Quick start for C API usage

This is a short introduction to the PROJ C API. In the following section we create two simple programs that illustrate how to transform points between two different coordinate systems, and how to convert between projected and geodetic (geographic) coordinates for a single coordinate system. Explanations for individual code sniplets and the full programs are provided.

See the following sections for more in-depth descriptions of different parts of the PROJ API or consult the `API reference <reference/index>` for specifics.

Before the PROJ API can be used it is necessary to include the `proj.h` header file. Here `stdio.h` is also included so we can print some text to the screen:

../../../examples/pj_obs_api_mini_demo.c

Let's declare a few variables that'll be used later in the program. Each variable will be discussed below. See the `reference for more info on data types <reference/datatypes>`.

../../../examples/pj_obs_api_mini_demo.c

For use in multi-threaded programs the `PJ_CONTEXT` threading-context is used. In this particular example it is not needed, but for the sake of completeness we demonstrate its use here.

../../../examples/pj_obs_api_mini_demo.c

Next we create the `PJ` transformation object `P` with the function `proj_create_crs_to_crs`.

../../../examples/pj_obs_api_mini_demo.c

Here we have set up a transformation from geographic coordinates to UTM zone 32N. In general, this is a transformation between two different coordinate reference systems (one of which is here in geographic coordinates). The related function `proj_create` can be used to set up transformations that are not available through the PROJ database, for instance for converting geodetic coordinates to a custom definition of a map projection.

`proj_create_crs_to_crs` takes as its arguments:

- the threading context `C` created above,
- a string that describes the source coordinate reference system (CRS),

- a string that describes the target CRS and
- an optional description of the area of use.

It is recommended to create one threading context per thread used by the program. This ensures that all `PJ` objects created in the same context will be sharing resources such as error-numbers and loaded grids.

If you are sure that `P` will only be used by a single program thread, you may pass `NULL` for the threading context. This will assign the default thread context to `P`.

The strings for the source and target CRS may be any of:

- PROJ strings, e.g. `+proj=longlat +datum=WGS84 +type=crs`,
- CRS identified by their code, e.g. `EPSG:4326` or `urn:ogc:def:crs:EPSG::4326`, or
- a well-known text (WKT) string, e.g.:

```
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["geodetic latitude (Lat)",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433]],
        AXIS["geodetic longitude (Lon)",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433]],
    USAGE[
        SCOPE["unknown"],
        AREA["World"],
        BBOX[-90,-180,90,180]],
    ID["EPSG",4326]]
```

Warning

The use of PROJ strings to describe a CRS is not recommended. One of the main weaknesses of PROJ strings is their inability to describe a geodetic datum, other than the few ones hardcoded in the `+datum` parameter.

`proj_create_crs_to_crs` will return a pointer to a `PJ` object, or a null pointer in the case of an error. The details of the error can be retrieved using `proj_context_errno`. See `errorhandling` for further details.

Now that we have a normalized transformation object in `P`, we can use it with `proj_trans` to transform coordinates from the source CRS to the target CRS, but first we will discuss the interpretation of coordinates.

By default, a `PJ` transformation object accepts coordinates expressed in the units and axis order of the source CRS, and returns transformed coordinates in the units and axis order of the target CRS.

For most geographic CRS, the units will be in degrees. In rare cases, such as EPSG:4807 / NTF (Paris), this can be grads. For geographic CRS defined by the EPSG authority, the order of coordinates is latitude first, longitude second. When using a PROJ string, the order is the reverse; longitude first, latitude second.

For projected CRS, the units may vary (metre, us-foot, etc.). For projected CRS defined by the EPSG authority, and with EAST / NORTH directions, the order might be easting first, northing second, or the reverse. When using a PROJ string, the order will be easting first, northing second, except if the `+axis` parameter modifies it.

If you prefer to work with a uniform axis order, regardless of the axis orders mandated by the source and target CRS, you can use the `proj_normalize_for_visualization` function.

`proj_normalize_for_visualization` takes a threading context and an existing `PJ` object, and generates from it a new `PJ` that accepts as input and returns as output coordinates using the traditional GIS order. That is, longitude followed by latitude, optionally followed by elevation and time for geographic CRS, and easting followed by northing for most projected CRS.

../../../examples/pj_obs_api_mini_demo.c

Next we create a `PJ_COORD` coordinate object, using the function `proj_coord`.

The following example creates a coordinate for 55°N 12°E (Copenhagen).

Because we have normalized the transformation object with `proj_normalize_for_visualization`, the order of coordinates is longitude followed by latitude, and the units are degrees.

../../../examples/pj_obs_api_mini_demo.c

Now we are ready to transform the coordinate into UTM zone 32, using the function `proj_trans`.

../../../examples/pj_obs_api_mini_demo.c

`proj_trans` takes as its arguments:

- a `PJ` transformation object,
- a `PJ_DIRECTION` direction, and
- the `PJ_COORD` coordinate to transform.

The direction argument can be one of:

- `PJ_FWD` -- "forward" transformation from source CRS to target CRS.
- `PJ_IDENT` -- "identity", return the source coordinate unchanged.

- `PJ_INV` -- "inverse" transformation from target CRS to source CRS.

It returns the new transformed `PJ_COORD` coordinate.

We can perform the transformation in reverse (from UTM zone 32 back to geographic) as follows:

../../../examples/pj_obs_api_mini_demo.c

Before ending the program, we need to release the memory allocated to our objects:

../../../examples/pj_obs_api_mini_demo.c

A complete compilable version of the example code can be seen below:

../../../examples/pj_obs_api_mini_demo.c

The following example illustrates how to convert between a CRS and geodetic coordinates for that CRS.

../../../examples/crs_to_geodetic.c