Bash Script for Evaluating Font Collections against one or more Languages

The script below should run on any contemporary Linux system. Copy the script into a new text file, name it FindFont (or whatever works for you). Placing it in a location that is already part of your $PATH will make life easier; be sure to set the execute flag (e.g. using “sudo chmod” or an equivalent command from a file manager GUI). Instructions are found in the early comments of the code itself as well as in the earlier part of this document.

```
#!/bin/bash
# FindFont - Find all Fonts containing one or more specified characters;
#            Frank Oberle แฟรงคโอเบอลท่าน: November 2016
#            This searches through each .ttf or .otf in some specified directories (see Where2Look below) to
#            find and list all fonts containing a defined set of characters. Several other attributes of each
#            "matching" font are listed as well.

# PURPOSE:   It is often useful to easily determine which fonts have support for one or more scripts and,
#            how correctly each of those fonts reports its support to an operating system or application.
#            If, for instance, it is necessary to combine Greek, Thai, and Hindi in a single document, it
#            would be ideal to locate which fonts support all of these in order to achieve some level of
#            "harmony." Unfortunately, even though many utilities exist to look within single fonts, I've
#            found none that would look through several at once. Furthermore, a significant number of fonts
#            don't correctly report which languages or scripts they provide support for (those mean quite
#            different things, but that's outside the scope of a shell script comments section). Hence, this
#            primitive, but useful shell script.
#            Fonts that don't correctly report thier contents and capabilities are often subject to being
#            unceremoniously replaced by word processors such as Libre/OpenOffice Writer and others.

# ALSO:      By default (but can be changed by setting variable values at the beginning), the script will
#            generate sparate text files: one containing a simple list of all matching fonts that report all
#            of their capabilities correctly, and another containing a list of fonts that may have structural
#            problems causing them to report their capabilities incorrectly, incompletely, or not at all.
#            This latter should be reviewed to determine if these fonts should be repaired or replaced.
#            This also generates an .fodt file (listing the "matching" fonts) that can be loaded into a word
# (optional) processor as the basis for a "font sample" document. Unfortunately, although many available
#            word processors can open and read .odt files, there are none I'm aware of that will permit all
#            of the fonts to be displayed correctly, making this a somewhat quixotic effort.
#            LibreOffice Writer, for instance, "helps" out by making apparently random substitutions of the
#            fonts when it encounters a "foreign" character set/language or whatever and, even worse, gives
#            no indication at all that it has done so. Combined with a slavish conformity to the rather odd
#            and illogical "Complex Text Layout" (CTL) definitions, creating such a "font sample" document
#            in such a word processor is far more of a bother than it ought to be. Nonetheless, if you have
#            a "publishing" application, the generated .fodt file may be useful as a starting point.

# DEPENDENCIES: The utility ttfdump, installed or available with most Linux distributions and Windows.
#               The utility fc-query, available for most Linux distributions and many Windows versions
#               A minimal understanding of the differences among languages, scripts, characters and glyphs;
#               one reason for this is so that you don't become confused by my blatant disregrd for those
#               distinctions in order to achieve my immediate goals !!
#               To add new Script or Language definitions to this script, some knowledge of how to construct
#               regular expressions is necessary. A pdf document was supplied with this script that explains
#               the layout of the targets the regular expressions are intended to match.
#               Finally, the Bash shell, of course. This script should work with any recent version of Linux
#               and may even work with Microsoft's new bash shell for Windows, since the other utilities
#               mentioned above are also available for Windows.

# USAGE: Right now, this is called as FindFont [1st script/language] [2nd script/language] [3rd ...] etc.
#        See the convertKeyword() routine below to define what "script/language" means; note that you may
#        need to add to this "case" statement to suit your own needs. Comments there will (maybe) explain
#        how. If no parameter is given, this will by default search for fonts containing Thai Unicode
#        characters; for most users, it probably makes more sense to simply have the script produce usage
#        instructions in such a case, but I did this for my own selfish purposes so it doesn't. It's easy
#        enough to change the "if [ $# == 0 ]; then" section in the MAIN SCRIPT DEFINITION ROUTINE below if
#        you wish to do so.
#        Currently recognized arguments are these: (case-insensitive, but require a minimum of 4 characters)
#        Arab[ic],  Arme[nian],  Bibl[ical (Hebrew)], Cyri[llic],  Deva[nagari], Fars[i],     Gree[k],
#        Hebr[ew],  Hind[i],     Iran[ian],           Laot[ian],   Pers[ian],    Russ[ian],   Thai,
#        Yidd[ish], Box Drawing, Curr[ency],          Domi[noes],  Frac[tions],  Liga[tures], Musi[c]

# BUGS: Test Characters and Test Sample Strings in Right-to-Left (RTL) scripts such as Arabic and Hebrew are
#       handled poorly when assigned to Bash variables. The order of characters in the search list is
#       reversed, and the order of the words in Sample strings is reversed, although the order of the
#       characters within the words is maintained. I lost patience attempting to figure out a work-around so
#       just be aware that it occurs. It really doesn't affect the purpose of this script as I use it.

# References:
#   Evaluating-fonts-for-multilingual-use.pdf                    # Document explaining this shell script
#   http://unicode.org/charts/                                   # View/download official Unicode charts;
#                                                                #   look up code points by number, etc.
#   My own rants:
```

```
#     https://bugs.documentfoundation.org/show_bug.cgi?id=92655     # Relevant pdf attachments on this link:
#                                                                    > "General discussion of Complex Text...
#                                                                    > "Detailed steps to reproduce the bugs.
#   A very nice additional rant from someone I've never met:
#     https://eev.ee/blog/2015/05/20/i-stared-into-the-fontconfig-and-the-fontconfig-stared-back-at-me/
#                                                                  #   look up code points by number, etc.
# Pan-Unicode Fonts: These are usually way too large to be of any practical use, but as a benchmark when you
#                    need to see something without worrying about whether the font contains a specific
#                    glyph, having one or two of these available can be helpful.
#   http://unifoundry.com/unifont.html                             # Font containing utilitarian (read:
#                                                                    ugly) representations of more Unicode
#                                                                    characters/glyphs than any other font.
#   www-sul.stanford.edu/depts/sysdept/info/CODE2000.TTF           # Code2000, 2001 & 2002: better looking
#                                                                  # and almost as comprehensive as Unifont.
#

####### OPENING: Check if ttfdump and fc-query are installed and, if not, exit with an appropriate message.
if ! ttfdumpExists=$(which ttfdump); then                          # If ttfdump utility is not installed
  echo "The ttfdump utility is required but can't be located."     # Display a warning message
  echo "If it's not installed, try running:"                       # Display a suggestion to the user
  echo "   sudo apt install ttfdump"
  echo " (or use whatever package command is appropriate for your distro, e.g. pacman, yum, etc.)"
  echo "Otherwise, check your path."
  exit                                                             # End the Script without going further
fi

if ! fcqueryExists=$(which fc-query); then                         # If fc-query utility is not installed
  echo "The fc-query utility is required but can't be located."    # Display a warning message
  echo "The fc-query utility is part of the fontconfig package"
  echo "Try running:"                                              # Display a suggestion to the user
  echo "   sudo apt install fontconfig"
  echo " (or use whatever package command is appropriate for your distro, e.g. pacman, yum, etc.)"
  echo "Otherwise, check your path."
  exit                                                             # End the Script without going further
fi

#if ! fcqueryExists=$(which Fontaine); then                         # If the Fontaine app is not installed
#  echo "The Fontaine application is not installed."               # Display an informational message
#  echo "Fontaine is not required for this script, but can be useful in analyzing font(s) of interest"
#  echo "without the need to use a full-blown font editor such as FontForge."
#  echo "If you are willing and able to compile it, the source code can be freely downloaded:"
#  echo "   For information, see  http://www.unifont.org/fontaine/ - OR - to download it directly"
#  echo "                  go to  https://sourceforge.net/projects/fontaine/files/latest/download"
#fi

#### VARIABLE DECLARATIONS    # The basics
           Origin=$(pwd)                                           # Save current directory so we can return
           debug='oFf'                                             # Set to 'ON' to debug certain sections
#           Where2Look=$(echo ~/.fonts)                            # Check only User-specific fonts
#           Where2Look=$(echo /usr/share/fonts/truetype)           # Check only for system fonts (all users)
           Where2Look=$(echo ~/.fonts /usr/share/fonts/truetype)   # Linux std locations; modify as needed
#           Where2Look=$(echo ~/Documents/Fonts_All)               # My own stash of uninstalled fonts
           Verbosity=1                                             # How much info to report (1, 2, 3)
                                                                   # Currently not implemented
           FODTGen=1                                               # Generate an .fodt file listing fonts
                                                                   # '1' turns it ON; anything else OFF
           FODTDOC="TestDoc"                                       # Created as $Origin/TestDoc.fodt in the
                                                                   #     directory where script was started
           FPassGen=1                                              # '1' creates a file listing 'good' fonts
           LLFN="PASS"                                             # LangListFileName name completed below
           SuspectGen=1                                            # '1' creates listing of 'suspect' fonts
           SFLFN="SuspiciousFonts.txt"                             # SuspiciousFontListFileName

# Not all of these need declaration in BASH, but just in case someone attempts to convert this to a real app
                           # Number of Arguments passed to this script on the command line
declare -i CMIdx                                                   # Int counter for number of arguments
declare -i NumArgsAccepted                                         # Int counter for upper # of args
           NumArgsAccepted=6                                       # ARBITRARY; this is all I ever use ...
declare -i ArgsFound                                               # Integer counter: Number of args passed
           ArgsFound=0                                             # ArgsFound Counter initialized to 0
                           # Number of Fonts examined
declare -i FontsChecked                                            # Int counter for # of fonts examined
           FontsChecked=0                                          # FontsChecked initialized to 0
declare -i FontsMatched                                            # Int counter for # of matching fonts
           FontsMatched=0                                          # FontsMatched initialized to 0
                           # Number of Language Codes examined
declare -i LangIdx                                                 # Pointer for Per-Font Language Arrays
declare -a LangAbbrevList                                          # Array of lang codes to be looked for
declare -a LangList                                                # Per-Font Array of found lang Keywords
declare -a LangsMatched                                            # Array of matching langs
declare -i LangMatchFailures                                       # Int counter for # of unmatchedlangs
           LangMatchFailures=0                                     # LangMatchFailures initialized to 0
declare -i FinalLangCount                                          # Integer value for counting langs found
                           # Number of Open Type Capability Matches and Failures
declare -i OTCapIdx                                                # Tracks OT capabilities in each font
```

```
declare -a OTFMatches                                           # Array of OTF Capabilty Matches
declare -a OTFMatchFailures                                     # Array of Open Type Tag
declare -i MissingOTFMatches                                    # No of Missing Open Type Capability Tags
          MissingOTFMatches=0                                   # Initialize Missing OTF Tags to 0
                            # Number of Character Map Matches and Failures
declare -a CMapsMatched                                         # Array of OTF Capabilty Matches
declare -i CMMatchSuccesses                                     # No of Character Map Match Successes
declare -i MissingCMMatches                                     # No of Missing Character Map Entries
          MissingCMMatches=0                                    # MissingCMMatches initialized to 0
declare -i CMapMatchFailures                                    # Int counter for # unmatched charsets
          CMapMatchFailures=0                                   # CMapMatchFailures initialized to 0

declare -i FullMatchListIdx                                     # Tracks full matches over all fonts
          FullMatchListIdx=0                                    # Int counter for # full matches
declare -i FullMatchFlag                                        # Tracks full matches over each font
          FullMatchFlag=1                                       # Assume success until a failure for each
declare -a FullMatchList                                        # List of fonts showing all requirements
                            # Cosmetic stuff for screen output
          MajorSeparator=$(printf "=%.0s" {1..128})             # For beginning and end of entire report
          MinorSeparator=$(printf "~%.0s" {1..128})             # For separating each font rpt section
          MiniSeparator=$(printf "~%.0s" {1..106})              # For separating each summary section
          # 36 chars right just; 4 digits right just; open string; line feed
          Fmt="%36s %-4s %s\n"                                  # For use with printf statements below

# Bash doesn't preprocess scripts, so functions like writeSample(), convertKeyword(), and inspectFont() must
# be defined before they are called ...

# writeSample() writes a sample line/section for each font found to contain the specified character(s) to the
# fodt output file which will serve as the basis for creating a word processor font sample document.
# It assumes that the file has been opened; any other parts of the file are written in line below. This
# function is only called in statements controlled by the value of the $FODTGen variable.
writeSample()                                                   # Only required for .fodt creation
  {
  echo -e "   <text:p >$1</text:p>"                  >> $DemoDoc   # Add this file to our output fodt
  # Note the no-break spaces (0x00a0) after <text:p > below; this is so LibreOffice doesn't discard them !
  echo -e "   <text:p >     $2</text:p>"             >> $DemoDoc   # List actual characters to output fodt
  echo -e "   <text:p >     $3</text:p>"             >> $DemoDoc   # Add the font Slant, Weight and Width
  echo -e "   <text:p >     Sample Text:$4</text:p>" >> $DemoDoc   # Add sample text to our output fodt
  echo -e "   <text:p/>"                             >> $DemoDoc   # Add a blank line after each font name
  }                                                             # White space ignored by LO-Writer

# convertKeyWord() interprets a processed (uppercase & trimmed) KeyWord to create various required values ...
# Here we can define some scripts of interest; in this context the Script name is used as the variable name,
#      but we could just as easily give the variables Language names if that makes more sense in context.
#      This is really cheating, since we're only looking for representative character(s) from particular
#      Script(s) - which can be misleading, as many Greek characters are present for use with Mathematics
#      even when full Greek language support isn't present. See comment under "CYRL" in the case statement.
# CASE Statement: For testing I've used arbitrary 4 letter abbreviations; this could probably be refined to
#                 use ISO 639 two (639-1) or three (639-2) character language codes for convenience, although
#                 we're really looking for a particular Script here rather than a particular Language. For
#                 quick and dirty purposes, this will suffice for now. (Cyrillic, for instance, is not a
#                 Language, but is a Script used by several Languages, each of which may have its own ISO 639
#                 language code.)  See "MAIN SCRIPT DEFINITION ROUTINE" below.
# HexCode: These are the hex codes in 0x0000 format representing Unicode values of representative sample
#          characters that we will search for. This will give a somewhat independent view of what Script(s)
#          each font contains.
# TestChar: These are the actual Unicode glyphs assigned to the $HexCode values above: There are no checks to
#           see that these actually match those glyphs, so be warned!
# CharMap: A bitmap is contained in each font where each bit represents one possible position defined by the
#          Unicode Standard (http://unicode.org/charts/). A "1" bit means the character is present while a
#          "0" indicates that it isn't. The output from fc-query is arranged in rows of eight (8) thirty-two
#          bit words arranged in four bytes each. These bytes (0x00-0xff) do not represent values but simply
#          positions, so are interpreted differently than you might expect. At the start of each row is an
#          offset value: if, for example, that value is "000e:" then the bits in that row indicate the
#          presence or absence of Unicode positions 0x0e00 through 0x0eff. Note that if no bits in the range
#          of a particular offset value are set, that row is simply not included in the output. Typically,
#          a row defines the presence of assignments from one to three or more Unicode Planes.
#          $CharMap is a regular expression to determine if appropriate matching lines exist.
#          Examples of how these are formed are given in comments at the end if I remember to include them.
# Lang: This is an entirely arbitrary designator that I use for my own convenience; in some cases it isn't
#       even a language at all. Neither "Cyrillic" nor "Devanagari" for instance are Languages, but Scripts;
#       and "Ligatures" and "Box Drawing" certainly aren't Languages either. It's just a mnemonic for me.
# LangCode: RFC-3066 is the source for the Lang(uage) Codes used below and by the Linux fc-query utility; for
#           sample listings and values, see https://www.w3.org/International/articles/language-tags/
#           For Region & Language Codes, see: http://www.i18nguy.com/unicode/language-identifiers.html
#           For Language Tags: https://www.microsoft.com/typography/otspec/languagetags.htm
#                          and: https://www.microsoft.com/typography/otfntdev/standot/features.aspx
# ScriptTag: Part of "capability:" section as reported for a fonts when using fc-query
#            ISO 15924: 4 char Alpha Script Codes: http://www.unicode.org/iso15924/iso15924-codes.html
#                I am using: otlayout:arab  otlayout:cyrl  otlayout:dev2  otlayout:deva otlayout:grek
#                            otlayout:hebr  otlayout:musc  otlayout:thai  (Only the last four letters!)
#            ISO 15924: 3 digit Script Codes: http://www.unicode.org/iso15924/iso15924-num.html
#            See a list at: https://www.microsoft.com/typography/otspec/scripttags.htm
#            Because the definitions of OFF/OT script tags predate ISO 15924 and Unicode Script property
#            assignments, the script tags provided by the fonts don't always conform to ISO 15924. The
```

```
#            resolution of conflicting proposals also resulted in alternate tags that essentially refer to
#            the same Unicode script definitions: for example, 'deva' and 'dev2' are virtually the same.
#            Script Tags supposedly indicate the font's ability to properly arrange characters that are
#            formed from more than one glyph*: a Thai character that needs to have both a vowel and a tone
#            mark above it; such placement needs to be altered if only one of those is required. It is very
#            important to remember that that - even if the font reports this ability for a certain script,
#            it doesn't imply that it does this rearrangment very well - but that's another issue.
#            * including: composition, decomposition, substitution, smallcaps, alternates, ligatures, et al.
# Sample: A sample word or phrase in characters of the Script/Language we are examing; this is used to
#         demonstrate certain capabilities if applicable; otherwise it's just that: a sample of the script.
#
# ISO 15295, which gives both 4 letter and numeric codes, is certainly more appropriate for this utility,
#    but the likelihood of a typical user knowing these is rather low, so I didn't attempt to do that.
#    To make things more interesting, many Unicode planes contain glyphs that are not really part of any
#    spoken language; there are no ISO 15295 script codes for Box Drawing characters, Emoji, Musical Symbols
#    and similar. So modify this listing to suit whatever identities you wish; just remember to also modify
#    the Keyword input translation sequences in the next section to suit what you are using.
convertKeyWord()
  {
   case "$1" in                                                    # Evaluate 1st (only) arg passed in ...
    "ARAB" ) HexCode="0x0639 0x0633 0x0626"                        # Only chars from basic alphabet
             TestChar="ع س ئ "                                     # N.B. MUST USE NON-BREAKING SPACES!
             # The following pattern looks only for the basic (ISO 8859-6) Arabic alphabet which, although
             # insufficient for "real-world" use, is all that's needed for the purposes of this script.
             CharMap="0006:[[:space:]01-9a-f]\{11\}[7f][f]\{5\}[ef]"                        # 258/32/15/32/32
             Lang="Arabic"
             LangCode="ar"                                         # Arabic (ISO 639-1)
             # fc-match uses only 'ar': The following are regional language versions:
             # ar-LB ar-LY ar-MA ar-MR ar-OM ar-PS ar-QA ar-SA ar-SD ar-SO ar-SY ar-TD ar-TN ar-YE
             # ar-AE ar-BH ar-DZ ar-EG ar-IL ar-IN ar-IQ ar-JO ar-KW
             ScriptTag="arab"
             Sample="هو مكتوب لي النصي عينة في العربية"            # "My sample script is written in Arabic"
                                                                   # RTL Words are reversed when $assigned
             # What that means essentially is that on a terminal output, the RTL Words, although having their
             # letters arranged correctly from right to left, are themselves written left to right. In the
             # fodt file, however, they are shown correctly. I attempted to "fix" this in a number of ways,
             # e.g. by wrapping the Arabic between RLE (0x202b) or RLO (0x202e) and PDF (0x202c) codes (see
             # http://www.unicode.org/reports/tr9/) but gave up trying, since it really didn't affect the
             # purpose for which this script was intended. See comments in other Right-to-Left Scripts.
             # Arab/160: Arabic Script Unicode blocks: 0x0600-; 0x0750-; 0x08a0-; 0xfb50-; 0xfe70-
         ;;
    "ARME" ) HexCode="0x0580 0x0583 0x0587"                                                 # 258/31/31/16/30
             TestChar="ր փ և"                                      # N.B. MUST USE NON-BREAKING SPACES!
             CharMap="0005:[[:space:]01-9a-f]\{10\}fffe[[:space:]01-9a-f]\{5\}fe7[[:space:]f]\{13\}e"
             # Interestingly, of the 31 fonts on my system that contain the test characters ($TestChar above)
             # as well as the language code "hy" ($LangCode below) all but 1 match this pattern. The one that
             # doesn't match is DejaVuSans-ExtraLight.ttf, which is missing the 0x0559 character ("Armenian
             # modifier letter left half ring" to use the Unicode term), making the "fe7" portion of the
             # CharMap pattern "fc7" instead. All 21 of the other DejaVu fonts on my system have this glyph
             # but I haven't pursued why that might be, since I don't use Armenian. I've included Armenian
             # only because it shares an fc-query output row (0005:) with Hebrew, and Hebrew is one of the
             # examples in my pdf "Evaluating Fonts for use in Multi-Lingual Documents" which explains how
             # to interpret/filter these lines using grep.
             Lang="Armenian"
             LangCode="hy"                                         # Really! I don't know the origin of "hy"
             ScriptTag="armn"
             Sample="Ինչ եմ խոսում եմ հայերեն:"                    # "Do you speak Armenian?"
             # Armenian Script Unicode block: 0x0530-0x058f; Armenian Ligatures are 0xfb13-0xfb17
         ;;
    "BIBL" ) HexCode="0x05d0 0x05d3 0x05d8 0x05dd 0x05e9 0x05a3 0x05b3"                     # 258/6/4/6/6
             # This finds fonts with the Hebrew Alphabet AND Cantillation Marks 0x0591-0x05af (טעמי המקרא)
             # $TestChar does not include the cantillation marks referenced in $HexCode above because they
             # are very difficult to see without being "attached" to a "supporting" character. If such a
             # character is used (as in $Sample below), it confuses bash anyway as each is really two
             # characters. That's why the mismatch (7 hex codes and only 5 test characters)
             # Since the script only actually looks at the hex codes, this really makes no difference.
             TestChar="א ד ט ם ש";                                 # N.B. MUST USE NON-BREAKING SPACES!
                                                                   # RTL Chars are reversed when $assigned
             CharMap="0005:[[:space:]01-9a-f]\{37\}fffe[[:space:]01-9a-f]\{5\}ffff"  # Cosmetic concatenation
             CharMap=$CharMap"[[:space:]01-9a-f]\{14\}00[[01]\{1\}[[078f]\{1\}07ff"  #  for printing source
             # Regarding the [078f] portion of the pattern above: in addition to the alphabet, a value of:
             # : 7 (0 1 1 1) means only Yiddish Digraphs (0x5f0-0x5f2) are present, no add'l punctuation
             # : 8 (1 0 0 0) means only additional punctuation (0x5f3-0x5f4) is present
             # : 0 (0 0 0 0) means neither of the above is present
             # : f (1 1 1 1) means both Yiddish Digraphs as well as additional punctuation is present.
             Lang="Hebrew";
             # From the Jewish 'Shema Yisrael' 'אֶחָד' Prayer: "May his name be blessed forever and ever."
             # Sample="מי יתן שם להתברך לנצח נצחים שלו."   # With no markings
             Sample="בָּרוּךְ שֵׁם כְּבוֹד מַלְכוּתוֹ לְעוֹלָם וָעֶד."  # RTL Words are reversed when $assigned
             # Note: If the words are reversed here, they appear in the proper order on the screen and in
             #       the .fodt file, but the characters within each word are in reverse order. Because some
             #       characters are altered due to their display order, things can get really bizzare. Sigh!
             LangCode='he'                                         # Hebrew (ISO 639-1)
             LangCode='he'                                         # Hebrew (ISO 639-1)
             # Languages spoken in Israel: ar-IL (Arabic) en-IL (English) he (Hebrew) yi (Yiddish)
```

```
        ScriptTag="hebr"
        # Hebr/125: Hebrew Script Unicode blocks: 0x0590-0x05ff; 0xfb00-0xfb4f (Presentation forms)
        # 0591-05af (Cantillation Marks); 05b0-05c7 (Points and Punctuation));
        # 05d0-05ea (Actual alphabet) 05f0-05f4 (Yiddish digraphs q.v. and additional punctuation)
    ;;
"CYRI" ) HexCode="0x0411 0x0414 0x042f 0x0496"                                  # 258/83/83/64/83
        # Here, this essentially means "Russian" (which points to this case anyway); see note.
        TestChar="Б Д Я Ж"                                  # N.B. MUST USE NON-BREAKING SPACES!
        CharMap="0004:[[:space:]]ffff[[:space:]01-9a-f]\{5\}ffffffff[[:space:]01-9a-f]\{5\}ffff"
        # Note: Here I'm only looking for the basic Russian alphabetic characters. The "anything{5}"
        #       gaps eliminate checking for some Cyrillic extensions in the ranges from 0x0400-0x040f
        #       and 0x0450-0x045f (and, of course, beyond); if you care about these $CharMap will
        #       need to be modified accordingly.
        Lang="Cyrillic"
        LangCode='ru'                                       # Russian (ISO 639-1)
        # Note that "Cyrillic" is a script, not a language; here I am treating it as if it refers to
        #      the Russian language; for my own use, that makes things easier, but beware!!!
        # Other languages that use Cyrillic script:
        # az-Cyrl (Azerbaijani), ru-RU (Russian), sr-Cyrl (Serbian), uz-Cyrl (Uzbek)
        # Note that Serbian, for example, uses different glyphs for some of its characters - one reason
        #      this routine is not meant for "production" use.
        ScriptTag="cyrl"
        Sample="Доброе утро"                                # "Good Morning"
        # Cyrillic Script Unicode block: 0x0400-0x04ff
    ;;
"DEVA" ) HexCode="0x0919 0x0921 0x0935";                                                # 258/4/4/4/4
        # Here, this essentially means "Hindi" (which points to this case anyway); see note.
        TestChar="ङ ड व"                                    # N.B. MUST USE NON-BREAKING SPACES!
        CharMap="0009:[[:space:]01-9a-f]\{8\}"
        # Note:
        Lang="Devanagari"
        LangCode='hi'                                       # Hindi (ISO 639-1)
        # See the first note in the "CYRL" case; this is for my own convenience.
        # SOME other (of >120) languages that use Devanagari script:
        # kok (Konkani), mr (Marathi), ne (Nepali), pi (Pali), sd-IN (Sindhi) and, of course,
        # sa (classical Sanskrit)
        ScriptTag="deva"
        Sample="मेरे नमूना स्क्रिप्ट हिंदी में लिखा है"    # "My sample script (is) written in Hindi"
        # The Sample text is displayed correctly in the fodt output, but letters are not joined together
        # properly on the terminal display.
        # The # marking the comment is at character position 83, whereas in most other lines it is at
        # position 69; this is because the character count of the sample is higher than it appears due
        # to the glyph composition that takes place with this particular Hindi sequence.
        # Bash decomposes this into individual glyphs on my terminal screen, but the output is rendered
        # correctly on the .fodt output, or when copied 'as is' from the terminal to LibreOffice Writer
        # and other applications. I originally thought that was because none of the mono-spaced terminal
        # fonts on my system report support for ISO 15924 script 'deva' or for the ISO 639-1 language
        # code 'hi' (which none of them do). I later became convinced it may be because of the terminal
        # itself; if I set the terminal profile to use FreeSerif (a proportional spaced font that does
        # report the 15924 and 639-1 codes correctly, the decomposition persists. It is also evident
        # that the terminal in this case forces the variable width glyphs of FreeSerif into mono-spaced
        # cells (and looks awful in the process as would be expected). Compare this to Thai below.
        # Deva/317: Devanagari Script: Unicode blocks: 0x0900-0x097f; Extended block is 0xa8e0-0xa8ff
    ;;
"GREE" ) HexCode="0x1f00 0x1f01 0x1f0f 0x1fa0 0x1fa1 0x1faf"    #                  # 258/66/66/58/66
        TestChar="ά ὰ Ἇ ᾧ ᾠ Ὧ"                             # N.B. MUST USE NON-BREAKING SPACES!
        # All of the letters in the standard Greek Alphabet - even many that are identical to Latin
        #     characters, e.g. B, H, K, O, P, and Y - are used in Mathematics, so simply looking for
        #     a selection of Greek alphabetic characters won't really indicate support for the Greek
        #     language. When looking at my own font collection I found 66 fonts that contained all of
        #     the needed composite characters. All of them did contain the 'el' language code, but only
        #     58 of them had the 'grek' Script Tag. Hence, the use of Greek composite characters here.
        # The $CharMap below doesn't test for ALL extended Greek characters, but is sufficient...
        CharMap="001f:[[:space:]]3f3ffff[[:space:]01-9a-f]\{30\}ffff"    # incl: 1f00-1f15 and 1fa0-1fb3
        Lang="Greek"
        LangCode='el'                                       # Greek (`Ellenic) (ISO 639-1)
        # fc-match uses only 'el', but there are el-CY (Cyprus) and el-GR (Greece) codes as well.
        # Grek/200: Greek Script Unicode block: 0x0370-0x03ff
        #         Greek Extended Unicode block: 0x1f00-0x1fff  # MORE IMPORTANT FOR ACTUAL GREEK
        ScriptTag="grek"
        Sample="Καλημέρα, είπε ο Αριστοφάνης με Βάτραχοι του"  # "Good Morning, said Ar... to his Frogs"
    ;;
"HEBR" ) HexCode="0x05d0 0x05d3 0x05d8 0x05dd 0x05e9"                           # 258/34/32/22/34
        # Hebrew Script is used not only for modern and biblical Hebrew, but also for Yiddish, an
        #       entirely different spoken language (though mostly spoken by Jewish Europeans); to
        #       illustrate handling alternate languages using the same script, see YIDD below.
        # Note that the order of these characters is reversed from that of the Hex Codes: Hebrew is RTL!
        # On the screen output, they will be listed left-to-right however. (it's a bash thing)
        # This CharMap pattern finds ONLY THE BASIC Hebrew Alphabet (0x05d0-0x05ea) # See other CharMaps
        TestChar="ש ם ט ד א";                               # N.B. MUST USE NON-BREAKING SPACES!
                                                            # RTL Chars are reversed when $assigned
        CharMap="0005:[[:space:]01-9a-f]\{55\}ffff[[:space:]01-9a-f]\{10\}7ff"
        Lang="Hebrew";
        Sample="בוקר טוב";                                  # "Good Morning"
                                                            # RTL Words are reversed when $assigned
```

```
        LangCode='he'                                           # Hebrew (ISO 639-1)
        # Languages spoken in Israel: ar-IL (Arabic) en-IL (English) he (Hebrew) yi (Yiddish)
        ScriptTag="hebr"
        # Hebr/125: Hebrew Script Unicode blocks: 0x0590-0x05ff; 0xfb00-0xfb4f (Presentation forms)
        # 0591-05af (Cantillation Marks); 05b0-05c7 (Points and Punctuation));
        # 05d0-05ea (Actual alphabet) 05f0-05f4 (Yiddish digraphs q.v. and additional punctuation)
    ;;
"LAOO" ) HexCode="0x0eae 0x0ec3 0x0ed5";                                                # 258/9/9/9/9
        TestChar="ธ ใ ๕";                                      # N.B. MUST USE NON-BREAKING SPACES!
        CharMap="000e:[[:space:]01-9a-f]\{37\}fef02596[[:space:]]3bffecae[[:space:]]33ff3f5f"
        # CharMap is '000e:'37x(spaces, 0s or 1s), then pattern (including required spaces
        Lang="Laotian";
        LangCode='lo'                                           # Lao (ISO 639-1)
        ScriptTag="lao"                                         # Strictly speaking, this is "lao "
        Sample="ท้าบสามาดเว้าพาสาลาวอ?";
        # Laoo/356: Lao Script Unicode block: 0x0e81-0x0eff (sparse block as shown below)
        # e81-e82 e84 e87-e88 e8a e8d e94-e97 e99-e9f ea1-ea3 ea5 ea7 eaa-eab ead-eb9 ebb-ebd ec0-ec4
        # ec6 ec8-ecd ed0-ed9 edc-edd:  ede (Khmu Gaw) and edf Khmu Nyaw) exist but are seldom used.
    ;;
"PERS" ) HexCode="0x06af 0x0698 0x0686 0x067e"                                          # 258/18/16/18/18
        TestChar="گ ژ چ پ"                                      # N.B. MUST USE NON-BREAKING SPACES!
        # The following pattern, like that used for Arabic above, looks only for the basic (ISO 8859-6)
        # Arabic alphabet (qv), but adds 4 characters that appear only in Persian to narrow the search.
        CharMap="0006:[[:space:]01-9a-f]\{11\}[7f][f]\{5\}[ef]"
        CharMap=$CharMap"[[:space:]01-9a-f]\{10\}[4-7c-f]"      # concatenate Persian-only char 0x067e
        CharMap=$CharMap"[[:space:]01-9a-f]\{9\}[13579bdf]"     # concatenate Persian-only char 0x0698
        CharMap=$CharMap"[01-9a-f]\{4\}[4-7c-f]"                # concatenate Persian-only char 0x0686
        CharMap=$CharMap"[[:space:]01-9a-f]\{6\}[8-f]"          # concatenate Persian-only char 0x06af
        Lang="Persian"                                          # aka known informally as Farsi
        LangCode="fa"                                           # fa-IR (IR for Iranian Farsi)
        ScriptTag="arab"                                        # Uses Arabic script and
        Sample="اسکریپت نمونه من این است که به زبان فارسی نوشته" # "My sampl scrpt is writtn in Persian"
        # * Persian has four letters more than the Arabic alphabet: ژ, چ, پ, and گ.
        # Arab/160: Arabic Script Unicode blocks: 0x0600-; 0x0750-; 0x08a0-; 0xfb50-; 0xfe70-
    ;;
"THAI" ) HexCode="0x0e01 0x0e09 0x0e14 0x0e42 0x0e55";                                  # 258/65/62/65/65
        TestChar="ก ฉ ด โ ๕";                                  # N.B. MUST USE NON-BREAKING SPACES!
        CharMap="000e:[[:space:]]ffffffe[[:space:]]87ffffff[[:space:]]0fffffff"
        Lang="Thai";
        LangCode='th'                                           # Thai (ISO 639-1)
        # Thai/352: Thai Script Unicode block: 0x0e01-0x0e7f
        ScriptTag="thai"
        Sample="แฟรงคโอเบอลทานี้บ่ปี่ญ ฌ๚";                     # the final 4 check for glyph arrangement
        # The # marking the comment is at character position 78, whereas in most other lines it is at
        # position 69; this is because the character count of the sample is higher than it appears due
        # to the multi-glyph compositions that take place with this particular Thai sequence. Unlike
        # Hindi however (see above), this sample displays correctly on the terminal (as well as on the
        # various outputs) because all of the Thai vowels and tone marks used are "dead keys."
    ;;
"YIDD" ) HexCode="0x05d0 0x05d3 0x05d8 0x05f0 0x05f1"                                   # 258/34/32/22/34
        TestChar="א ד ט ו ױ";                                  # N.B. MUST USE NON-BREAKING SPACES!
        # Substituted 2 Yidish-only digraphs in the hex codes, but these are not displayed here.
        # The Yiddish Language is spoken in Israel and various European countries. For its alphabet
        # it uses Hebrew Script, but with the addition of specific Yiddish Digraphs (0x05f0-0x05f2).
        # Digraphs are two glyphs which remain separate glyphs, but are placed very close together.
        # Note that the order of these characters is reversed from that of the Hex: Yiddish is also RTL!
        # On the screen output, they will be listed left-to-right however (it's a bash thing)
        CharMap="0005:[[:space:]01-9a-f]\{55\}ffff[[:space:]01-9a-f]\{10\}7ff"              # 32/34/258
        # Regarding the [078f] portion of the pattern above: in addition to the alphabet, a value of:
        # : 7 (0 1 1 1) means only Yiddish Digraphs (0x5f0-0x5f2) are present, no add'l punctuation
        # : 8 (1 0 0 0) means only additional punctuation (0x5f3-0x5f4) is present
        # : 0 (0 0 0 0) means neither of the above is present
        # : f (1 1 1 1) means both Yiddish Digraphs as well as additional punctuation is present.
        Lang="Yiddish";
        # Yiddish "looks like Hebrew but doesn't sound like Hebrew" (Translation of $Sample below).
        Sample="קוקט ווי העברעיש אָבער טוט נישט געדונט ווי העברעיש"
        LangCode='yi'                                           # Yiddish (ISO 639-1)
        ScriptTag="hebr"
        # Hebr/125: Hebrew Script Unicode blocks: 0x0590-0x05ff; 0xfb00-0xfb4f (Presentation forms)
        # 05d0-05ea (Actual alphabet) 05f0-05f4 (Yiddish digraphs and additional punctuation)
    ;;
# The Keywords below don't represent any "official" category, but are merely things I've looked for:
"BOXD" ) HexCode="0x250c 0x2500 0x2518"                                                 # 258/94/_/_/91
        TestChar="┌ ─ ┘"                                       # N.B. MUST USE NON-BREAKING SPACES!
        CharMap="0025:[[:space:]01-9a-f]\{8\}"                  # TO DO: Incomplete: NEEDS FIXING
        Lang="Box Drawing"                                      # Non-standard name for output here
        LangCode='99'                                           # Language not relevant
        Sample="╔╤╤╗ ┌──┘";
        # Box Drawing Script Unicode block: 0x2500-0x257F
    ;;
"CURR" ) HexCode="0x20ac 0x20ad 0x20b9 0x20aa 0x20a9"                                   # 258/42/_/_/42
        # The currncy symbols in the lines above and below are: Euro, Kip, Rupee, Shekel, Yen, Won
        TestChar="€ ₭ ₹ ₪ ₩";                                  # N.B. £ (0xa3) and ¥ (0xa5) are "Latin"
        # Currency Plane occupies right half of word 5 and left half of word 6 in the 0020: row
        CharMap="0020:[[:space:]01-9a-f]\{50\}[[:space:]01-9a-f]\{9\}"  # Could permit all 0s: FIX THIS!
```

```
                # "0020: ffffffff fffffcff ffffffff fff3001f 001f7fff 03ffffff ffff0000 0001ffff"
                #  echo ${CMap:46:9} pulls out relevant part:    7fff 03ff
                #  These characters cannot ALL be 0 !!   # But this will do for the moment
            Lang="Currency";                                        # Non-standard name for output here
            LangCode='99'                                           # Used universally; language not relevant
            Sample="\$5 = ₩5,682.98 = €4.59 = ₪19.31 = 158,435﷼ etc."; # Note: "$" must be escaped in bash!
            # Currency Symbols Unicode block: 20a0-20cf; also in a variety of other scripts
            # http://www.xe.com/symbols.php and https://gist.github.com/bzerangue/5484121 shows collections.
            # 0x20a1 (₡ Costa Rica Colon), 0x20ac (€ {various} Euro), 0x00a3 (£ {various} Pound),
            # 0xfdfc (﷼ Iranian Real),    0x20aa (₪ Israeli Shekel), 0x00a5 (¥ Japanese Yen),
            # 0x20a9 (₩ Korean Won),       0x20ad (₭ Laotian Kip),    0x20b1 (₱ Philippine Peso),
            # 0x0e3f (฿ Thai Baht),        0x20b9 (₹ Indian Rupee - also see devanagari letter U+0930)
        ;;
    "DOMI" ) HexCode="0x1f053";                                     # Merely an example of Unicode values
            TestChar="🁓";                                          #   higher than 0xffff; see "MUSI" below
            CharMap="01f0:[[:space:]01-9a-f]\{8\}"                  #   for comments about that range.  258/0
            Lang="Dominoes";                                        # Non-standard name for output here
            LangCode='99'                                           # Language not relevant
            Sample="🁓🁢🁛";                                        # Only used for fodt generation
            # Domino Tiles Unicode block: 0x1f030-0x1f09f;  1f030-1f093 are used
        ;;
    "FRAC" ) HexCode="0x00bc 0x00bd 0x00be"                                                 # 258/215/_/_/215
            TestChar="¼ ½ ¾"                                        # N.B. MUST USE NON-BREAKING SPACES!
            CharMap="0000:[[:space:]01-9a-f]\{46\}[01-9a-f]"        # ANY, not all of the above 3 fractions
            Lang="Ligatures";                                       # Non-standard name for output here
            LangCode='99'                                           # Language not relevant
            Sample="One-quarter is ¼; one half is ½; three-quarters is ¾."
        ;;
    "LIGA" ) HexCode="0xfb00 0xfb01 0xfb02 0xfb03 0xfb04 0xfb05 0xfb06"                     # 258/20/_/_/20
            # Characters above and below are cherry-picked from the 0xfb00-0xfb4f Block
            TestChar="ff ﬁ ﬂ ffi ffl ﬅ ﬆ"                  # N.B. MUST USE NON-BREAKING SPACES!
            CharMap="00fb:[[:space:]01-9a-f]\{7\}7f"                # 20/258        # 0xfb00-0xfb4f Block
            # ALTERNATE GROUPINGS OF LIGATURES for when you just need to feel depressed ...
        #   HexCode="0x0c6"                                                             # 258/3/_/_/3
        #   TestChar="Æ";                                            # N.B. MUST USE NON-BREAKING SPACES!
        #   CharMap="0000:" # [[:space:]01-9a-f]\{7\}" # 7f"         # 0/3/258
      #      HexCode="0x0e6"                                                          # 258/10/_/_/10
      #      TestChar="æ";                                         # N.B. MUST USE NON-BREAKING SPACES!
      #      CharMap="0000:" # [[:space:]01-9a-f]\{7\}" # 7f"       # 2/10/258
     #       HexCode="0x152 0x153"                                                    # 258/9/_/_/9
     #       TestChar="Œ œ";                                        # N.B. MUST USE NON-BREAKING SPACES!
     #       CharMap="0015:" # [[:space:]01-9a-f]\{7\}" # 7f"       # 9/258
            Lang="Ligatures";                                       # Non-standard name for output here
            LangCode='99'                                           # Language not relevant
            Sample="effective or effective: efficiency or efficiency: ﬆupendous or stupendous";
            # Alphabetic Presentation Forms Unicode block: 0xfb00-0xfb4f
            # See: https://en.wikipedia.org/wiki/List_of_precomposed_Latin_characters_in_Unicode
            # C1 Controls and Latin-1 Supplement Unicode block: 0080-00ff! Not recommended by Unicode but...
            # Latin Ligatures, like a few other natural groupings, are scattered all over the place, so:
            # TO DO: CAN MULTIPLE HEX CODE GROUPS (0080 & fb00) BE SENT BELOW? NEED TO CHECK WHAT I DID...
        ;;
    "MUSI" ) # HexCode="0x1d106 1d10b 0x1d120 0x1d160";             # REPLACED: See next assignment line.
"           # TestChar="𝄆 𝄋 𝄞 𝅘𝅥𝅯"
            # The "official"* Musical Symbols Unicode block is nominally 0x1d100-1d1ff, with the segments
            #     1d100-1d126, 1d129-1d158, 1d15a-1d172, and 1d17b-1d1e8 being the actual characters
            #     * The "official" version was introduced in version 3.1 of Unicode (March 2001)
            # These characters are all present in both .ttf and .otf versions of FreeMono for example but,
            #     as with other scripts that begin beyond 0xffff, they are not reported by ttfdump or any
            #     other font utility I've been able to locate.
            # MuseScore, for example, uses their own MScore font, which has glyphs in a private use segment
            #     (e.g. 0xe19b) but that's not generally usable due to the proprietary encoding.
            # Therefore: use the limited set of Musical Symbols located in the Unicode Miscellaneous Symbols
            #     block that runs from 0x2600-0x26ff; the following are the applicable symbols for music.
            HexCode="0x2669 0x266a 0x266b 0x266c 0x266d 0x266e 0x266f"                  # 258/34/_/1/34
            #   ♩ 2669 quarter note             ♪ 266A eighth note        ♫ 266B beamed eighth notes
            #   ♬ 266C beamed sixteenth notes   ♭ 266D music flat sign    ♮ 266E music natural sign
            #   ♯ 266F music sharp sign
            TestChar="♩ ♪ ♫ ♬ ♭ ♮ ♯";                                # TestChar="𝄆 𝄋 𝄞 𝅘𝅥𝅯"; # TestChar="   ";
            CharMap="0026:[[:space:]01-9a-f]\{8\}"                  # Was: ="01d1:[[:space:]01-9a-f]\{8\}"
            Lang="Music";                                           # Non-standard name for output here
            LangCode='99'                                           # Language not relevant
            ScriptTag="musc"
"           Sample="𝄞♯𝅗𝅥𝅘𝅥𝅘𝅥𝅯𝅘𝅥𝅯"                                  # Only used for fodt generation
        ;;
        * ) HexCode="0x0041 0x0042 0x0079 0x007a";
            TestChar="A B y z";                                     # N.B. MUST USE NON-BREAKING SPACES!
            CharMap="0000:[[:space:]01-9a-f]\{8\}"
            Lang="English";
            LangCode='en'                                           # English (ISO 639-1)
            Sample="Good Morning";
            # C0 controls and Basic Latin Unicode block: 0x0000-0x007f (formerly called lower ASCII)
        ;;
  esac
  # Note that all variable definitions are GLOBAL (the default in Bash), so any caller has easy access.
}
```

```
### MAIN SCRIPT DEFINITION ROUTINE: Interprets parameters passed to this shell script, and calls the
# convertKeyWord() function to grab several values for each Script/Language we are interested in looking at.
# Here we define the particular scripts we are interested in; one to $NumArgsAccepted may be specified as
# command line parameters, but if none are given explicitly, we'll look for fonts containing Thai characters.
echo $MajorSeparator
if [ $# == 0 ]; then                                                # If TRUE could just show usage and exit
  # The default to Thai is for my own convenience; as currently written, up to $NumArgsAccepted parameters
  # can be given from the following supported (and case-insensitive) entries:
  #    Arab[ic],  Arme[nian],  Bibl[ical (Hebrew)], Cyri[llic],  Deva[nagari], Fars[i],     Gree[k],
  #    Hebr[ew],  Hind[i],     Iran[ian],           Laot[ian],   Pers[ian],    Russ[ian],   Thai,
  #    Yidd[ish], Box Drawing, Curr[ency],          Domi[noes],  Frac[tions],  Liga[tures], Musi[c]
  # Otherwise, for any unrecognized keyword, this will search for fonts containing Latin characters
  echo "INFO: No command line parameters given; we're looking for Thai characters by default"
  Keyword="THAI"
  convertKeyWord $Keyword                                           # Grab specific values for this language
  TestCodeList=$HexCode; LangCodeList=$LangCode; CMapList=$CharMap
  SampleText=$Sample                                                # This reverses word order in RTL phrases
  Message=$TestChar' ('$HexCode')'
  CharMsg="'"$TestChar"'"
  LangList[1]=$Keyword
  LangAbbrevList[1]=$LangCode
  OTFCapList=$ScriptTag
else
  for arg in `seq 1 $NumArgsAccepted`; do                           # Wander through each argument passed in
    if [ ${!arg} ]; then                                            # If any "arg"th argument was passed
      Keyword=$(echo ${!arg} |cut -c1-4 |tr '[:lower:]' '[:upper:]')  # Create 4 char upper case keyword
      # Unicode planes contain SCRIPTS, although any Script may be used by multiple languages.
      # These "translations" convert languages I commonly refer to into the Scripts they use.
      # This is NOT scalable, as some languages can be written in more than one script!
      #    For example: Azerbaijani, Japanese, Kyrgyzstani, Moldovan, Mongolian, and Turkmenistani: Beware!
      #    Inuktitut can be written in its own syllabary, a modified Cherokee alphabet or with Latin letters.
      if [ $Keyword == "HIND" ]; then Keyword="DEVA"; fi            # Convert Language to required Script
      if [ $Keyword == "RUSS" ]; then Keyword="CYRI"; fi            # Convert Language to required Script
      if [ $Keyword == "FARS" ]; then Keyword="PERS"; fi            # Make Farsi an alias for Persian
      if [ $Keyword == "IRAN" ]; then Keyword="PERS"; fi            # Make Iranian an alias for Persian
      if [ $Keyword == "LAOT" ]; then Keyword="LAOO"; fi            # Compensate for odd abbreviation
      convertKeyWord $Keyword                                       # Create variables for this argument
      TestCodeList=$TestCodeList" "$HexCode; LangCodeList=$LangCodeList" "$LangCode   # Expand lists
      CMapList=$CMapList" "$CharMap;         SampleText=$SampleText" "$Sample         # Expand lists
      ((ArgsFound++))
      if [ $ArgsFound == $# ] && [ $# != 1 ]; then                  #
        Message=$Message' and '$TestChar' ('$HexCode')'             # Need "and" for last entry but not first
        CharMsg=$CharMsg' and '"'"$TestChar"'"
      else
        Message=$Message' '$TestChar' ('$HexCode')'                 # Otherwise just separate with spaces
        CharMsg=$CharMsg' '"'"$TestChar"'"
      fi
      # In either case above, $Message has the embedded RTL Characters from $TestChar in a reversed order
      # I suspect this is a side effect of font rendering mechanisms interpreting spaces as "Latin"
      LangList[arg]=$Keyword
      LangAbbrevList[arg]=$LangCode
      OTFCapList=$OTFCapList" "$ScriptTag
    fi
  done                                                              # Done: for arg in `seq 1 $NumArgsAcce...
fi                                                                  # Done: if [ $# == 0 ]

### OPEN AN .fodt FILE (if the FODTGen flag is set) and create the initial part of the header
# Now we prepare to create a demonstration document that can be loaded into LibreOffice Writer or other
# application that can read .fodt files (bare xml versions of .odt files). We need to establish a full
# path name for the file, because we will be in different directories as we write to it, which can get ugly.
# TO DO: Include Font style information when creating the .fodt output ::: gave up; not recognized by LO
if [ $FODTGen == 1 ]; then
  DemoDoc=$Origin"/"$FODTDOC".fodt"
  # Experimental stuff: See /mnt/Library/Ubuntu/Unity_Screen_Elements.fodt for things to rip off...
  echo '<?xml version="1.0" encoding="UTF-8"?>' > $DemoDoc             # CREATE NEW FILE; then append below
  echo ''                                                                                  >> $DemoDoc
  echo '<office:document'                                                                  >> $DemoDoc
  echo ' xmlns:office="urn:oasis:names:tc:opendocument:xmlns:office:1.0"'                  >> $DemoDoc
  echo ' xmlns:style="urn:oasis:names:tc:opendocument:xmlns:style:1.0"'                    >> $DemoDoc
  echo ' xmlns:text="urn:oasis:names:tc:opendocument:xmlns:text:1.0"'                      >> $DemoDoc
  echo ' xmlns:table="urn:oasis:names:tc:opendocument:xmlns:table:1.0"'                    >> $DemoDoc
  echo ' xmlns:fo="urn:oasis:names:tc:opendocument:xmlns:xsl-fo-compatible:1.0"'           >> $DemoDoc
  echo ' xmlns:meta="urn:oasis:names:tc:opendocument:xmlns:meta:1.0"'                      >> $DemoDoc
  echo ' xmlns:number="urn:oasis:names:tc:opendocument:xmlns:datastyle:1.0"'               >> $DemoDoc
  echo ' xmlns:script="urn:oasis:names:tc:opendocument:xmlns:script:1.0"'                  >> $DemoDoc
  echo ' xmlns:loext="urn:org:documentfoundation:names:experimental:office:xmlns:loext:1.0"'  >> $DemoDoc
  echo ' xmlns:field="urn:openoffice:names:experimental:ooo-ms-interop:xmlns:field:1.0"'   >> $DemoDoc
  echo ' xmlns:formx="urn:openoffice:names:experimental:ooxml-odf-interop:xmlns:form:1.0"' >> $DemoDoc
  echo ' xmlns:css3t="http://www.w3.org/TR/css3-text/"'                                    >> $DemoDoc
  echo ' office:version="1.2"'                                                             >> $DemoDoc
  echo ' office:mimetype="application/vnd.oasis.opendocument.text">'                       >> $DemoDoc
  echo ' <office:styles>'                                                                  >> $DemoDoc
  echo ' </office:styles>'                                                                 >> $DemoDoc
  echo ' <office:body>'                                                                    >> $DemoDoc
```

```
  echo '  <office:text>'                                                                    >> $DemoDoc
  echo '   <text:p >Font Samples for selected fonts:</text:p>'                               >> $DemoDoc
  echo '   <text:p >This file is '$DemoDoc'</text:p>'                                        >> $DemoDoc
  echo '   <text:p/>'                                                                       >> $DemoDoc
  echo '   <text:p >Thai Script Sample: นถึกูเขยีนโดยแฟรงคโอเบอลี/text:p>'                  >> $DemoDoc
  echo '   <text:p >Devanagari Script Sample: यह अंग्रेजी भाषा नहीं है (Hindi Language)</text:p>'       >> $DemoDoc
  echo '   <text:p >Hebrew Script Sample: זו השפה העברית; N.B. Hebrew is right to left</text:p>' >> $DemoDoc
  echo '   <text:p/>'                                                                       >> $DemoDoc
  echo '   <text:p/>'                                                                       >> $DemoDoc
           # $Message below looks strange in the .fodt file if it contains an RTL character
           # but it is read and displayed correctly by LibreOffice Writer
  echo '   <text:p >The following is a list of fonts in the directories:</text:p>'          >> $DemoDoc
  echo '   <text:p >    '$Where2Look'</text:p>'                                             >> $DemoDoc
  echo '   <text:p >that contain the character(s) '$Message'</text:p>'                      >> $DemoDoc
  echo '   <text:p/>'                                                                       >> $DemoDoc
fi                                                                  # Done (momentarily): if [ $FODTGen ==...

# Generate a list of 'suspicious' fonts, i.e. those that may require replacement
if [ $SuspectGen == 1 ]; then                                       # Switch set at beginning of this script
  SFLFN=$Origin"/"$SFLFN
  printf "This file is: $SFLFN\n"                                                            > $SFLFN
  printf "This lists Font Files that may need repair or replacement due possible errors.\n"  >> $SFLFN
  printf "Note: When examining multiple Scripts/Languages, not all suspect fonts may appear.\n"  >> $SFLFN
  printf "$MiniSeparator\n\n"                                                                >> $SFLFN
  printf "The following directory tree(s) were examined: $Where2Look\n\n"                    >> $SFLFN
fi                                                                  # Done: if [ $SuspectGen == 1 ]

inspectFont()                                 # Lists info about each font containing the specified HexCode(s)
  {
   ((FontsChecked++))                                               # Increment number of fonts examined
   FullMatchFlag=1   # Assume a full match until proven otherwise
   #### This section looks in each font for one or more specific characters from a particular script:
   CSetMatch=$(fc-match $Location/$DirName$fontf charset)
   for HexCode2Find in $TestCodeList; do                            # Examine each hex code group in turn
     for OneCode in $HexCode2Find; do                               # Check each hex code in $TestCodeList
       if [ $debug == 'ON' ]; then printf "%40s" "Checking $OneCode in $fontf: "; fi
       # Use ttfdump to find out if this font contains the hex code sequences we're currently looking for.
       # SymLinks cause errors here, so send them to the bit bucket: I'm too lazy to extract all link info
       #    since only one link was installed by my OS as a fallback for Japanese, which I don't use. YMMV
       if TmpOut=$(ttfdump -t cmap $Location/$DirName$fontf 2>/dev/null | grep $HexCode2Find); then
         Success="Yes"                                              # Success contingent on entire loop
                                                 if [ $debug == 'ON' ]; then echo $OneCode " found ..."; fi
       else
         Success="No"                                               # Any "No" causes a failure for this font
                       if [ $debug == 'ON' ]; then echo $OneCode " NOT found: skipping to next font ..."; fi
         break 2                                                    # If ANY Code not found, exit both
       fi                                                           # Done: if TmpOut=$(ttfdump -t cmap ...
     done                                                           # Done: for OneCode in $HexCode2Find
   done                                                             # Done: for HexCode2Find in $TestCodeList
   if [ "$Success" = "Yes" ]; then                                  # If ALL HexCodes in TestCodeList found
     ((FontsMatched++))                                             # Increment Num of fonts w/all hex codes
     printf "$Fmt" $fontf "( located in:" $Location/$DirName" )"
     printf "%38s %s\n" " " "Potential match $FontsMatched of $FontsChecked Fonts checked so far... "
         # The echo below is used as an intermediary to remove leading spaces from fc-query output line
         FntSty=$(echo $(fc-query "$Location/$DirName$fontf" |  \
                                  grep "style"               |  \
                                  sed s/"style:"//           |  \
                                  sed s/"stylelang*"//)      |  \
                                  cut -c 1-72)                  # Trim the output for screen display
         FntSlt=$(fc-query "$Location/$DirName$fontf" | grep "slant"  | sed s/"slant:"//)  #
         FntWgt=$(fc-query "$Location/$DirName$fontf" | grep "weight" | sed s/"weight:"//) #
         FntWid=$(fc-query "$Location/$DirName$fontf" | grep "width"  | sed s/"width:"//)  #
     printf "%36s %s\n" " " "  Font Style begins: $FntSty"
     FntSltWgtWid=$(echo $FntSlt","$FntWgt", and"$FntWid);
     printf "%36s %36s %6s, %8s and %8s\n" " " "  Font Slant, Weight, and Width are:" \
                                          $FntSlt $FntWgt $FntWid

     if [ $FODTGen == 1 ]; then                                     # If an .fodt file was requested
       writeSample "$fontf contains the requested character(s) ..."   \
                   "$fontf is located in: $Location/$DirName"         \
                   "Font Slant, Weight, and Width are: $FntSltWgtWid" \
                   "$SampleText"                                    # Report this font in the output fodt
     fi                                                             # Done:if [ $FODTGen == 1 ]

     #### Now check the Language Support reported by this font to see if it's correct
     LangIdx=0                                                      # Language Code: Index for array
     for OneCode in $LangCodeList; do                               # Check each language to be reported
       ((LangIdx++))                                                # Increment lang code index
       PLFSwitch=0                                                  # PerLangFoundSwitch limits to 1 match
       if TmpOut=$(fc-query "$Location/$DirName$fontf" | grep "|$OneCode|") # Does fc-match find OneCode
       then
         printf "$Fmt" " " "√ fc-query correctly reports the ISO 639-1 Language Code: '$OneCode'"
         if [ $PLFSwitch == 0 ]; then                               # So we don't double count errors
           ((LangsMatched[LangIdx]++))
           printf "$Fmt" " " \
```

```
            "  ...match number ${LangsMatched[LangIdx]} for the ISO 639-1 Language Code '$OneCode'"
      ((PLFSwitch++))                                       # Could just be set to 1
    fi
  else
    # Don't print a negative result if this is a fake language (e.g. currency, music symbols, etc.)
    if [ $OneCode != '99' ]; then
       # echo -en $ErrColor # This works stand-alone but not in script, and it doesn't work with printf
       # ErrColor='\e[1;41;37m' #(Red on White) # # NmlColor='\e[27m'  # echo -en $ErrColor
      printf "$Fmt" ">>" "X fc-query FAILED TO REPORT the ISO 639-1 Language Code '$OneCode'"
      FullMatchFlag=0                                       # No Failures will be added to list
                        if [ $debug == 'ON' ]; then echo "FullMatchFlag set back to "$FullMatchFlag; fi
      if [ $SuspectGen == 1 ]; then                         # Switch set at beginning of this script
        printf "For '$OneCode':    $Location/$DirName$fontf "                                >> $SFLFN
        printf "FAILED TO REPORT this ISO 639-1 Language Code.\n"                            >> $SFLFN
      fi                                                    # Done: if [ $SuspectGen == 1 ]
    else
      printf "$Fmt" "   " "- Code '$OneCode' is not a language, so no language reporting was attempted."
    fi                                                      # Done: if [ $OneCode != '99' ]
    ((LangMatchFailures++))                                 # Increment Num fonts lacking lang code
  fi                                                        # Done: if TmpOut=$(fc-match...
done                                                        # Done: for OneCode in $LangCodeList

#### Check the Open Type Layout capability for this font (can be in both TrueType and OpenType fonts)
OTCapIdx=0                                                  # OpenType Capabilities: Index for array
for OneCap in $OTFCapList; do                               # Check OTF capability for each font
  ((OTCapIdx++))
  POCFSwitch=0                                              # PerOtfCapFoundSwitch limits to 1 match
  if TmpOut=$(fc-query "$Location/$DirName$fontf" | grep "capability:\(.*\)otlayout:$OneCap")
  then
    printf "$Fmt" " " "√ fc-query correctly reports ISO 15924 Script Support Code: '$OneCap'"
    if [ $POCFSwitch == 0 ]; then                           # So we don't double count matches
      ((OTFMatches[OTCapIdx]++))
      printf "$Fmt" " " \
             "  ...match number ${OTFMatches[OTCapIdx]} for the ISO 15924 Script Code '$OneCap'"
      ((POCFSwitch++))                                      # Could just be set to 1
    fi
  else
    printf "$Fmt" ">>" "X fc-query FAILED TO REPORT Script Support for ISO 15924 code: '$OneCap'"
    ((OTFMatchFailures[OTCapIdx]++))                        # Increment Num fonts lacking OTF Caps
    if [ $SuspectGen == 1 ]; then                           # Switch set at beginning of this script
      printf "For '$OneCap':  $Location/$DirName$fontf: "                                    >> $SFLFN
      printf "Font doesn't report Script Support for this ISO 15924 Code.\n"                 >> $SFLFN
    fi                                                      # Done: if [ $SuspectGen == 1 ]
    FullMatchFlag=0                                         # No Failures will be added to list
  fi                                                        # Done: if TmpOut=$(fc-match...
done                                                        # Done: for OneCap in $OTFCapList

#### Now check the Character Set Map reported by this font to see if the expected CMap is available
CMMIdx=0                                                    # Character Map Matches: Index for array
for CMap in $CMapList; do
  ((CMMIdx++))
  PFCMFSwitch=0                                             # PerFontCMapFoundSwitch is a SWITCH
  Seg=$(echo $CMap | cut -c 1-25)                           # Truncated version for display only
  if CMap=$(fc-query "$Location/$DirName$fontf" | grep "$CMap"); then
    printf "$Fmt" " " \
           "√ fc-query correctly found a Character Map Segment beginning '$Seg'"
    if [ $PFCMFSwitch == 0 ]; then                          # So we don't double count matches
      ((CMapsMatched[CMMIdx]++))                            # Increment number of charsets found
      printf "$Fmt" " " \
             "  ...match number ${CMapsMatched[CMMIdx]} for the Character Set Segment beginning '$Seg'"
      # TO DO (Maybe): HERE we need to update the counter for each char map within the font !!
      ((PFCMFSwitch++))                                     # Could just be set to 1
    fi                                                      # Done: if [ PFCMFSwitch == 0 ]
  else
    #### THIS IS ALL IN FAILURE MODE: LINE CONTAINING CMap couldn't be found; explain what was found
    ((CMapMatchFailures++))                                 # Increment number of charsets NOT found
    FullMatchFlag=0                                         # No Failures will be added to list
    printf "$Fmt" ">>" "X fc-query FAILED TO FIND A CHARACTER MAP SEGMENT DEFINED AS '$Seg'"
    SegHdr=$(echo $Seg | cut -c 1-5)                        # Truncate to just line number requested
    ActualLine=$(echo $(fc-query "$Location/$DirName$fontf" | grep "$SegHdr"))
    # Match failure might be an existing line that doesn't match or no relevant line at all
    if Alternate=$(fc-query "$Location/$DirName$fontf" | grep "$SegHdr"); then
      printf "$Fmt" " " "  ...found: '$ActualLine'"          # Show existing line for comparison
      if [ $SuspectGen == 1 ]; then                         # Switch set at beginning of this script
        printf "For '$SegHdr': $Location/$DirName$fontf: "                                   >> $SFLFN
        printf "Font doesn't match the Character Map specified.\n"                           >> $SFLFN
        printf "             Character Map reported was '$ActualLine'.\n"                    >> $SFLFN
        printf "             This might be due to one or more missing characters"            >> $SFLFN
        printf " in the font's bitmap.\n"  >> $SFLFN
      fi                                                    # Done: if [ $SuspectGen == 1 ]
    else
      printf "$Fmt" " " "  ...No relevant line for '$SegHdr' was found for this font."
      if [ $SuspectGen == 1 ]; then                         # Switch set at beginning of this script
        printf "For '$SegHdr': $Location/$DirName$fontf: "                                   >> $SFLFN
        printf "No relevant line beginning with '$SegHdr' was found.\n"                      >> $SFLFN
```

```
          fi                                                                    # Done: if [ $SuspectGen == 1 ]
        fi                                                                      # Done: Alternate=$(fc-query "$Locat...
      fi                                                                        # Done: if CMap=$(fc-query "$Locati...
    done                                                                        # Done: for CSet in $CMapList; do
    if [ $FullMatchFlag == 1 ]; then                                            # Full match flag still ON for this font?
      ((++FullMatchListIdx))
      FullMatchList[FullMatchListIdx]=$Location/$DirName$fontf
      printf "%38s %s\n" " " \
             "Complete Match $FullMatchListIdx of the $FontsMatched potential matches so far... "
    else
                          if [ $debug == 'ON' ]; then echo "No Full Match for "$Location/$DirName$fontf; fi
    fi
    echo $MinorSeparator
  else
                if [ $debug == 'ON' ]; then echo $Location/$DirName$fontf": Font Number:" $FontsChecked; fi
  fi                                                                            # Done: if [ "$Success" = "Yes" ]

 }                                                                              # Done: inspectFont() function

### MAIN FONT EXAMINATION ROUTINE: Calls inspecFont() to examine each font in each location:
echo -e "Fonts containing the Unicode Character(s):"$CharMsg                   # RTL characters are in REVERSE ORDER
echo ".............Looking in directory Trees: "$Where2Look
echo "............Checking for Language code(s):"$LangCodeList
echo "......Checking for Script Support code(s):"$OTFCapList
echo $MajorSeparator
for Location in $Where2Look                                                     # Font directories defined at beginning
do                                                                              # Examine each font directory in turn
  cd $Location                                                                  # Switch to next font directory
  # First check any fonts in this parent directory
  fontlist=$(ls -1L | grep -i \.[ot]tf )                                        # Create a list of local ttf/TTF files
  # As near as I can tell, all Open Type fonts may be either .ttf or .otf, but not all .ttf files are (or
  # have) Open Type capabilities (e.g. older .ttf fonts). The difference between fonts with Open Type
  # capabilities is that those with a .ttf extension use quadratic Bézier splines curves, and those with an
  # .otf extension use cubic Bézier spline curves (a remnant of the older PostScript Type 1 designs).
  # Beware of .ttf files that report no Open Type capabilities; they may be outdated and need replacement!
  # A collection of TrueType files packaged together has the suffix TTC, but I don't look at them here.
  for fontf in $fontlist; do                                                    # Examine each font file in turn
    inspectFont $fontlist
  done                                                                          # Done: for fontf in $fontlist
  # Now do all of the above again for each font in each subdirectory (effective limit is 2 levels!)
  DirList=$(ls -d */)                                                           # Create a list of subdirectories
  for DirName in $DirList; do                                                   # Examine each subdirectory in turn
    fontlist=$(ls -1 $DirName | grep -i \.ttf )                                 # Create a list of local ttf/TTF files
    for fontf in $fontlist; do                                                  # Examine each font file in turn
      inspectFont $fontlist
    done                                                                        # Done: for fontf in $fontlist
  done                                                                          # Done: for DirName in $DirList
done                                                                            # Done: for Location in $Where2Look

# Begin printing the on-screen summary of the font examination
echo $MajorSeparator
printf "* Search Result:%5d Truetype/Opentype files were examined for the specified characters.\n"         \
       $FontsChecked
if [ $FODTGen == 1 ]; then                                                      # If an .fodt file was requested
  printf "   <text:p >* Search Result:%5d Truetype files were examined, and</text:p>\n"                    \
         $FontsChecked                                                                          >> $DemoDoc
fi                                                                              # Done:if [ $FODTGen == 1 ]

# Print results of the character searches in the fonts ...
printf "                %5d of those files (listed above) contain all the character(s)$CharMsg.\n"         \
       $FontsMatched
echo -e "                      Text Sample(s) for this run: '$SampleText '"    # Incorrectly orders RTL Words
if [ $FODTGen == 1 ]; then                                                      # If an .fodt file was requested
  printf "   <text:p >               %5d of those files contain all the character(s)$CharMsg.</text:p>\n" \
         $FontsMatched                                                                          >> $DemoDoc
fi                                                                              # Done:if [ $FODTGen == 1 ]
printf "%21s %s\n" "                 " $MiniSeparator

for LangIdx in `seq 1 $NumArgsAccepted`; do                                     # Check Lang for each possible argument
  FinalLangCount=${LangsMatched[$LangIdx]}
  LangCode2=${LangList[$LangIdx]}
  LangAbbrev=${LangAbbrevList[$LangIdx]}
  if [ $LangCode != '99' ]; then                                                # Skip for fake languages (math, etc.)
    if [ $LangMatchFailures != 0 ]; then
      if [ $FinalLangCount != 0 ]; then
        printf "        WARNING:%5d of those $FontsMatched files contained the Language Code '$LangAbbrev'\
 ($LangCode2), BUT $LangMatchFailures FILE(s) DID NOT!\n" ${LangsMatched[$LangIdx]}
       if [ $FODTGen == 1 ]; then                                               # If an .fodt file was requested
         printf "   <text:p >        WARNING:%5d of those $FontsMatched files contained the Language Code '\
$LangAbbrev' ($LangCode2), BUT $LangMatchFailures FILE(s) DID NOT!</text:p>\n" \
                ${LangsMatched[$LangIdx]} >> $DemoDoc
       fi                                                                       # Done:if [ $FODTGen == 1 ]
      else
        if [ $LangAbbrev ]; then
          printf "               %5d of those $FontsMatched files contained the Language Code \
```

```
'$LangAbbrev' ($LangCode2).\n" ${LangsMatched[$LangIdx]}
          if [ $FODTGen == 1 ]; then                                  # If an .fodt file was requested
            printf "%32s %-5s %52s %s\n" "   <text:p >                  "${LangsMatched[$LangIdx]}         \
          " of those $FontsMatched files contained the Language Code '$LangAbbrev' ($LangCode2).</text:p>" \
                                                                                                  >> $DemoDoc
          fi                                                          # Done:if [ $FODTGen == 1 ]
        fi                                                            # Done: if [ $LangAbbrev ]
      fi                                                              # Done: if [ $FinalLangCount != 0 ]
    else
      if [ $LangAbbrev ]; then
        printf "               %5d of those $FontsMatched files contained the Language Code '$LangAbbrev'\
 ($LangCode2).\n" ${LangsMatched[$LangIdx]}
        if [ $FODTGen == 1 ]; then                                    # If an .fodt file was requested
          printf "   <text:p >                %5d of those $FontsMatched files contained the Language Code\
 '$LangAbbrev' ($LangCode2).</text:p>\n" ${LangsMatched[$LangIdx]} >> $DemoDoc
        fi                                                            # Done:if [ $FODTGen == 1 ]
      fi                                                              # Done: if [ $LangAbbrev ]
    fi                                                                # Done: if [ $LangMatchFailures != 0 ]
  fi                                                                  # Done: if [ $LangCode != '99' ]
done                                                                  # Done: for LangIdx in `seq 1 $NumArgs...
printf "%21s %s\n" "                 " $MiniSeparator                 # Separate Lang Code stats from OTFCap

# Print results of the Open Type language support in the fonts ...
OTCapIdx=0                                                            # OpenType Capabilities: Index for array
for OneCap in $OTFCapList; do                                         # Report OTF capability for each font
  ((OTCapIdx++))
  OTFMatchSuccesses=${OTFMatches[$OTCapIdx]}
  MissingOTFMatches=${OTFMatchFailures[$OTCapIdx]}
  if [ $MissingOTFMatches != 0 ]; then
    printf "        WARNING:%5d of those $FontsMatched files contained the ISO 15924 Script Code '$OneCap',\
 BUT: $MissingOTFMatches FILE(s) DID NOT!\n" $OTFMatchSuccesses
    if [ $FODTGen == 1 ]; then
      printf "   <text:p >        WARNING:%5d of those $FontsMatched files contained the ISO 15924 Script\
 Code '$OneCap', BUT: $MissingOTFMatches FILE(s) DID NOT!</text:p>\n" $OTFMatchSuccesses  >> $DemoDoc
    fi                                                                # Done:if [ $FODTGen == 1 ]
  else
    printf "               %5d of those $FontsMatched files contained the ISO 15924 Script Code\
 '$OneCap'.\n" $OTFMatchSuccesses
    if [ $FODTGen == 1 ]; then
      printf "   <text:p >               %5d of those $FontsMatched files contained the ISO 15924 Script\
 Code '$OneCap'.</text:p>\n" $OTFMatchSuccesses  >> $DemoDoc
    fi                                                                # Done:if [ $FODTGen == 1 ]
  fi                                                                  # Done: if [ $MissingOTFMatches != 0 ]
done                                                                  # Done: for OneCap in $OTFCapList
printf "%21s %s\n" "                 " $MiniSeparator                 # Separate OTFCap stats from CharMap

# Print results of the character set queries to the fonts ...
CMMIdx=0                                                              # Character Map Matches: Index for array
for CMap in $CMapList; do
  ((CMMIdx++))
  CMMatchSuccesses=${CMapsMatched[$CMMIdx]}
  MissingCMMatches=${CMapMatchFailures[$CMMIdx]}
  Seg=$(echo $CMap | cut -c 1-35)                                     # Truncated version for display only
  MiniSeg=$(echo $CMap | cut -c 1-13)                                 # Truncated version for WARNINGs only
  if [ $MissingCMMatches != 0 ]; then
    printf "        WARNING:%5d of those $FontsMatched files contained the Character Map segment beginning\
 '$MiniSeg', BUT: $MissingCMMatches FILE(s) DID NOT!\n" $CMMatchSuccesses
    if [ $FODTGen == 1 ]; then
      printf "   <text:p >        WARNING:%5d of those $FontsMatched files contain the Character Map segment\
 beginning '$Seg', BUT: $MissingCMMatches FILE(s) DID NOT!</text:p>\n" $CMMatchSuccesses  >> $DemoDoc
    fi                                                                # Done:if [ $FODTGen == 1 ]
  else
    if [ $MissingCMMatches ]; then
      printf "               %5d of those $FontsMatched files contained the Character Map segment\
 beginning '$Seg'.\n" $CMMatchSuccesses
      if [ $FODTGen == 1 ]; then
        printf "   <text:p >               %5d of those $FontsMatched files contained the Character Map\
 segment beginning '$Seg'.</text:p>\n" $CMMatchSuccesses  >> $DemoDoc
      fi                                                              # Done:if [ $FODTGen == 1 ]
    fi
  fi
done                                                                  # Done: for CSet in $CMapList; do

# Generate a file listing all of the matches that SUPPOSEDLY meet all our criteria:
if [ $FPassGen == 1 ]; then                                           # Switch set at beginning of this script
  for FNC in `seq 1 $ArgsFound`; do                                   # Build out the file name
    LLFN=$LLFN"_"${LangList[$FNC]}
  done
  LLFN=$Origin"/"$LLFN".txt"                                          # Add path and extension for permissions
  printf "%21s %s\n" "                 " $MiniSeparator             # Separate reporting section
  printf "                > Created file $LLFN listing complete matches.\n" # Notify user of file name.
  printf "This file is: " $LLFN                                                                  > $LLFN
  printf "Suitable Fonts for mixing multiple Scripts/Languages: \n"                             >> $LLFN
  printf "$MiniSeparator\n"                                                                     >> $LLFN
  printf "The following directory tree(s) were examined: $Where2Look\n"                         >> $LLFN
```

```
  printf "\n"                                                                                    >> $LLFN
  printf "  $FullMatchListIdx fonts found of the $FontsChecked font files examined:\n"          >> $LLFN
  printf "    a) contained the characters: $CharMsg\n"                                         >> $LLFN
  printf "    b) reported the corresponding Language Code(s): $LangCodeList\n"                 >> $LLFN
  printf "    c) reported the corresponding Script Code(s): $OTFCapList\n"                     >> $LLFN
  printf "    d) matched all the defined Character Map Segment(s)\n"                           >> $LLFN
  printf "  $MiniSeparator\n"                                                                  >> $LLFN
  printf "\n"                                                                                  >> $LLFN
  printf "  A List of those potentially usable Font files (for further evaluation) is:\n"     >> $LLFN
  for Id in `seq 1 $FullMatchListIdx`; do                          # Create the actual list of fonts
    printf "%5s: %s\n" $Id ${FullMatchList[$Id]}                                               >> $LLFN
  done
fi                                                                 # Done: if [ $FPassGen == 1 ]

# Indicate on screen that the list of potentially faulty fonts was created and give its name
if [ $SuspectGen == 1 ]; then                                      # Switch set at beginning of this script
  printf "%21s %s\n" "                " $MiniSeparator           # Separate reporting section
  printf "                    > Created file $SFLFN listing of possibly faulty fonts.\n"
fi                                                                 # Done: if [ $SuspectGen == 1 ]

echo $MajorSeparator                                               # Screen Report completed!

# Here we complete the .fodt output file with a summary; with echo, actual spaces can be used with echo.
if [ $FODTGen == 1 ]; then                                         # Switch set at beginning of this script
  echo '   <text:p/>'                                                                          >> $DemoDoc
  echo '  </office:text>'                                                                      >> $DemoDoc
  echo ' </office:body>'                                                                       >> $DemoDoc
  echo '</office:document>'                                                                    >> $DemoDoc
fi                                                                 # Done if [ $FODTGen == 1 ]

##### END OF CODE HERE ###############################################################################
```