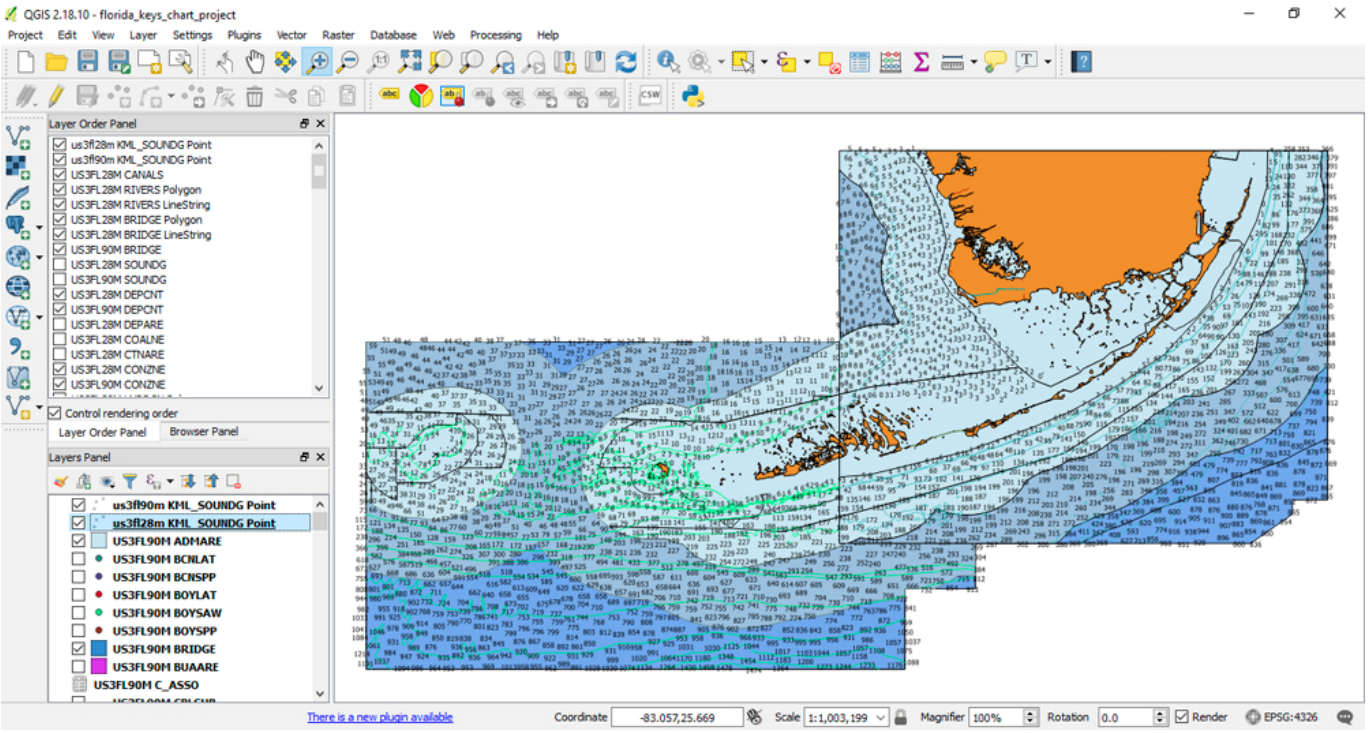
Using Python and GDAL to Split Multipart SOUNDG Lat/Long/Depth from NOAA S-57 ENC Files into KML Files for use with GIS Software

Dave Liske, Luna Pier, Michigan
August 2017

Abstract: A 2003 Python script named get_soundg.py extracted ELEV from the multipart SOUNDG layer, but in 2017 was found to generate numerous errors. Building on the excellent foundation the code provided while extracting to the KML file type provides a viable solution for what appears to be a common request.



Extracted SOUNDG layer data from us3fl90m.000 and us3fl28m.000, displayed as Labels in QGIS 2.18. This work is part of a larger personal art project.

This code is an update of get_soundg.py developed by Frank Warmerdam, a Vice President of the Open Source Geospatial Foundation and the primary developer of the Geospatial Data Abstraction Library (GDAL) until version 1.3.2, in 2003. The purpose of the get_soundg.py file was to extract sounding data from an S-57 file and write the data to a Shapefile.

When run using US3FL28M.000 from the National Oceanic and Atmospheric Administration (NOAA) Office of Coast Survey in June 2017 the file exhibited a number of errors, specifically:

- The ESRI Shapefile driver does not support the ImageList and StringList field types for the FFPT_RIND and LNAM_REFS fields respectively.
- An error was subsequently generated for each feature in the resulting file, which for US3FL28M.000 would have been 2,531 errors if python.exe hadn't stopped listing errors after 1,000.

The Shapefile was, however, created and was usable. But when the Shapefile was imported into Quantum GIS 2.18, QGIS indicated there was no CRS in the resulting Shapefile.

Following are the changes made in June to August 2017 when get_soundg.py was saved as s57_kml_extract_soundg.py:

- Added error handling if SOUNDG layer or source S-57 file is not found.
- Deleted second usage argument: Added code to create target file in same directory as source S-57 file.
- Target File Type: Changed to KML. Errors are subsequently handled correctly, i.e. the errors are suppressed while the FFPT_RIND and LNAM_REFS fields are also not included in the resulting KML file.
- Added WGS84 CSR to the target KML file.
- Changed created layer name from 'out' to 'kml_soundg'. When imported into QGIS the resulting layer is displayed as <kmlfile>_kml_soundg_Point in the Browser Panel.
- Added target fields 'LAT' and 'LONG' for xcoord and ycoord. The precision of these fields is set to 7 in accordance with International Hydrographic Organization (IHO) publication 'S-57 APPENDIX B.1 Annex A – Use of the Object Catalogue for ENC' (Electronic Navigation Chart).
- Added calculations to convert 'LAT' and 'LONG' values to Degrees, Minutes, Seconds, and NSEW, storing these values in individual fields for each sounding.
- Added a field for an assembled string formatted as "D M S [NSEW]"
- Changed target field 'ELEV' to 'DEPTHM', which is accurate for soundings.
- Changed precision for 'DEPTHM' field from 4 to 0 to reflect chart usage.
- Added calculated 'DEPTHF' field for soundings in feet.
- Changed feat2 to feat1 to prevent searching for unused feat1.
- Added completion messages.

To get there:

- Download and install the appropriate current version of QGIS for your platform. This will also install Python.
- Download the desired ENCs from the NOAA Office of Coast Survey web site, and extract them.
- Download the ZIP file containing s57_kml_extract_soundg.py and extract the file to a known folder.

Sample usage using OSGeo4W Shell, which is installed with QGIS:

```
C:\>python c:\users\rob\desktop\s57_kml_extract_soundg.py
c:\users\rob\desktop\FL_ENCs\ENC_ROOT\us3fl28m\us3fl28m.000
```

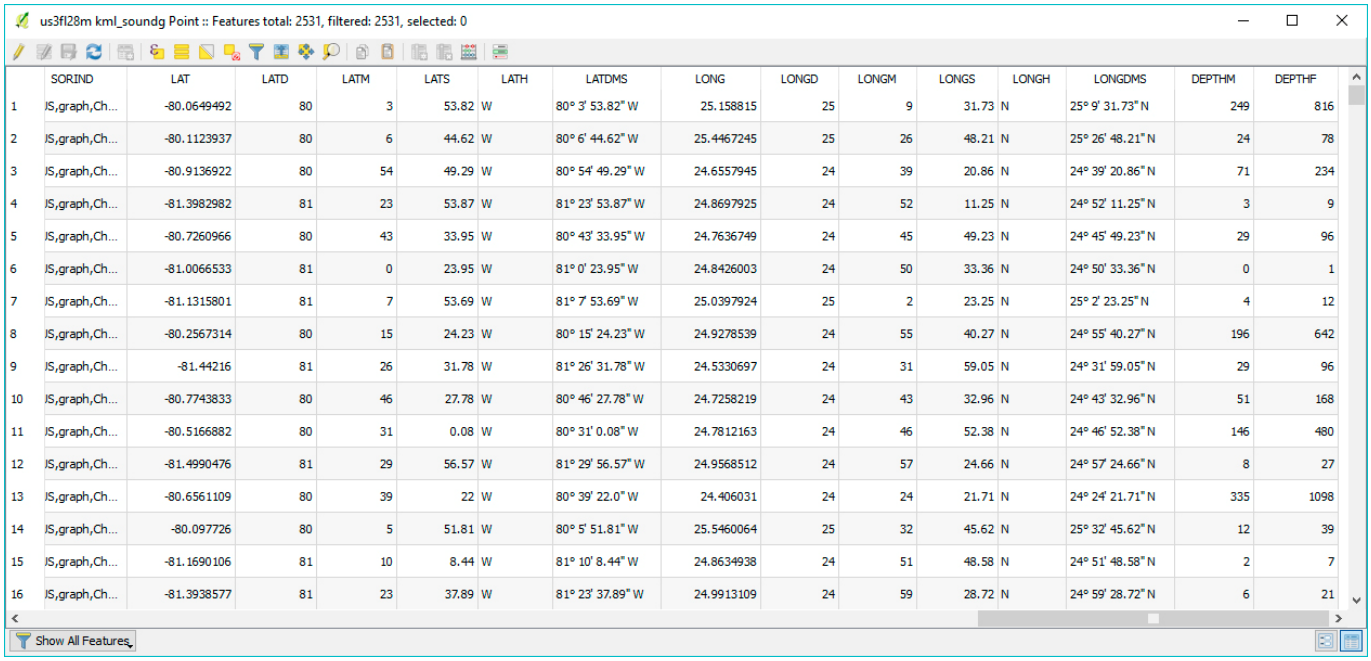
The result will be:

```
C:\>python c:\users\rob\desktop\s57_kml_extract_soundg.py
c:\users\rob\desktop\FL_ENCs\ENC_ROOT\us3fl28m\us3fl28m.000

c:\users\rob\desktop\FL_ENCs\ENC_ROOT\us3fl28m\us3fl28m.kml created ...
SOUNDG layer created ...
2531 features extracted
```

To view the soundings in QGIS 2.18:

- Import the KML file into QGIS.
- In the Layer Properties dialog, turn off Symbols, and set the Labels tab to show data from one of the 'DEPTH(z)' fields.



	SORIND	LAT	LATD	LATM	LATS	LATH	LATDMS	LONG	LONGD	LONGM	LONGS	LONGH	LONGDMS	DEPTHM	DEPTHF
1	/S,graph,Ch...	-80.0649492	80	3	53.82	W	80° 3' 53.82" W	25.158815	25	9	31.73	N	25° 9' 31.73" N	249	816
2	/S,graph,Ch...	-80.1123937	80	6	44.62	W	80° 6' 44.62" W	25.4467245	25	26	48.21	N	25° 26' 48.21" N	24	78
3	/S,graph,Ch...	-80.9136922	80	54	49.29	W	80° 54' 49.29" W	24.6557945	24	39	20.86	N	24° 39' 20.86" N	71	234
4	/S,graph,Ch...	-81.3982982	81	23	53.87	W	81° 23' 53.87" W	24.8697925	24	52	11.25	N	24° 52' 11.25" N	3	9
5	/S,graph,Ch...	-80.7260966	80	43	33.95	W	80° 43' 33.95" W	24.7636749	24	45	49.23	N	24° 45' 49.23" N	29	96
6	/S,graph,Ch...	-81.0066533	81	0	23.95	W	81° 0' 23.95" W	24.8426003	24	50	33.36	N	24° 50' 33.36" N	0	1
7	/S,graph,Ch...	-81.1315801	81	7	53.69	W	81° 7' 53.69" W	25.0397924	25	2	23.25	N	25° 2' 23.25" N	4	12
8	/S,graph,Ch...	-80.2567314	80	15	24.23	W	80° 15' 24.23" W	24.9278539	24	55	40.27	N	24° 55' 40.27" N	196	642
9	/S,graph,Ch...	-81.44216	81	26	31.78	W	81° 26' 31.78" W	24.5330697	24	31	59.05	N	24° 31' 59.05" N	29	96
10	/S,graph,Ch...	-80.7743833	80	46	27.78	W	80° 46' 27.78" W	24.7258219	24	43	32.96	N	24° 43' 32.96" N	51	168
11	/S,graph,Ch...	-80.5166882	80	31	0.08	W	80° 31' 0.08" W	24.7812163	24	46	52.38	N	24° 46' 52.38" N	146	480
12	/S,graph,Ch...	-81.4990476	81	29	56.57	W	81° 29' 56.57" W	24.9568512	24	57	24.66	N	24° 57' 24.66" N	8	27
13	/S,graph,Ch...	-80.6561109	80	39	22	W	80° 39' 22.0" W	24.406031	24	24	21.71	N	24° 24' 21.71" N	335	1098
14	/S,graph,Ch...	-80.097726	80	5	51.81	W	80° 5' 51.81" W	25.5460064	25	32	45.62	N	25° 32' 45.62" N	12	39
15	/S,graph,Ch...	-81.1690106	81	10	8.44	W	81° 10' 8.44" W	24.8634938	24	51	48.58	N	24° 51' 48.58" N	2	7
16	/S,graph,Ch...	-81.3938577	81	23	37.89	W	81° 23' 37.89" W	24.9913109	24	59	28.72	N	24° 59' 28.72" N	6	21

LAT, LONG, calculated DMS, hemisphere, DMS[NSEW], DEPTHM, and DEPTHF fields in the imported us3fl28m kml_soundg_Point layer from the generated KML file.

To export the 'DEPTH(z)' labels from QGIS 2.18 as an SVG file for use in other applications:

- Immediately prior to exporting, open the Layer Properties dialog to the Labels tab.
- In the top-right of the Labels tab, click the Automated Placement Settings button. Ensure 'Draw text as outlines' is checked.
- Click 'OK' to exit the two dialogs.
- Click Project => New Print Composer.
- Click 'OK' on the Composer Title dialog (no input is required).
- Click Layout => Add Map.
- Immediately draw a rectangle on the canvas. After a time, the 'DEPTH(z)' labels will appear.
- Click Composer => Export as SVG. A warning will appear about several problems due to bugs and deficiencies in the SVG exporter. Click 'Close'.
- In the File Save dialog, include the desired SVG file name and location and click 'OK'.
- Uncheck 'Render map labels as outlines' and click 'Save'.
- The SVG file will be exported.

Following is the complete code for s57_kml_extract_soundg.py:

```
#############################################################################
# $Id: s57_kml_extract_soundg.py 2017-08-01 Dave Liske <dave@micuisine.com>
#
# Purpose:  Extract SOUNDGings from an S-57 dataset, and write them to
#           KML format, creating one feature for each sounding, and
#           adding the latitudes, longitudes and depths as attributes
#                 for easier use.
# Authors:  Frank Warmerdam, 2003 version, warmerdam@pobox.com
#                 Dave Liske, 2017 version, dave@micuisine.com
#
#############################################################################
# Changes, June - August 2017
#
# This file is an update of get_soundg.py developed by Frank Warmerdam in 2003.
```

```
# The purpose of the get_soundg.py file was to extract sounding data from an
# S-57 file and write the data to a Shapefile. When run using US3FL28M.000
# from NOAA in June 2017 the file exhibited a number of errors, specifically:
#
#     * The ESRI Shapefile driver does not support the ImageList and StringList
#           field types for the FFPT_RIND and LNAM_REFS fields respectively.
#     * An error was subsequently generated for each feature in the resulting
#           file, which for US3FL28M.000 would have been 2,531 errors if python.exe
#           hadn't stopped listing errors after 1,000.
#
# The Shapefile was, however, created and was usable. But when the Shapefile
# was imported into QGIS 2.18, QGIS indicated there was no CRS in the
# resulting Shapefile.
#
# Following are the changes made from June to August 2017 when get_soundg.py
# was saved as s57_kml_extract_soundg.py:
#
#     * Added error handling if SOUNDG layer or source S-57 file is not found.
#     * Deleted second usage argument: Added code to create target file in same
#           directory as source S-57 file.
#     * Target File Type: Changed to KML. Errors are subsequently handled
#           correctly, i.e. the errors are suppressed while the FFPT_RIND and
#           LNAM_REFS fields are also not included in the resulting KML file.
#     * Added WGS84 CSR to the target KML file.
#     * Changed created layer name from 'out' to 'kml_soundg'. When imported
#           into QGIS the resulting layer is displayed as
#           <kmlfile>_kml_soundg_Point in the Browser Panel.
#     * Added target fields 'LAT' and 'LONG' for xcoord and ycoord. The precision
#           of these fields is set to 7 IAW IHO publication 'S-57 APPENDIX B.1
#           Annex A - Use of the Object Catalogue for ENC'.
#     * Added calculations to convert 'LAT' and 'LONG' values to Degrees,
#           Minutes, Seconds, and NSEW, storing these values in individual
#           fields for each sounding.
#     * Added a field for an assembled string formatted as "D M S [NSEW]"
#     * Changed target field 'ELEV' to 'DEPTHM', which is accurate for soundings.
#     * Changed precision for 'DEPTHM' field from 4 to 0 to reflect chart usage.
#     * Added calculated 'DEPTHF' field for soundings in feet.
#     * Changed feat2 to feat1 to prevent searching for unused feat1.
#     * Added completion messages.
#
#     To view the soundings in QGIS 2.18:
#
#     * Import the KML file into QGIS.
#     * In the Layer Properties dialog, turn off Symbols, and set the Labels
#           tab to show data from the 'DEPTH' field.
#
#     To export the 'DEPTH' labels from QGIS 2.18 as an SVG file for use
#     in other applications:
#
#     * Immediately prior to exporting, open the Layer Properties dialog to
#           the Labels tab.
#     * In the top-right of the Labels tab, click the Automated Placement
#           Settings button. Ensure 'Draw text as outlines' is unchecked.
#     * Click 'OK' to exit the two dialogs.
#     * Click Project => New Print Composer.
#     * Click 'OK' on the Composer Title dialog (no input is required).
#     * Click Layout => Add Map.
#     * Immediately draw a rectangle on the canvas. After a time, the 'DEPTH'
#           labels will appear.
#     * Click Composer => Export as SVG. A warning will appear about several
#           problems due to bugs and deficiencies in the SVG exporter.
```

```
#           Click 'Close'.
#     * In the File Save dialog, include the desired SVG file name and location
#           and click 'OK'.
#     * Ensure 'Render map labels as outlines' is unchecked and click 'Save'.
#     * The SVG file will be exported. The individual text fields will be
#           editable in Adobe Illustrator, etc.
#
###############################################################################
# Copyright (c) 2003, Frank Warmerdam <warmerdam@pobox.com>
# Additional Copyright (c) 2017, Dave Liske <dave@micuisine.com>
#
# Permission is hereby granted, free of charge, to any person obtaining a
# copy of this software and associated documentation files (the "Software"),
# to deal in the Software without restriction, including without limitation
# the rights to use, copy, modify, merge, publish, distribute, sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included
# in all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
# OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
# THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
# FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
# DEALINGS IN THE SOFTWARE.
###############################################################################-

try:
    from osgeo import ogr
except ImportError:
    import ogr

import sys
import os.path
import osgeo.osr as osr

###############################################################################-
def Usage():
    print('')
    print('Usage: s57_kml_extract_soundg.py <source s57file>')
    sys.exit(1)

def NoSoundG():
    print('')
    print('Error: SOUNDG layer or source S-57 file not found.')
    sys.exit(1)

###############################################################################-
# Argument processing

if len(sys.argv) != 2:
    Usage()

s57filename = sys.argv[1]

# Specify the target KML file in the same directory
filename, file_extension = os.path.splitext(s57filename)
kmlfilename = filename
```

```
kmlfilename += ".kml"

############################################################################-
# Open the S57 file, and find the SOUNDG layer.

ds = ogr.Open( s57filename )

try:
      src_soundg = ds.GetLayerByName( 'SOUNDG' )
except AttributeError:
      NoSoundG()

############################################################################-
# Create the output KML file.

kml_driver = ogr.GetDriverByName( 'KML' )
kml_ds = kml_driver.CreateDataSource( kmlfilename )

# Import CSR WGS84
srs = osr.SpatialReference()
srs.ImportFromEPSG(4326)

# Specify the new SOUNDG layer
kml_layer = kml_ds.CreateLayer( 'kml_soundg', srs, geom_type = ogr.wkbPoint25D )

src_defn = src_soundg.GetLayerDefn()
field_count = src_defn.GetFieldCount()

# Display progress message
print('')
print kmlfilename, "created ..."

############################################################################-
# Copy the SOUNDG schema, duplicating the current fields, and create the LAT,
#     LONG and DEPTH fields for the extracted Point and calculated data

out_mapping = []
for fld_index in range(field_count):

      src_fd = src_defn.GetFieldDefn( fld_index )
      fd = ogr.FieldDefn( src_fd.GetName(), src_fd.GetType() )
      fd.SetWidth( src_fd.GetWidth() )
      fd.SetPrecision( src_fd.GetPrecision() )
      if kml_layer.CreateField( fd ) != 0:
            out_mapping.append( -1 )
      else:
            out_mapping.append( kml_layer.GetLayerDefn().GetFieldCount() - 1 )

fd = ogr.FieldDefn( 'LAT', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LATD', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LATM', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
```

```
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LATS', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LATH', ogr.OFTString )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LATDMS', ogr.OFTString )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LONG', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LONGD', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LONGM', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LONGS', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LONGH', ogr.OFTString )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'LONGDMS', ogr.OFTString )
fd.SetWidth( 12 )
fd.SetPrecision( 7 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'DEPTHM', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 0 )
kml_layer.CreateField( fd )

fd = ogr.FieldDefn( 'DEPTHF', ogr.OFTReal )
fd.SetWidth( 12 )
fd.SetPrecision( 0 )
kml_layer.CreateField( fd )

# Display progress message
print "SOUNDG layer created ..."

############################################################################
```

```
#  Process the data for the new fields

feat = src_soundg.GetNextFeature()

# The degree symbol needs to be defined for use
deg = u"\u00b0"

# GetFeatureCount can be expensive, so ...
iCount = 0

while feat is not None:

    multi_geom = feat.GetGeometryRef()

    for iPnt in range(multi_geom.GetGeometryCount()):
        # Get the multipart data
        pnt = multi_geom.GetGeometryRef( iPnt )

        # Get the layer definition from the KML file
        feat1 = ogr.Feature(feature_def=kml_layer.GetLayerDefn())

        for fld_index in range(field_count):
            feat1.SetField( out_mapping[fld_index], feat.GetField( fld_index ) )

        # Get the LATITUDE in Degrees.Decimal format
        soundinglat = pnt.GetX( 0 )

        # Extract the Degrees
        latdegrees = int(abs(soundinglat))
        # Extract the decimals
        trunclatdegrees = (abs(soundinglat) - latdegrees)
        # Calculate the Minutes
        latminutes =  int(trunclatdegrees * 60)
        # Calculate the Seconds
        latseconds = round(((trunclatdegrees * 3600) % 60), 2)

        # Negative LATITUDE is West of the Zero Meridian
        if soundinglat <= 0:
            lathemisphere = "W"
        else:
            lathemisphere = "E"

        # Assemble the LATDMS string
        latstring = str(latdegrees)
        # Encode the defined degree symbol as utf-8 and add it
        latstring += deg.encode('utf8')
        latstring += " "
        latstring += str(latminutes)
        latstring += "' "
        latstring += str(latseconds)
        latstring += '" '
        latstring += lathemisphere

        # Write the LATITUDE fields
        feat1.SetField( 'LAT', soundinglat )
        feat1.SetField( 'LATD', latdegrees )
        feat1.SetField( 'LATM', latminutes )
        feat1.SetField( 'LATS', latseconds )
        feat1.SetField( 'LATH', lathemisphere )
        feat1.SetField( 'LATDMS', latstring )
```

```
        # Get the LONGITUDE in Degrees.Decimal format
        soundinglong = pnt.GetY( 0 )

        # Extract the Degrees
        longdegrees = int(abs(soundinglong))
        # Extract the decimals
        trunclongdegrees = (abs(soundinglong) - longdegrees)
        # Calculate the Minutes
        longminutes =  int(trunclongdegrees * 60)
        # Calculate the Seconds
        longseconds = round(((trunclongdegrees * 3600) % 60), 2)

        # Negative LONGITUDE is South of the equator
        if soundinglong <= 0:
            longhemisphere = "S"
        else:
            longhemisphere = "N"

        # Assemble the LONGDMS string
        longstring = str(longdegrees)
        # Encode the defined degree symbol as utf-8 and add it
        longstring += deg.encode('utf8')
        longstring += " "
        longstring += str(longminutes)
        longstring += "' "
        longstring += str(longseconds)
        longstring += '" '
        longstring += longhemisphere

        # Write the LONGITUDE fields
        feat1.SetField( 'LONG', soundinglong )
        feat1.SetField( 'LONGD', longdegrees )
        feat1.SetField( 'LONGM', longminutes )
        feat1.SetField( 'LONGS', longseconds )
        feat1.SetField( 'LONGH', longhemisphere )
        feat1.SetField( 'LONGDMS', longstring )

        # Use metres as provided and calculate feet
        soundingmetres = pnt.GetZ( 0 )
        soundingfeet = soundingmetres * 3.28084

            # Write the DEPTH fields
        feat1.SetField( 'DEPTHM', soundingmetres )
        feat1.SetField( 'DEPTHF', soundingfeet )

            # Create the feature
        feat1.SetGeometry( pnt )
        kml_layer.CreateFeature( feat1 )
        feat1.Destroy()

      iCount = iCount +1
    feat.Destroy()

    feat = src_soundg.GetNextFeature()

#############################################################################
# Cleanup

print iCount, "features extracted"

kml_ds.Destroy()
```

```
ds.Destroy()
```

Python Latitude/Longitude Formulas and Associated String Manipulations Used in this Script

Splitting the Filename, Changing the Extension

Near the beginning of this script, we use os.path.splitext to split the filename and its extension. We then add ".kml" to the filename and use this for the new file.

```
import os.path
# Specify the target KML file in the same directory
filename, file_extension = os.path.splitext(s57filename)
kmlfilename = filename
kmlfilename += ".kml"
```

Converting Decimal Degrees to Degrees, Minutes, Seconds

The latitude and longitude as extracted from an S-57 file are formatted in what's known as Decimal Degrees. The degrees from the Zero Meridian are to the left of the decimal, with negative numbers indicating the hemisphere. The degrees are extracted first using the ABS function, forced to an integer using INT, and assigned to a variable. This is then subtracted from the initial value, and the remainder is then converted into Minutes and Seconds using standard formulas. Seconds are stored with a precision of 2.

Finding the hemisphere is a simple matter of testing whether or not the initial Decimal Degrees value is negative.

```
# Get the LATITUDE in Degrees.Decimal format
soundinglat = pnt.GetX( 0 )
# Extract the Degrees
latdegrees = int(abs(soundinglat))
# Extract the decimals
trunclatdegrees = (abs(soundinglat) - latdegrees)
# Calculate the Minutes
latminutes = int(trunclatdegrees * 60)
# Calculate the Seconds
latseconds = round(((trunclatdegrees * 3600) % 60), 2)
# Negative LATITUDE is West of the Zero Meridian
if soundinglat <= 0:
     lathemisphere = "W"
else:
     lathemisphere = "E"
```

Assembling Degrees, Minutes, Seconds into a DMS String

The degree symbol, "°", isn't handled well in Python. We need to define it in hexadecimal, and then encode it as utf-8 when it's included as part of a string. Once we've defined the degree symbol, assembling a string as DMS [NSEW] becomes quite simple using the variables as assigned in the above example.

```
# The degree symbol needs to be defined for use
deg = u"\u00b0"
# Assemble the LATDMS string
latstring = str(latdegrees)
# Encode the defined degree symbol as utf-8 and add it
latstring += deg.encode('utf8')
latstring += " "
latstring += str(latminutes)
latstring += "' "
latstring += str(latseconds)
```

```
latstring += '" '
latstring += lathemisphere
```

Converting Metres to Feet

S-57 files are defined as per the International Hydrographic Organization, with its headquarters in Monaco. As such, soundings are encoded in Metres. For use in the United States and similar countries, we'll convert the soundings to Feet.

```
# Use metres as provided and calculate feet
soundingmetres = pnt.GetZ( 0 )
soundingfeet = soundingmetres * 3.28084
```