For nRow = 1 To 16
  x = DimArray(16)
  x(0) = sChars(nRow)
  For nCol = 1 To 16
    If i > 31 Then
      x(nCol) = CHR$(i)
    Else
      x(nCol) = ""
    End If
    i = i + 1
  Next
  sTData(nRow) = x
Next

oDoc.lockControllers()
oText = oDoc.getText()
oCurs = oText.createTextCursorByRange(oText.getEnd())
oFonts() = SortedFontDescriptors(oDoc, False)
oText.insertControlCharacter(oCurs, _
```

5

```
        com.sun.star.text.ControlCharacter.LINE_BREAK, False)

For i = LBound(oFonts()) to UBound(oFonts())
  'If i > 7 Then Exit For
  REM We only obtained unique font names, so no need to check for that here.
  sFontName = oFonts(i).Name
  oCurs.CharWeight = com.sun.star.awt.FontWeight.BOLD
  oText.insertString(oCurs, "Table ", FALSE)
  Call AddTableExpressionField(oDoc, oCurs)
  oText.insertString(oCurs, ".", FALSE)
  oCurs.CharWeight = com.sun.star.awt.FontWeight.NORMAL
  oText.insertString(oCurs, " Font: "" & sFontName & """, FALSE)

  oTable = oDoc.createInstance("com.sun.star.text.TextTable")
  oTable.initialize(17, 17)
  oText.insertTextContent(oCurs, oTable, False)
  oText.insertControlCharacter(oCurs, _
        com.sun.star.text.ControlCharacter.LINE_BREAK, False)
  oTable.setDataArray(sTData())
  oTableCurs = oTable.createCursorByCellName("A2")
  oTableCurs.goRight(16, True)
  oTableCurs.CharHeight = 10
  oTableCurs.gotoCellByName("A2", False)
  oTableCurs.goDown(15, True)
  oTableCurs.CharHeight = 8
  oTableCurs.CharWeight = com.sun.star.awt.FontWeight.BOLD

  oTableCurs.gotoCellByName("B2", False)
  oTableCurs.gotoEnd(True)
  oTableCurs.CharHeight = 8
  oTableCurs.CharFontName = sFontName
Next

Dim oDescriptor 'The search descriptor
Dim oFound 'The found range
Dim oFoundAll 'The found range
oDescriptor = oDoc.createSearchDescriptor()
oDescriptor.SearchString = "Table"
oDescriptor.SearchCaseSensitive = True
oDescriptor.SearchWords = True
oFoundAll = oDoc.findAll(oDescriptor)
For i = 0 to oFoundAll.getCount()-1
  oFound = oFoundAll.getByIndex(i)
  oFound.ParaStyleName = "_clisting"
  If (i > 0) AND (i MOD 2 = 0) Then
    oFound.PageDescName = oFound.PageStyleName
  End If
```

```
    Next

TheEnd:
  oDoc.unlockControllers()
End Sub
```

4 Sorting the font descriptors

The macro in Listing 4 demonstrates how to obtain the list of font descriptors and then sort them.

Listing 4. Return a sorted list of font descriptors.

```
Function SortedFontDescriptors(oDoc, bDuplicates As Boolean)
  Dim oWindow          'Container window AWT XDevice interface.
  Dim oFonts()         'Array of awt FontDescriptor structures.
  Dim oFinalFonts()    'Font descriptors to return.
  Dim nIndex() As Long 'Indirect sorted index into.
  Dim sName$           'Font name.
  Dim i&               'General index variable.
  Dim j&               'General index variable.
  Dim n&               'General utility variable.
  Dim k&               'General utility variable.
  Dim bChanged As Boolean

  oWindow  = oDoc.getCurrentController().getFrame().getContainerWindow()
  oFonts() = oWindow.getFontDescriptors()
  ReDim nIndex(UBound(oFonts()))
  n = UBound(oFonts())
  REM First, obtain an index into the unique values.
  For i = 0 to n
    nIndex(i) = i
  Next

  REM Sort the data!
  For i = 0 To n-1
    bChanged = False
    For j = 1 To n-i
      If oFonts(nIndex(j-1)).Name > oFonts(nIndex(j)).Name Then
        k = nIndex(j-1)
        nIndex(j-1) = nIndex(j)
        nIndex(j) = k
        bChanged = True
      End If
    Next
    If NOT bChanged Then Exit For
  Next

  REM Copy the font descriptor.
```

7

```
      sName = ""
      oFinalFonts() = DimArray(n)
      j = -1
      For i = 0 To n
        If bDuplicates OR (sName <> oFonts(nIndex(i)).Name) Then
          j = j + 1
          oFinalFonts(j) = oFonts(nIndex(i))
          sName = oFinalFonts(j).Name
        End If
      Next
      REM Bug alert, if I do NOT perform a ReDim Preserve, then the returned
      REM value is Empty; even if the array does not change in size.
      ReDim Preserve oFinalFonts(j)
      SortedFontDescriptors = oFinalFonts()
    End Function
```

5 Utility functions

This section contains miscellaneous macros that are not directly related to obtaining font information.

5.1 Add a "Table" caption field (SetExpression).

Notice that this document contains captions such as "Listing 4". The macro in Listing 5 creates and inserts a SetExpression text field that uses the "Table" master text field. You must set the number style to use. The number style is obtained from Listing 6.

Listing 5. *Insert a SetExpression text field, used to index table entries.*
```
    Sub AddTableExpressionField(oDoc, oCurs)
      Dim oField           ' This is the SetExpression field that is inserted.
      Dim oMasterField     ' The master field for the SetExpression field.
      Dim sMasterFieldName$ ' This is the name of the master field.
      Dim oText            ' The documents text object.

      oField = oDoc.createInstance("com.sun.star.text.TextField.SetExpression")
      oField.Content = "Table+1"
      oField.NumberFormat = FindNumberStyle("#,##0.00", oDoc)
      oField.NumberingType = com.sun.star.style.NumberingType.ARABIC
      sMasterFieldName = "com.sun.star.text.FieldMaster.SetExpression.Table"
      oMasterField = oDoc.getTextFieldMasters().getByName(sMasterFieldName)
      oField.attachTextFieldMaster(oMasterField)
      oText = oCurs.getText()
      oText.insertTextContent(oCurs, oField, False)
    End Sub
```

5.2 Find a number style

The first argument is the string representation of the numeric format. If the numeric format exists, then the corresponding key value is returned. If the format does not exist, then the numeric format is created and the key value is returned.

Listing 6. *Find a number format style, create it if required.*

```
Function FindNumberStyle(sFormat$, Optional doc, Optional locale)
  Dim oDoc       'The document to use.
  Dim oFormats   'The numerical formats object.
  Dim formatNum& 'The key of the number format to use.
  Dim aLocale As New com.sun.star.lang.Locale

  oDoc = IIf(IsMissing(doc), ThisComponent, doc)
  oFormats = oDoc.getNumberFormats()
  If ( Not IsMissing(locale)) Then
    aLocale = locale
  End If
  formatNum = oFormats.queryKey (sFormat, aLocale, TRUE)
  If (formatNum = -1) Then
    formatNum = oFormats.addNew(sFormat, aLocale)
    If (formatNum = -1) Then formatNum = 0
  End If
  FindNumberStyle = formatNum
End Function
```

5.3 convert constant group values to text

The FontPitch constant group contains the values DONTKNOW, FIXED, and VARIABLE. The values are represented as an integer, but this is difficult for a human to quickly understand. For example, an ideal usage is shown in Listing 7.

Listing 7. *Find a number format style, create it if required.*

```
Print  SafeArray(1, Array("DONTKNOW", "FIXED", "VARIABLE"), True)  'Prints 1 -
FIXED
```

Use the function in Listing 8 to create a text representation of numeric values. The first argument is the numeric value. The second argument is an array that contains the textual representation. The final argument, is optional and decides if the numeric value precedes the text value.

Listing 8. *Safely index an array to convert a constant to text.*

```
Function SafeArray(ByVal n As Long, x(), bIncN As Boolean)
  Dim s As String 'General string variable.

  If bIncN Then
    s = CStr(n) & " - "
  Else
```

```
      s = ""
    End If
    If LBound(x()) <= n AND n <= UBound(x()) Then
      SafeArray() = s & CStr(x(n))
    Else
      SafeArray() = s & "Invalid"
    End If
End Function
```

Use Listing 9 to apply SafeArray() to an entire column of data. The data is assumed to be an array that contains a single dimension array. For example, Array(Array(1, 2), Array(3, 4)). The column that is replaced, is assumed to contain a numeric value.

Listing 9. *Replace an entire column of constants with text.*

```
Sub SafeArrayColumns(ByVal nCol As Long, oData(), x(), bIncN As Boolean)
  Dim oRow()
  Dim i As Long
  Dim n As Long

  For i = LBound(oData()) + 1 To UBound(oData())
    oRow() = oData(i)
    If LBound(oRow()) <= nCol AND nCol <= UBound(oRow()) Then
      If IsNumeric(oRow(nCol)) Then
        oRow(nCol) = SafeArray(CLng(oRow(nCol)), x(), bIncN)
      End If
    End If
  Next i
End Sub
```