Cordex Data Extractor

The Cordex Data Extractor tool operates as an extractor independent of any specific formula. Projection involves mathematical processes, and our tool employs two distinct methods for this purpose. Typically, the tool employs the first method, switching to the second method automatically in the event of any encountered issues.

1- Using the Unidata netCDF-Java Library to execute these operations. For further details, please refer to the GitHub release page. Below, you can find the code we employ to determine the Grid Number (or Cell Number) using normal coordinates:

```
ucar.nc2.dt.grid.GridDataset gds =
ucar.nc2.dt.grid.GridDataset.open(filesNamePath);
ucar.nc2.dt.GridDatatype grid = gds.findGridDatatype(VarName);
ucar.nc2.dt.GridCoordSystem gcs = grid.getCoordinateSystem();
var index1 = gcs.findXYindexFromCoord(lat, lon, null);
var index2 = gcs.findXYindexFromLatLon(lat, lon, null);
```

If the code encounters an error, the tool will automatically convert your standard coordinates to rotated coordinates, based on the rotation axis of Cordex domains.

If the Cordex domain axis in your file differs from those on the cordex website, the tool will prompt you to select and confirm the appropriate domain after you have selected the files. The code used for the process of converting standard coordinates to rotated coordinates is as follows:

```
private static Point convert2Rotated_N(Point your, bool direct, Point pole)
    {
        var lon = your.X * Math.PI / 180; // Convert degrees to radians
        var lat = your.Y * Math.PI / 180;

        var theta = -90 + pole.Y; // Rotation around y-axis
        var phi = pole.X; // Rotation around z-axis

        phi = (phi * Math.PI) / 180; // Convert degrees to radians
        theta = (theta * Math.PI) / 180;
```

```
        var x = Math.Cos(lon) * Math.Cos(lat); // Convert from spherical to
cartesian coordinates

        var y = Math.Sin(lon) * Math.Cos(lat);

        var z = Math.Sin(lat);

        double x_new = 0;

        double y_new = 0;

        double z_new = 0;

        if (direct) // Regular -> Rotated

        {

           x_new = Math.Cos(theta) * Math.Cos(phi) * x + Math.Cos(theta) *
Math.Sin(phi) * y + Math.Sin(theta) * z;

           y_new = -Math.Sin(phi) * x + Math.Cos(phi) * y;

           z_new = -Math.Sin(theta) * Math.Cos(phi) * x - Math.Sin(theta) *
Math.Sin(phi) * y + Math.Cos(theta) * z;

        }
```

```
        else // Rotated -> Regular
        {
            phi = -phi;
            // theta = theta;

            x_new = Math.Cos(theta) * Math.Cos(phi) * x + Math.Sin(phi) * y +
Math.Sin(theta) * Math.Cos(phi) * z;
            y_new = -Math.Cos(theta) * Math.Sin(phi) * x + Math.Cos(phi) * y -
Math.Sin(theta) * Math.Sin(phi) * z;
            z_new = -Math.Sin(theta) * x + Math.Cos(theta) * z;
        }

        var lon_new = Math.Atan2(y_new, x_new); // Convert cartesian back to
spherical coordinates
        var lat_new = Math.Asin(z_new);
```

```
    lon_new = (lon_new * 180) / Math.PI; // Convert radians back to degrees

    lat_new = (lat_new * 180) / Math.PI;



    //   lon_new = (lon_new + 360) % 360;

    lon_new = lon_new + 180;

    Point grid_out = new Point() { X = lon_new, Y = lat_new };

    return grid_out;

}
```