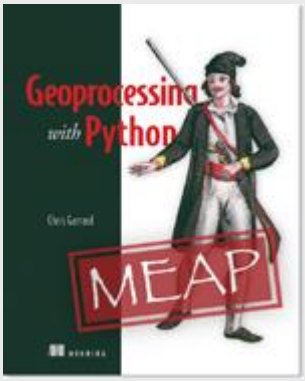

Using spatial references with OSR

By *Chris Garrard*

This article, excerpted from the book Geoprocessing with Python, will show you how to use OSR to assign SRS information to your data so that GIS software, including OSR, and know how to work with it.

Because spatial reference systems (SRS) are so important, most vector data formats provide a way to store this information with the data, and you need to know how to work with it. When using spatial data, one common task is to convert the dataset from one spatial reference system to another so that it can be used with other datasets or for a particular analysis.

Another reason you might need to convert between spatial reference systems is if you are using an online mapping solution that requires a Web Mercator projection in order to display data.

> **WARNING** Many GIS software packages will project on-the-fly, which means that they will automatically convert data to a different SRS when displaying it. For example, if you loaded in a dataset that used an Albers equal-area projection, the map would be drawn using that projection. But if you then loaded a second file that used UTM, it would be converted to Albers so it could be displayed correctly with the first. Of course, this only happens if both of the datasets have SRS information stored with them, because without that the software doesn't know what to do. Also, this process only changes what is in memory and does not alter what is stored on disk in any way. Although this behavior can be helpful when using a GIS, sometimes it leads people to assume that datasets use the same SRS when in reality they do not.

The osgeo package contains a module called OSR which is used to work with spatial reference systems. This article will show you how to use OSR to assign SRS information to your data so that GIS software, including OSR, know how to work with it.

Spatial reference objects

In order to work with a spatial reference system, you need a `SpatialReference` object that represents it. If you already have a layer or geometry that uses the SRS you want, then you

can get the SRS from it using the `GetSpatialRef` or `GetSpatialReference` function, respectively. Both of these functions will return `None` if the layer or geometry does not have a SRS stored with it.

Let's take a look at the information contained in one of these SRS objects. Perhaps the easiest way is to print it out, which will display a nicely formatted description of the SRS in Well-known text (WKT) format. The states_48 shapefile uses a geographic, or unprojected, coordinate system along with the North American Datum of 1983 (NAD83).

```
>>> ds = ogr.Open(r'D:\OSPyBookData\US\states_48.shp')
>>> print(ds.GetLayer().GetSpatialRef())
GEOGCS["GCS_North_American_1983",
    DATUM["North_American_Datum_1983",
        SPHEROID["GRS_1980",6378137.0,298.257222101]],
    PRIMEM["Greenwich",0.0],
    UNIT["Degree",0.0174532925199433]]
```

You can tell that this is not a projected SRS because it does not have a PROJCS entry, only a GEOGCS one. If you were looking at a projected SRS, there would be more information describing the parameters of the coordinate system, such as the UTM example show in figure 1.

```
PROJCS["NAD83 / UTM zone 12N",
    GEOGCS["NAD83",
        DATUM["North_American_Datum_1983",
            SPHEROID["GRS 1980",6378137,298.257222101,
                AUTHORITY["EPSG","7019"]],
            TOWGS84[0,0,0,0,0,0,0],
            AUTHORITY["EPSG","6269"]],
        PRIMEM["Greenwich",0,
            AUTHORITY["EPSG","8901"]],
        UNIT["degree",0.0174532925199433,
            AUTHORITY["EPSG","9122"]],
        AUTHORITY["EPSG","4269"]],
    PROJECTION["Transverse_Mercator"],
    PARAMETER["latitude_of_origin",0],
    PARAMETER["central_meridian",-111],
    PARAMETER["scale_factor",0.9996],
    PARAMETER["false_easting",500000],
    PARAMETER["false_northing",0],
    UNIT["metre",1,
        AUTHORITY["EPSG","9001"]],
    AXIS["Easting",EAST],
    AXIS["Northing",NORTH],
    AUTHORITY["EPSG","26912"]]
```

Figure 1 Well-known text for the NAD83 UTM Zone 12N spatial reference system.

Well-known text is not the only string representation of a SRS. I like PROJ.4 strings because they are especially concise. For example, this is the PROJ.4 string for the UTM SRS shown in figure 1:

```
'+proj=utm +zone=12 +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs '
```

The PROJ.4 Cartographic Projections Library is a popular open source library for converting data between projections, and you can read about the details of PROJ.4 definitions at http://trac.osgeo.org/proj/.

> **DEFINITION** Spatial Reference System Identifiers (SRIDs) are used to uniquely identify each spatial reference system, datum, and several other related items within a GIS. The software can use its own set of IDs, or it can use a common set such as EPSG (short for European Petroleum Survey Group) codes. These are the AUTHORITY entries in the WKT examples.

Fortunately, you do not have to print anything out in order to discover if a SRS is geographic or projected, because there are handy functions called `IsGeographic` and `IsProjected` to provide that information. You can also get other information about the SRS, although you do need to know the structure of a SRS in order to do so. Go back and look at the WKT in figure 1. You can use the `GetAttrValue` function to get the text corresponding to the first occurrence of each keyword such as PROJCS or DATUM, where the keywords are not casesensitive. Assuming that the `utm_sr` variable holds the SRS from figure 1, you could get the projection name like this:

```
>>> utm_sr.GetAttrValue('PROJCS')
'NAD83 / UTM zone 12N'
```

There are several AUTHORITY entries in the UTM SRS. Which one do you think will be returned by `GetAttrValue`? Let's try it and see:

```
>>> utm_sr.GetAttrValue('AUTHORITY')
'EPSG'
```

That didn't tell you much, since the first value of each one happens to be 'EPSG'. There is an optional parameter to `GetAttrValue` that lets you specify which child you want returned using its index. The string 'EPSG' is the first child, but the second is a number, so try getting it:

```
>>> utm_sr.GetAttrValue('AUTHORITY', 1)
'26912'
```

Why did it get the last one shown in figure 1 instead of the first? This is because items are nested inside each other in the SRS, and this one is the least nested, so it is the first one returned by the function.

So if `GetAttrValue` only grabs the first item that appears with a given keyword, how do you get the others? To get authority codes, or SRIDs, just pass the key for the SRID that you're interested in to `GetAuthorityCode`:

```
>>> utm_sr.GetAuthorityCode('DATUM')
'6269'

>>> utm_sr.GetProjParm(osr.SRS_PP_FALSE_EASTING)
500000.0
```

There are many other functions for getting information from an SRS, some of which only apply to certain types of spatial reference systems.

Creating spatial reference objects

Because you will not always be able to get an appropriate spatial reference object from a layer or geometry, you need to be able to create your own. Because I like short representations, my two favorite ways to do this are to use a standard EPSG code if it exists for the SRS I want to use, or a PROJ.4 string. As you just saw with the UTM example, the EPSG code for NAD83 UTM 12N is 26912, which you can pass to the `ImportFromEPSG` function after creating a blank `SpatialReference` object.

```
>>> from osgeo import osr
>>> sr = osr.SpatialReference()
>>> sr.ImportFromEPSG(26912)
0
>>> sr.GetAttrValue('PROJCS')
'NAD83 / UTM zone 12N'
```

The SRS you just created is equivalent to the UTM example from earlier. The zero returned by `ImportFromEPSG` means that the SRS was imported successfully. Interestingly, watch what happens if you use the PROJ.4 string you saw earlier:

```
>>> sr = osr.SpatialReference()
>>> sr.ImportFromProj4('''+proj=utm +zone=12 +ellps=GRS80
...                       +towgs84=0,0,0,0,0,0,0 +units=m +no_defs ''')
0
>>>
>>> sr.GetAttrValue('PROJCS')
'UTM Zone 12, Northern Hemisphere'
```

The datum name is no longer part of the SRS name because the datum was not specified in the PROJ.4 string. However, the GRS80 ellipsoid used by the NAD83 datum was part of the string, so the required information is still there (if you want to prove it to yourself, print out the WKT and compare the SPHEROID values to the ones from figure 1). To include the datum, add `+datum=NAD83` to the PROJ.4 representation.

> **TIP** You can look up EPSG codes, WKT, PROJ.4 strings, and several other representations of spatial reference systems at www.spatialreference.org.

Just as there are several different functions for exporting SRS information, there are also multiple methods for importing this information into a spatial reference object, including one

to import information from a URL. You can also create a spatial reference object from a WKT string without having to use one of the importer functions:

```
>>> wkt = '''GEOGCS["GCS_North_American_1983",
...             DATUM["North_American_Datum_1983",
...               SPHEROID["GRS_1980",6378137.0,298.257222101]],
...             PRIMEM["Greenwich",0.0],
...             UNIT["Degree",0.0174532925199433]]'''
>>>
>>> sr = osr.SpatialReference(wkt)
```

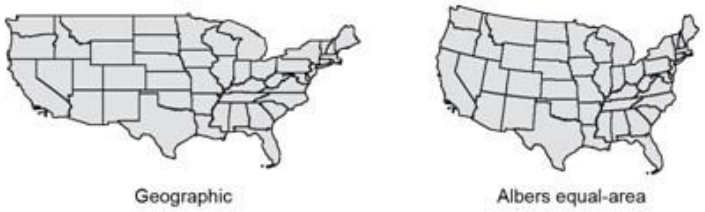

Figure 2 The lower 48 states shown using geographic coordinates and an Albers equal-area projection.

You can also build a spatial reference object yourself, although there are some projection-specific functions to help you with this. Let's branch out from UTM and build the Albers Conic Equal Area SRS that the United States Geological Survey uses for the lower 48 states (figure 2). The projection-specific function for Albers looks like this: `SetACEA(stdp1, stdp2, clat, clong, fe, fn)`

The parameters are standard parallel 1, standard parallel 2, latitude of center, longitude of center, false easting, and false northing, in that order. You could use this to build the USGS Albers spatial reference.

```
>>> sr = osr.SpatialReference()
>>> sr.SetProjCS('USGS Albers')
>>> sr.SetWellKnownGeogCS('NAD83')
>>> sr.SetACEA(29.5, 45.5, 23, -96, 0, 0)
>>> sr.Fixup()
>>> sr.Validate()
0
```

The first thing you did after creating an empty SRS was set a name for it, then specified a datum, and lastly you provided the required parameters for the Albers projection. The call to `Fixup` adds default values for missing parameters and reorders items so that they match the standard. The last thing you need to do is call `Validate` to make sure that you didn't forget anything. In this case, `Validate` returned a zero, which means everything is fine (in fact, many of the other functions in this example also returned zero, but I cut the returned values

out of the examples in the interest of space). Try leaving out the datum and see what happens when you call `Validate`. In that case, it should return 5, which means that the SRS is corrupt. This is because a SRS needs either a datum or a spheroid, neither of which is specified if you leave out the call to `SetWellKnownGeogCS`. If you turn on exception handling with `osr.UseExceptions(True)` then `Validate` will throw an exception instead of returning a number.

Assigning a SRS to data

It is a good idea to attach SRS information to your dataset whenever possible, so that you always know what coordinate system it uses. You can assign an SRS to layers and individual geometries, although all geometries in a layer share the same SRS. A data source cannot be assigned an SRS because individual layers are allowed to have different spatial reference systems.

If you've ever created new layer, you know that one of the parameters for the `CreateLayer` function is a spatial reference object. The default value for this parameter is `None`, because OGR cannot figure out what SRS the data use on its own. If you have a spatial reference object, you need to provide it when you create a new layer because there is no function to add a SRS to an existing layer.

```
sr = osr.SpatialReference() sr.ImportFromEPSG(26912)
ds = ogr.GetDriverByName('ESRI Shapefile').Open(r'D:\Temp', 1)
lyr = ds.CreateLayer('counties', sr, ogr.wkbPolygon)                    #A
```

#A The SRS is the second parameter when creating the layer

Now the new counties shapefile and all of the geometries contained in it will know that they use a UTM SRS (EPSG 26912). Of course, you really must create the geometries using UTM coordinates. Assigning a SRS to a layer does not magically convert all of the data to that coordinate system. All it does is provide information, so if you assign a different SRS than you're actually using, you're basically lying and causing confusion because nothing will know how to work with the data.

If you are working with individual geometries instead of layers, you might want to assign an SRS to a geometry. You can do this with the `AssignSpatialReference` function.

```
geom.AssignSpatialReference(sr)
```

Again, this does not force the geometry to use the assigned spatial reference, but instead provides information about the SRS, whether right or wrong.

Hopefully, you now know how to use OSR to assign SRS information to your data so that GIS software knows how it relates spatially to other data.