FieldenMaps.info
Co-ordinate Converter API

JavaScript source documentation

Version 1.2.003
Latest update: 4 May 2009

Contents

Incorporating Co-ordinate Converter functions into your site

Source files required:

cconv_defs.js	-	Data type definitions
cconv_func.js	-	Basic mathematical, data and string manipulation functions
cconv_trans.js	-	Co-ordinate transformation functions
cconv_params.js	-	Projection and parameter initialisation
cconv_ostn02.js	-	OSTN02 transformation matrix data & initialisation
cconv_cty.js	-	OS County Series data initialisation

All these files will be found at http://www.fieldenmaps.info/cconv/web/

Instructions for adding the Co-ordinate Converter API into your site:

It is requested that you link dynamically to the source files in your code rather than download them and copy them to your own site. This way the latest version of the code will always be called when the page is loaded.

For each of the required source files listed above you will need to insert a line between the `<head>` and `</head>` tags in your (X)HTML code as follows:

```
<script language="JavaScript" type="text/javascript"
    src="http://www.fieldenmaps.info/cconv/web/[filename].js"></script>
```

Replace *[filename]* with each file's name as listed above. You will then be able to utilise in your own code all the functions, data types and constants detailed in this documentation.

You are advised to link to the source files in the order they are listed above; some of the functions in the later script files use functions from the previous script files and an error may occur if one is called before the other.

Source file 1: cconv_defs.js

Data type definitions

You can declare a new variable of a particular data type by using either of the following methods:

Method 1: Define the variable as being of a certain data type, then afterwards define its sub-elements:

```
var mycoord = new coord;  mycoord.eastings = 123456;  mycoord.northings = 654321;
```

Method 2: Define the variable and its sub-elements at the same time (in the order listed by each data type):

```
var mycoord = new coord(123456, 654321);
```

Data types

geodesic — *Used for storing a geodetic (geodesic) co-ordinate (in degrees or radians)*

.latitude	*geodetic latitude*
.longitude	*geodetic longitude*

geoextra — *A special extension of* geodesic*, allowing storage of extra information*

.latitude	*geodetic latitude*	*N.B. If defining a variable using method 2 above, you must pass latitude and longitude as one variable of type* geodesic.
.longitude	*geodetic longitude*	
.extra	*string, containing extra information*	

dmsdata — *Used for storing a value as degrees/minutes/seconds (without signs)*

.degrees	*degrees portion of data (can only be an integer between 0 and 90 inclusive)*
.minutes	*minutes portion of data (can only be an integer between 0 and 59 inclusive)*
.seconds	*seconds portion of data ($0 \leq x < 60$)*

coord — *Used for storing a plain eastings and northing co-ordinate*

.eastings	*co-ordinate eastings*
.northings	*co-ordinate northings*

coordx — *A special extension of* coord*, allowing storage of extra information*

.eastings	*co-ordinate eastings*
.northings	*co-ordinate northings*
.extra	*string, containing extra information*

gridref — *Used for storing a simple grid reference*

.square	*string, containing grid square reference*
.eastings	*string, containing eastings portion of reference*
.northings	*string, containing northings portion of reference*

ellipsoid — *Used for storing ellipsoid data*

.a	*ellipsoid semi-major ('a') axis in metres*	
.b	*ellipsoid semi-minor ('b') axis in metres*	
.e2	*eccentricity squared (e^2)*	*(if omitted when using method 2 above, these are calculated automatically)*
.se	*second eccentricity*	

datum — *Used for storing geodetic datum Helmert transformation parameters*

.name	*string, containing datum name*	
.ellip	*defining ellipsoid (data type* ellipsoid*)*	
.dX	*difference in X-axis (in metres)*	*Helmert transformation parameters (relative to WGS84 datum)*
.dY	*difference in Y-axis (in metres)*	
.dZ	*difference in Z-axis (in metres)*	
.rX	*rotation about X-axis (in arcseconds)*	
.rY	*rotation about Y-axis (in arcseconds)*	
.rZ	*rotation about Z-axis (in arcseconds)*	
.sf	*scale difference (in parts per million)*	

TMgriddata , CSgriddata , BNgriddata *Used for storing specific projection data*

.ellip	*reference ellipsoid (data type* ellipsoid*)*	
.F0	*scale factor on central meridian*	
.Lat0	*latitude of true origin, in radians*	
.Lon0	*longitude of true origin, in radians*	
.unit	*grid unit, as fraction of international metre*	
.unitname	*string, containing grid unit name*	
.FE	*grid easting of true origin*	
.FN	*grid northing of true origin*	
.e_min	*minimum easting of grid area*	*(used for determining validity of co-ordinate entry and results)*
.e_max	*maximum easting of grid area*	
.n_min	*minimum northing of grid area*	
.n_max	*maximum northing of grid area*	

MeridianData *Used for storing county origins*

.Name	*name of projection origin*	
.Lat0	*latitude of projection origin, in radians*	
.Lon0	*longitude of projection origin, in radians*	
.e_min	*minimum relevant easting*	*(not currently in use)*
.e_max	*maximum relevant easting*	
.n_min	*minimum relevant northing*	
.n_max	*maximum relevant northing*	

CountyData *Used for storing counties*

.Name	*county name*
.Meridian	*index number of origin used for county*
.sdims	*sheet dimensions of county (data type* Sheet_WH*)*
.cfg	*integer from 1 to 7 describing which type of sheets are relevant to county (1 = 1:2500 shts; 2 = 6-inch quarter shts; 4 = 6-inch full shts; and sums thereof)*
.Sheets	*array, containing sheet data (data type* CountySheet*)* *N.B. This sub-element cannot be defined using method 2 above; if using method 2 you must omit it (a blank array is created automatically) and then assign values to it separately.*

CountySheet *Used for storing county sheets*

.ShtNum	*string, containing sheet reference number*
.ShtW	*distance from projection origin to west sheet line*
.ShtS	*distance from projection origin to south sheet line*
.deriv	*string, describing derivatives of full sheet | (not currently in use)*

Sheet_WH *Used for defining sheet dimensions of each county*

.width	*width of sheet*
.height	*height of sheet*

ctyshtres *Used for storing the result of a Geodetic-to-County Series transformation*

.cty	*index number of county within CtyList*
.sht	*index number of sheet within CtyList[cty].Sheets (not the same as sheet number)*
.rwe	*raw easting co-ordinate relative to projection origin for county*
.rwn	*raw northing co-ordinate relative to projection origin for county*

Source file 2: cconv_func.js

Basic mathematical, data and string manipulation functions

Degree/Radian conversion functions

function *deg2rad(degrees)*
Returns *degrees* converted to radians

function *rad2deg(radians)*
Returns *radians* converted to degrees

function *sec2rad(arcseconds)*
Returns *arcseconds* converted to radians

function *rad2sec(radians)*
Returns *radians* converted to arcseconds

function *sec2dms(arcseconds)*
Returns *arcseconds* converted to degrees, minutes and seconds (output is of type *dmsdata*)

Trigonometrical functions (input/output in radians)

function *Sec(number)*
Returns the secant (1/cosine) of *number*

function *arcsin(number)*
Returns the arcsine (inverse sine) of *number*

function *sinh(number)*
Returns the hyperbolic sine of *number*

function *cosh(number)*
Returns the hyperbolic cosine of *number*

function *tanh(number)*
Returns the hyperbolic tangent of *number*

function *sech(number)*
Returns the hyperbolic secant of *number*

function *arsinh(number)*
Returns the hyperbolic arcsine of *number*

function *artanh(number)*
Returns the hyperbolic arctangent of *number*

function *arsech(number)*
Returns the hyperbolic arcsecant of *number*

String/Number manipulation functions

function *RoundDec(number, places)*
Returns *number* rounded to *places* decimal places

function *stringmask(input_string, mask)*
Returns only those characters of *input_string* which also occur in *mask*

function *fivefig(input_string)*
Returns a five-character string containing *input_string* with the addition of trailing zeroes

function *smallnum2string(input_number)*
Returns a string containing *input_number* (where *input_number* is less than 1*)*
- fixes JavaScript's habit of converting tiny numbers to standard form (eg. '7e-7') in strings

Validation functions

function *isvalidgridref(square, easting, northing, projection_data, squares_array)*
Returns *true* if *square*, *easting* and *northing* (all strings) resolve as a valid grid reference, given *projection_data* (type *TMgriddata*, *CSgriddata* or *BNgriddata*) and *squares_array* (two-dimensional array of grid squares); otherwise, returns *false*

function *isvalidcoord(easting, northing, projection_data)*
Returns *true* if *easting* and *northing* result in a valid co-ordinate, given *projection_data* (type *TMgriddata*, *CSgriddata* or *BNgriddata*); otherwise, returns *false*

The functions below use the following arbitrary bounds for determining validity of co-ordinates:
Great Britain: [49°N ≤ latitude < 62°N], [10°W < longitude < 4°E]
Ireland: [51°N ≤ latitude < 56°N], [12°W < longitude ≤ 4°W]

function *isvalidgeodms(lat_d, lat_m, lat_s, lon_d, lon_m, lon_s, lon_sign)*
Returns *true* if *lat_d*, *lat_m* and *lat_s* (in degrees, minutes and seconds respectively) is a valid latitude, and *lon_d*, *lon_m* and *lon_s* (in degrees, minutes and seconds respectively) multiplied by *lon_sign* (must be 1 or -1) is a valid longitude in Great Britain; otherwise, returns *false*

function *isvalidgeo(latitude, longitude, longitude_sign)*
Returns *true* if *latitude* (in degrees) is a valid latitude and *longitude* (in degrees) multiplied by *longitude_sign* (must be 1 or -1) is a valid longitude in Great Britain; otherwise, returns *false*

function *isvalidigeodms(lat_d, lat_m, lat_s, lon_d, lon_m, lon_s)*
Returns *true* if *lat_d*, *lat_m* and *lat_s* (in degrees, minutes and seconds respectively) results in a valid latitude, and *lon_d*, *lon_m* and *lon_s* (in degrees, minutes and seconds respectively) results in a valid longitude for Ireland (longitude values should all be input as positive numbers even though all longitudes for Ireland are negative); otherwise, returns *false*

function *isvalidigeo(latitude, longitude)*
Returns *true* if *latitude* (in degrees) is a valid latitude and *longitude* (in degrees and must be negative) is a valid longitude in Ireland; otherwise, returns *false*

Source file 3: cconv_trans.js

Co-ordinate transformation functions

Simple Geodetic Datum Conversion

function *Geo2Geo(latitude, longitude, origin_datum, desired_datum)*

Converts a given *latitude* and *longitude* (in radians) in *origin_datum* (data type *datum*) to a latitude and longitude in *desired_datum* (data type *datum*), using a 3- or 7-parameter Helmert transformation. Generally speaking the results are accurate to a few metres depending on the quality of the original survey network, the density of the network used to ascertain the datum transformation parameters, and the size of the area relevant to the transformation. The result is returned in radians as data type *geodesic*.

N.B. This should **not** generally be used for conversions between WGS84/ETRS89 (GPS-compatible) and OSGB36 **or** OS Ireland 65 datums. Instead use the special geodetic datum conversion functions below which use more accurate transformations (OSTN02 for OSGB36, where available, and 3rd-order polynomial for OS Ireland 65).

Special Geodetic Datum Conversion

function *OSGB362WGS84(latitude, longitude)*

Converts a given *latitude* and *longitude* (in radians) in the OSGB36 datum to a latitude and longitude in the WGS84/ETRS89 (GPS-compatible) datum, using the OSTN02 transformation where available, and the Helmert transformation elsewhere. The result is returned in radians as data type *geoextra*.

function *WGS842OSGB36(latitude, longitude)*

Converts a given *latitude* and *longitude* (in radians) in the WGS84/ETRS89 (GPS-compatible) datum to a latitude and longitude in the OSGB36 datum, using the OSTN02 transformation where available, and the Helmert transformation elsewhere. The result is returned in radians as data type *geoextra*.

function *get_osgbshift(easting, northing)*

(Function integral to the WGS84/ETRS89<>OSGB36 conversions). Returns the easting and northing shifts required to convert a given *easting* and *northing* (computed using OSGB National Grid parameters but on WGS84/ETRS89 datum & ellipsoid) to OSGB National Grid easting and northing, using the OSTN02 transformation. If the *easting* and *northing* are out of range of the transformation, the easting and northing shifts are returned as 0. The result is returned as data type *coordx*.

Note: For the above three functions, the extra information included in the result is a string containing the transformation method employed to achieve the result:

- "OSTN02" - for when an entirely OSTN02 conversion has taken place;
- "OSTN02/Helmert" - for the area around the edge of the OSTN02 transformation area (*get_osgbshift* computes the shift in such a way as to achieve a smooth join between the two transformation types);
- "Helmert" - when an entirely standard Helmert transformation has taken place (in this case *get_osgbshift* cannot compute an easting/northing shift).

function *OSI652WGS84(latitude, longitude)* *Please note this is spelt with a capital letter 'i' not a number '1'*

Converts a given *latitude* and *longitude* (in radians) in the OS Ireland 65 datum to a latitude and longitude in the WGS84/ETRS89 (GPS-compatible) datum, using the third-order polynomial regression transformation published by OSi. The result is returned in radians as data type *geodesic*.

function *WGS842OSI65(latitude, longitude)* *Please note this is spelt with a capital letter 'i' not a number '1'*

Converts a given *latitude* and *longitude* (in radians) in the WGS84/ETRS89 (GPS-compatible) datum to a latitude and longitude in the OS Ireland 65 datum, using the third-order polynomial regression transformation published by OSi. The result is returned in radians as data type *geodesic*.

function *get_osishift(latitude, longitude)*

(Function integral to the WGS84/ETRS89<>OS Ireland 65 conversions). Returns the latitude and longitude shifts required to convert a given *latitude* and *longitude* (in radians) in the OS Ireland 65 datum to a latitude/longitude in the WGS84/ETRS89 (GPS-compatible) datum, using the official OSi third-order polynomial regression transformation. The result is returned in radians as data type *geodesic*.

Conversions between geodetic and projected co-ordinates

Remember! When converting from geodetic co-ordinates to projected co-ordinates, ensure that the geodetic co-ordinates are in the same datum as the projection which you are converting to is based upon.

Transverse Mercator Projection

Suitable for an extremely wide range (c. 7,000km either side of origin longitude, and latitudes up to 85°) with sub-millimetre accuracy in both forward and inverse conversions.

function *Geo2TM(latitude, longitude, grid_data)*

Converts a given *latitude* and *longitude* (in radians) to a Transverse Mercator easting and northing. Also input *grid_data* of type *TMgriddata*. The result is returned as data type *coord*.

function *TM2Geo(easting, northing, grid_data)*

Converts a given Transverse Mercator *easting* and *northing* to latitude and longitude. Also input *grid_data* of type *TMgriddata*. The result is returned in radians as data type *geodesic*.

Cassini Projection

Suitable for a relatively limited range (c. 10° either side of origin longitude, and latitudes up to 75°) with millimetre accuracy in both forward and inverse conversions.

function *Geo2CS(latitude, longitude, grid_data)*

Converts a given *latitude* and *longitude* (in radians) to a Cassini projection easting and northing. Also input *grid_data* of type *CSgriddata*. The result is returned as data type *coord*.

function *CS2Geo(easting, northing, grid_data)*

Converts a given Cassini projection *easting* and *northing* to latitude and longitude. Also input *grid_data* of type *CSgriddata*. The result is returned in radians as data type *geodesic*.

Bonne Projection

Suitable for a very wide range (c. 70° either side of origin longitude, and latitudes up to 85°) with millimetre accuracy in both forward and inverse conversions.

function *Geo2BN(latitude, longitude, grid_data)*

Converts a given *latitude* and *longitude* (in radians) to a Bonne projection easting and northing. Also input *grid_data* of type *BNgriddata*. The result is returned as data type *coord*.

function *BN2Geo(easting, northing, grid_data)*

Converts a given Bonne projection *easting* and *northing* to latitude and longitude. Also input *grid_data* of type *BNgriddata*. The result is returned in radians as data type *geodesic*.

Simplified GB National Yard Grid Conversion

Simple linear scaling to convert between two grid systems on the same projection but with differing units.

function *NG2YG(easting, northing)*

Converts a given OSGB National Grid *easting* and *northing* to an OSGB National Yard Grid easting and northing. The result is returned as data type *coord*.

function *YG2NG(easting, northing)*

Converts a given OSGB National Yard Grid *easting* and *northing* to an OSGB National Grid easting and northing. The result is returned as data type *coord*.

Ordnance Survey County Series (Great Britain & Isle of Man)

Conversions for the various county series projections in use in Great Britain a & Isle of Man, utilising Cassini Projection conversion functions and thus currently suitable only for areas of limited extent.

function *Geo2Cty(latitude, longitude)*

Converts a given *latitude* and *longitude* (in radians) in the OSGB36 datum to raw OS County Series results (county-index, sheet-index, raw projection easting, raw projection northing). The result is returned as an array of data type *ctyshtres* (q.v.).

function *Geo2CtyRaw(latitude, longitude, county_index)*

Converts a given *latitude* and *longitude* (in radians) in the OSGB36 datum to raw OS County Series co-ordinates, given *county_index*, the index number of the county relevant to the area. The result is returned in feet as data type *coord*.

function *Geo2CtyString(latitude, longitude)*

Converts a given *latitude* and *longitude* (in radians) in the OSGB36 datum to OS County Series results in human-readable format. The result is returned as an array of strings listing the sheets upon which the point occurs.

function *CtyRaw2Geo(easting, northing, county_index)*

Converts a given *easting* and *northing* (in feet, relative to the projection origin of the county identified by *county_index*) to a latitude and longitude in the OSGB36 datum. The result is returned in radians as data type *geodesic*.

Grid Reference Conversions

Conversion between raw co-ordinates and grid references.

function *conv_EN2GR(easting, northing, squares_array)*

Converts a given *easting* and *northing* to a Grid Reference, given *squares_array* (a two-dimensional array of 100km grid square strings). The result is returned as data type *gridref*.

function *conv_GR2EN(square, east, north, squares_array)*

Converts a grid reference (consisting of three parts: *square*, *east* and *north*) to a pure easting and northing, given *squares_array* (a two-dimensional array of 100km grid square strings). The result is returned as data type *coord*.

Miscellaneous/internal functions

function *calc_M(lat_diff, lat_sum, n, b, F0)*

(Function integral to most conversions between latitude/longitude and projected easting/northing). Returns the distance along the central meridian of projection between the projection origin and a given latitude, given *lat_diff* (latitude **minus** latitude of projection origin, in radians), *lat_sum* (latitude **plus** latitude of projection origin, in radians), *n* (ellipsoidal $(a-b)\div(a+b)$), *b* (ellipsoidal semi-minor axis) and *F0* (scale factor on central meridian of projection). If *lat_diff* and *lat_sum* are identical, the result is the distance from the equator to that latitude along the central meridian of projection. The result is returned in the same units as the ellipsoidal parameter *b*.

function *Geo2CSsimple(easting, northing, grid_data)*
function *CS2Geosimple(latitude, longitude, grid_data)*

Internal functions for partial conversion between geodetic and Cassini projection co-ordinates.

function *Geo2BNsimple(easting, northing, grid_data)*
function *BN2Geosimple(latitude, longitude, grid_data)*

Internal functions for partial conversion between geodetic and Bonne projection co-ordinates.

Source file 4: cconv_params.js

Projection and parameter initialisation

Constants

pi ≈ 3.141592653589793	*Mathematical constant π*
FO1 = 0.3048007491	*'Foot of O1' - basis of measurement for early OS maps*

String masks

numerics = '0123456789'	*String mask for numerics*
numerics_d = '0123456789.'	*String mask for numerics and decimal points*
numerics_a = '0123456789ABCW'	*String mask for numerics and the letters 'A', 'B' , 'C' and 'W'*
alphanumerics = 'A..Za..z0..9'	*String mask for alphanumeric characters*
alphabetics = 'A..Za..z'	*String mask for alphabetical characters*

Ellipsoids

Airy (type *ellipsoid*)	*Airy 1830 ellipsoid* (used for Great Britain/Ireland)
AiryMod (type *ellipsoid*)	*Airy Modified ellipsoid* (used for Ireland)
GRS80 (type *ellipsoid*)	*GRS80 ellipsoid* (Current international geodetic ellipsoid)
Int24 (type *ellipsoid*)	*International 1924 ellipsoid* (Former international ellipsoid)

Datum parameters for simple 3- or 7-parameter Helmert transformations

WGS84 (type *datum*)	*WGS84/ETRS89 international GPS-compatible datum*
ED50 (type *datum*)	*European Datum 1950, suitable for GB, Ireland & Channel Is.*
OSGB36 (type *datum*)	*Ordnance Survey Great Britain 1936 datum*
OSI65 (type *datum*) *Please note this is spelt with a capital letter 'i' not a number '1'*	*Ordnance Survey Ireland 65 datum*

Projection parameters

OSNG (type *TMgriddata)*	*Ordnance Survey GB National Grid* *- used as the basis for OS maps of Great Britain since the 1940s* *- based on OSGB36 datum*
OSYG (type *TMgriddata)*	*Ordnance Survey GB National Yard Grid* *- used as the basis for OS maps of Great Britain in the 1930s* *- based on OSGB36 datum*
OSIG (type *TMgriddata)*	*Ordnance Survey Irish Grid* *- used as the basis for OS maps of Ireland since the 1960s* *- based on OS Ireland 65 datum*
ITMG (type *TMgriddata*)	*Irish Transverse Mercator (ITM) Grid* *- currently being introduced across Irish mapping* *- based on WGS84/ETRS89 datum*
CDEL (type *CSgriddata*)	*Ordnance Survey GB Cassini projection (Delamere origin)* *- used for OS maps of Eng. & Wales from 1870s (+ Scotland from 1920s) to 1940s* *- based on OSGB36 datum*

Projection parameters (continued)

WOCG (type *CSgriddata*)	*War Office Cassini Grid* *- used on War Office maps of Great Britain from the 1920s to the 1940s* *- based on OSGB36 datum*
WOIG (type *CSgriddata*)	*War Office Irish Grid* *- used on War Office maps of Ireland from the 1920s to the 1950s* *- based on OS Ireland 65 datum*
BONS (type *BNgriddata*)	*Ordnance Survey Bonne (Scotland) projection* *- used as the basis for OS maps of Scotland from the 1850s to the 1920s* *- based on OSGB36 datum*
BONI (type *BNgriddata*)	*Ordnance Survey Bonne (Ireland) projection* *- used as the basis for OS maps of Ireland from the 1850s to the 1990s* *- based on OS Ireland 65 datum*
UTM29_ED (type *TMgriddata*)	*Universal Transverse Mercator, Zone 29, ED50 datum*
UTM30_ED (type *TMgriddata*)	*Universal Transverse Mercator, Zone 30, ED50 datum*
UTM31_ED (type *TMgriddata*)	*Universal Transverse Mercator, Zone 31, ED50 datum* *- used on some international maps of the British Isles*
UTM29_WGS (type *TMgriddata*)	*Universal Transverse Mercator, Zone 29, WGS84 datum*
UTM30_WGS (type *TMgriddata*)	*Universal Transverse Mercator, Zone 30, WGS84 datum*
UTM31_WGS (type *TMgriddata*)	*Universal Transverse Mercator, Zone 31, WGS84 datum* *- used for international grid referencing of the British Isles*

(Additionally, OSNGgps of type TMgriddata *is also defined, but for internal calculation use only)*

Grid Reference parameters

OSNG_GS (2-dim. array)	*OS GB National Grid 100km squares (2-letter reference)*
OSNG_NS (2-dim. array)	*OS GB National Grid 100km squares (numeric reference)*
OSIG_GS (2-dim. array)	*OS Irish Grid 100km squares (single-letter reference)*
WOCG_GS (2-dim. array)	*War Office Cassini Grid 100km squares (2-letter ref.)*
WOIG_GS (2-dim. Array)	*War Office Irish Grid 100km squares (2-letter ref.)*

Miscellaneous conversion parameters

osiA, osiB (2-dim. arrays)	*Coefficients for third-order polynomial transformation between Ordnance Survey Ireland 65 latitude/longitude and WGS84/ETRS89 (GPS-compatible) latitude/longitude.*
osik0, osilatm, osilonm	*More parameters for OS Ireland 65<>WGS84/ETRS89 latitude/longitude conversion.*

Source file 5: cconv_ostn02.js

OSTN02 transformation matrix data and initialisation

The code in this file initialises into memory a low-resolution version of the OSTN02 transformation matrix, for conversions between WGS84/ETRS89 (GPS-compatible) geodetic co-ordinates and OSGB National Grid co-ordinates. To initialise the data, the function *initostn02* is called when the source file is loaded.

The original full-resolution OSTN02 transformation matrix defines easting and northing 'shifts' for the corner of each kilometre square in Great Britain, also extending to some 10 kilometres off-shore. These 'shifts' convert WGS84/ETRS89 Transverse Mercator co-ordinates (directly calculated from WGS84/ETRS89 geodetic co-ordinates using the non-standard projection *OSNGgps*) to OSGB National Grid co-ordinates. The reverse transformation is achieved by iteration.

Currently Co-ordinate Converter makes use of this in the functions *WGS842OSGB36* and *OSGB362WGS84* to convert between WGS84/ETRS89 and OSGB36 geodetic co-ordinates.

In this version of the Co-ordinate Converter code, the OSTN02 data matrix is at one-third of full resolution in each dimension, for the sake of keeping the download size relatively manageable (c. 224KB). The global variable *ostnres* contains the resolution of the data in kilometres. In this build version of the code, *ostnres* is equal to 3.

Using a lower-resolution version of the OSTN02 data does cause a very slight inaccuracy, the discrepancy being up to 1cm when compared with the official, full-resolution transformation. For almost all purposes, however, this is more than accurate enough.

The easting and northing 'shifts' required to effect the transformation are stored in two two-dimensional arrays: *ostn02e* and *ostn02n*. The bounds and indices of the arrays *ostn02e* and *ostn02n* are determined by the resolution of the data. Only those points which are covered by the transformation are defined; if a direct query is made to one of these arrays any points which fall outside the transformation area will return 'undefined'.

Full technical detail on the transformation is available in the 'Transformations and OSGM02 user guide' PDF contained within OS's official OSTN02 zip file, downloadable from [http://www.ordnancesurvey.co.uk/oswebsite/gps/docs/OSTN02_OSGM02files.zip]

In summary, the following are defined:

ostnres	*resolution of the OSTN02 data, in kilometres*
ostn02e	*2-dimensional array containing OSTN02 easting shifts, in millimetres*
ostn02n	*2-dimensional array containing OSTN02 northing shifts, in millimetres*

Source file 6: cconv_cty.js

OS County Series data initialisation

The code in this file initialises into memory the projection parameters for the Ordnance Survey large-scale County Series (Great Britain & Isle of Man) and the co-ordinates of their constituent base sheets.

There are a total of 51 projection origins (numbered 0-50); these are stored in an array named *MeridList* as data type *MeridianData*. There are a total of 115 'counties' (numbered 0-114); these are stored in an array named *CtyList* as data type *CountyData*. The sheets themselves are stored in an array within each member of *CtyList* (namely *CtyList[n].Sheets*) as data type *CountySheet*.

The code in this file also defines the constants *s6in* and *s25in* (both data type *Sheet_WH*) – the dimensions in feet of the coverage of 6-inch full sheets and 1:2500 sheets respectively.

In order to effect the initialisation, the function *initctylist* is called when the source file is loaded.

Projection Origins *(MeridList)*

0 Ben Auler
1 Ben Clibrig
2 Ben Cleugh
3 Black Down
4 Brandon Down
5 Broadfield
6 Bengray
7 Bleasdale
8 Brown Carrick
9 Cleisham
10 Cairn Glasher
11 Caerloch
12 Corkmulaw
13 Cruach-na-Sleagh
14 Craigowl
15 Cyrn-y-Brain
16 Dumbarton Castle
17 Danbury Church Spire
18 Derrington Great Law
19 Dunnet Head
20 Ditchling
21 Dunnose
22 Dunrich
23 Edinburgh Castle
24 Foula
25 Forest Hill
26 Findlay's Seat
27 Hensbarrow
28 Hart Fell
29 Highgate
30 Hollingbourne
31 High Pike
32 Knock of Luce
33 Lanark Church Spire
34 Leith Hill Tower
35 Llangeinor
36 Mount Airy
37 Nantwich Church Tower
38 Rubers Law
39 Rippon Tor
40 South Berule
41 Sandhope Heights
42 Simonside
43 Scour-na-Lapich
44 St. Paul's
45 The Buck
46 Traprean Law
47 West Lomond
48 York Minster
49 Otley Church Tower
50 Stafford Castle

County Series *(CtyList)*

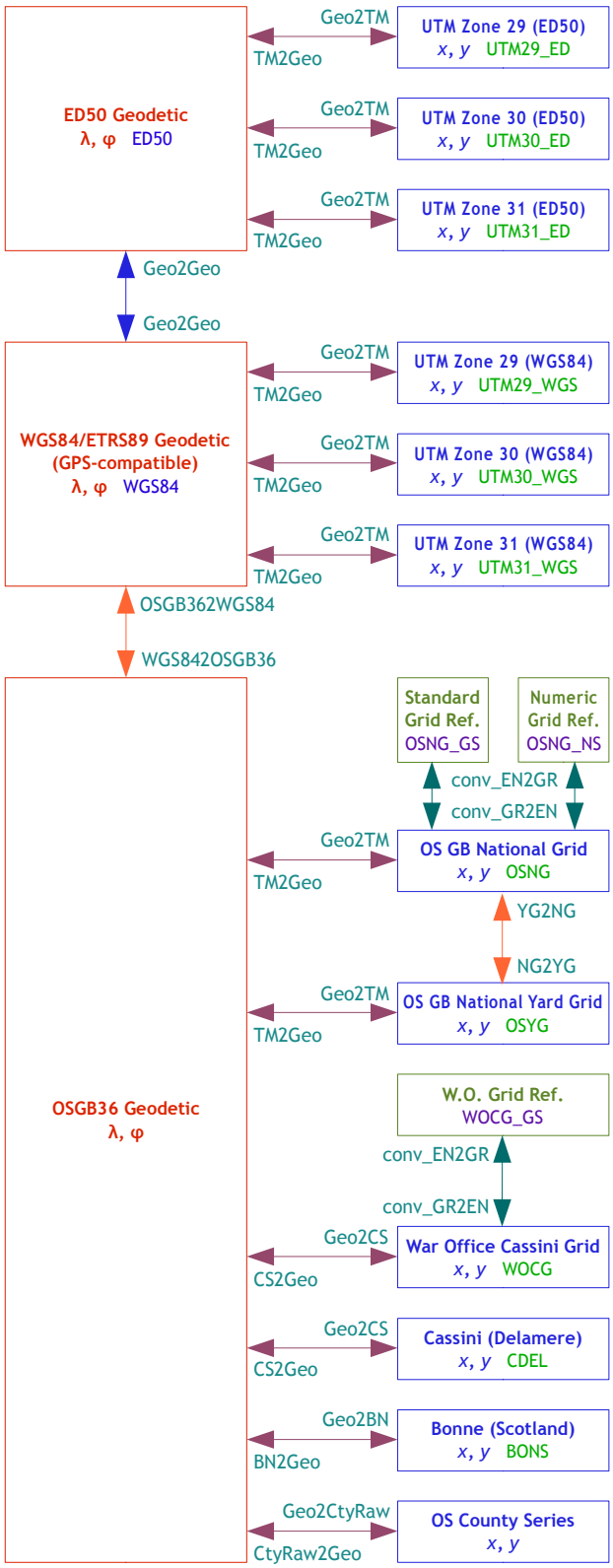
0 Anglesey
1 Bedfordshire
2 Berkshire
3 Brecknockshire
4 Buckinghamshire
5 Caernarvonshire
6 Cambridgeshire
7 Cardiganshire
8 Carmarthenshire
9 Cheshire
10 Cornwall
11 Cumberland
12 Denbighshire
13 Derbyshire
14 Devon
15 Dorset
16 Durham
17 Essex (1st Ed./Rev. 1862-96)
18 Essex (New Series 1913-)
19 Flintshire
20 Glamorgan
21 Gloucestershire
22 Hampshire & Isle of Wight
23 Herefordshire
24 Hertfordshire
25 Huntingdonshire
26 Isle of Man
27 Kent
28 Lancashire
29 Leicestershire
30 Lincolnshire
31 London (First Editions c.1850s, 6-inch sheets)
32 London (First Editions c.1850s, 1:2500 sheets)
33 London (Edition of 1894-96, 6-inch sheets)
34 London (Edition of 1894-96, 1:2500 sheets)
35 London (1915-) (Numbered sheets)
36 London (1915-) (Lettered 6-inch sheets)
37 Merionethshire
38 Middlesex
39 Monmouthshire
40 Montgomeryshire
41 Norfolk
42 Northamptonshire
43 Northumberland (1st Ed./Rev. 1855-97)
44 Northumberland (New Series 1913-)
45 Nottinghamshire
46 Oxfordshire
47 Pembrokeshire
48 Radnorshire
49 Rutland
50 Shropshire
51 Somerset
52 Staffordshire (Partial issues 1861-63)
53 Staffordshire (1875-)
54 Suffolk (Partial issues 1861-74)
55 Suffolk (1875-)
56 Surrey
57 Sussex
58 Tyneside (1:2500 sheets, 1894-1912)
59 Warwickshire
60 Westmorland
61 Wiltshire
62 Worcestershire
63 Yorkshire
64 Aberdeenshire
65 Angus (Forfarshire)
66 Argyllshire
67 Ayrshire
68 Banffshire
69 Berwickshire
70 Buteshire
71 Caithness
72 Clackmannanshire
73 Dumfriesshire
74 Dunbartonshire (1st Ed. 1858-61)
75 Dunbartonshire (2nd Ed. 1894-98)
76 Dunbartonshire (det.) (2nd Ed. 1894-98)
77 Dunbartonshire (New Series 1914-)
78 Edinburghshire (6-in. 1st Ed. 1850-52)
79 Edinburghshire (Midlothian) (1892-)
80 Elginshire (Morayshire)
81 Fifeshire (6-inch 1st Ed. 1852-55)
82 Fifeshire (1893-)
83 Haddingtonshire (6-in. 1st Ed. 1852-54)
84 Haddingtonshire (East Lothian) (1892-)
85 Inverness-shire
86 Isle of Lewis
87 Isle of Skye
88 Kincardineshire (1st Ed. 1863-1865)
89 Kincardineshire (New Series 1899-)
90 Kinross-shire (6-inch 1st Ed. 1853-54)
91 Kinross-shire (1894-)
92 Kirkcudbrightshire (6in. 1st Ed. 1845-50)
93 Kirkcudbrightshire (1893-)
94 Lanarkshire (1st Edn. 1856-59)
95 Lanarkshire (1892-)
96 Linlithgow (1st Ed./Rev. 1854-96)
97 Linlithgow (W.Loth.) (New Series 1913-)
98 Nairnshire
99 Orkney
100 Outer Hebrides
101 Peeblesshire (1st Edn. 1855-58)
102 Peeblesshire (1897-)
103 Perthshire
104 Renfrewshire
105 Ross & Cromarty
106 Roxburghshire (1st Ed./Rev. 1856-98)
107 Roxburghshire (New Series 1916-)
108 Selkirkshire
109 Shetland (Zetland)
110 Stirlingshire (1st Ed./Rev. 1858-96)
111 Stirlingshire (New Series 1913-)
112 Sutherland
113 Wigtownshire (6-inch 1st Edn. 1843-47)
114 Wigtownshire (1892-)

Practical application and transformation procedures

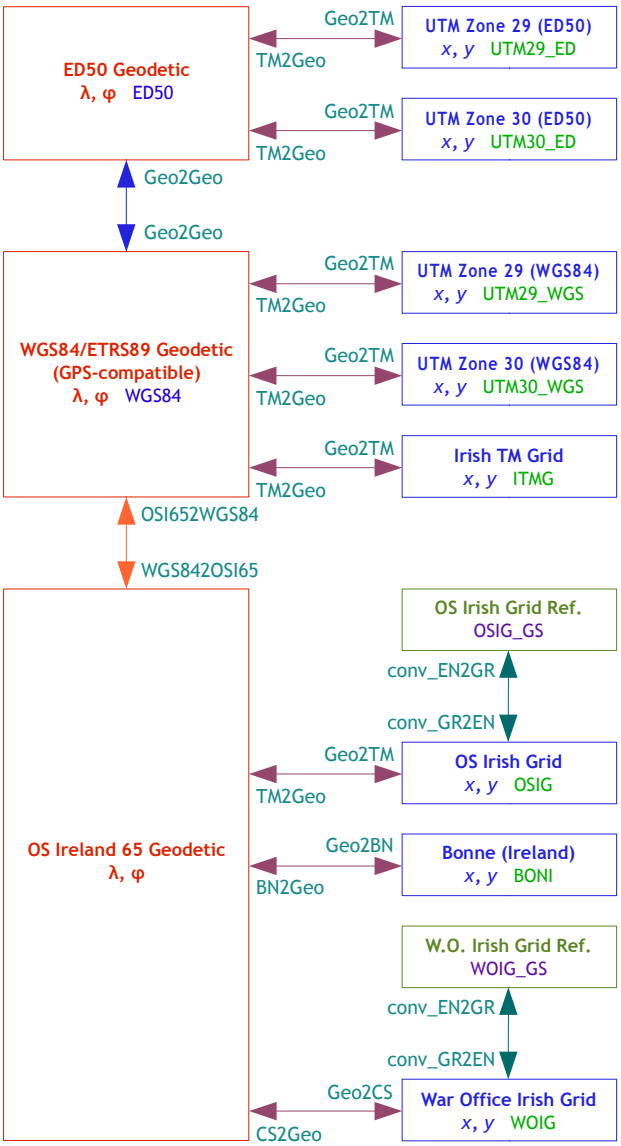
How to utilise the Co-ordinate Converter API functions

The diagrams below show which direct transformations are possible, along with the required transformation function, plus datum, projection and 'square array' parameters needed.
Attempting any other direct transformation will give inaccurate results.

Great Britain



Ireland



Transformations

Remember that *latitude* and *longitude* should be passed to a transformation function in **radians**.
Likewise the result of a transformation to *latitude* and *longitude* is returned in **radians**.

Refer to the diagrams on the previous page to determine the correct transformation process for your requirements.

Simple conversions requiring a single transformation stage can be achieved by referring to the relevant function in the '**cconv_trans.js**' section on pages 6-8.

For desired conversions with no direct path, transformation must take place via one or more intermediate co-ordinate system. For example, to convert from Great Britain War Office Grid Reference to OSGB National Grid co-ordinates, follow the most direct route possible:

1. Convert War Office Grid Ref. to War Office Grid co-ordinates using the conv_GR2EN function.
2. Convert that co-ordinate result to OSGB36 geodetic co-ordinates using the CS2Geo function.
3. Convert that geodetic result to OSGB National Grid co-ordinates using the Geo2TM function.

Transformations to and from OS County Series co-ordinates & sheet references are slightly different:

If using the Geo2CtyRaw transformation function you must supply a *latitude* and *longitude* (OSGB36) and, in addition, the index number of the county for which you require the raw co-ordinates (see list of county index numbers 2 pages back).

If you do not know the county but do know which projection origin for which you require the relevant easting and northing, you should refer to the list of projection origins and then use your own function to search through *CtyList[n].Meridian* until you find a match for the index number of the required projection origin. That county (*n*) should then be used as your county index number. An example is given in Code Example 9 on page 17.

The reverse transformation – CtyRaw2Geo – requires a raw easting and northing, and additionally the index number of the county relevant to the point being converted. Again, if the county is not known but the projection origin is, follow the procedure explained above to find a suitable county index number.

There are some useful alternative options for converting from geodetic to County Series co-ordinates.

The Geo2Cty transformation function is probably the most versatile. Supplied with a *latitude* and *longitude* (OSGB36) it returns results as an array of data type *ctyshtres* (q.v.). Should you require, you can then use your own functions to retrieve the relevant sheet co-ordinate data and calculate internal sheet co-ordinates of the point you have converted. An example is given in Code Example 10 on page 17.

A special transformation function (Geo2CtyString) has also been created. Given a *latitude* and *longitude* (OSGB36) it returns an array containing human-readable strings containing the 'sheet number(s)' of the base sheet(s) upon which the point occurs (sheet number in this context includes the county name).

Code Examples

Simple examples of JavaScript code utilising Co-ordinate Converter API functions

Code Example 1:

Converts OSGB National Grid co-ordinates [402040; 201835] to OSGB36 geodetic co-ordinates.

```
    // convert OSGB National Grid co-ordinates to OSGB36 geodetic co-ordinates
  var georesult = TM2Geo(402040, 201835, OSNG);
    // display result
  alert ('Lat. ' + rad2deg(georesult.latitude) + '; Lon. ' + rad2deg(georesult.longitude));
```

Code Example 2:

Converts WGS84/ETRS89 (GPS-compatible) geodetic co-ordinates [51.5°N; 1.5°W] to OSGB National Grid co-ordinates.

```
    // first convert WGS84/ETRS89 geodetic co-ordinates to OSGB36 geodetic co-ordinates
  var georesult = WGS842OSGB36(deg2rad(51.5), deg2rad(-1.5));
    // then convert OSGB36 geodetic co-ordinates to OSGB National Grid co-ordinates
  var cooresult = Geo2TM(georesult.latitude, georesult.longitude, OSNG);
    // display result
  alert ('Eastings = ' + cooresult.eastings + '; Northings = ' + cooresult.northings);
```

Code Example 3:

Converts Cassini (Delamere) projection co-ordinates [32,760 ft. East; 217,840 ft. South] to OSGB National Grid co-ordinates.

```
    // first convert Cassini co-ordinates to OSGB36 geodetic co-ordinates
  var georesult = CS2Geo(32760, -217840, CDEL);
    // then convert OSGB36 geodetic co-ordinates to OSGB National Grid co-ordinates
  var cooresult = Geo2TM(georesult.latitude, georesult.longitude, OSNG);
    // display result
  alert ('Eastings = ' + cooresult.eastings + '; Northings = ' + cooresult.northings);
```

Code Example 4:

Converts WGS84/ETRS89 (GPS-compatible) geodetic co-ordinates [56.25°N; 3.5°W] to Bonne (Scotland) projection co-ordinates.

```
    // first convert WGS84/ETRS89 geodetic co-ordinates to OSGB36 geodetic co-ordinates
  var georesult = WGS842OSGB36(deg2rad(56.25), deg2rad(-3.5));
    // then convert OSGB36 geodetic co-ordinates to Bonne (Scotland) co-ordinates
  var cooresult = Geo2BN(georesult.latitude, georesult.longitude, BONS);
    // set display strings relevant to negative or positive easting/northing values
  if (cooresult.eastings < 0) {var eorw = ' ft W'; } else {var eorw = ' ft E';}
  if (cooresult.northings < 0) {var nors = ' ft S'; } else {var nors = ' ft N';}
    // display result
  alert (Math.abs(cooresult.eastings) + eorw + '; ' + Math.abs(cooresult.northings) + nors);
```

Code Example 5:

Converts War Office Grid Reference [vP 397 552] to WGS84/ETRS89 (GPS-compatible) geodetic co-ordinates.

```
    // first convert War Office Grid Reference to War Office Grid co-ordinates
  var cooresult = conv_GR2EN('vP', '397', '552', WOCG_GS);
    // then convert War Office Grid co-ordinates to OSGB36 geodetic co-ordinates
  var georesult = CS2Geo(cooresult.eastings, cooresult.northings, WOCG);
    // then convert OSGB36 geodetic co-ordinates to WGS84/ETRS89 geodetic co-ordinates
  var georesult2 = OSGB362WGS84(georesult.latitude, georesult.longitude);
    // display result
  alert ('Lat. ' + rad2deg(georesult2.latitude) + '; Lon. ' + rad2deg(georesult2.longitude));
```

Code Example 6:

Converts OS National Yard Grid co-ordinates [1,250,000; 1,425,000] to OSGB Grid Reference.

```
    // first convert OS National Yard Grid co-ordinates to OSGB National Grid co-ordinates
  var cooresult = YG2NG(1250000, 1425000);
    // then convert OSGB National Grid co-ordinates to OSGB National Grid Reference
  var refresult = conv_EN2GR(cooresult.eastings, cooresult.northings, OSNG_GS);
    // display result
  alert ('Grid ref. ' + refresult.square + ' ' + refresult.eastings + ' ' + refresult.northings);
```

Code Example 7:

Converts UTM Zone 30 co-ordinates [565500,5449250], based on ED50 datum, to WGS84/ETRS89 (GPS-compatible) geodetic co-ordinates.

```
    // first convert UTM Zone 30 (ED50) co-ordinates to ED50 geodetic co-ordinates
  var georesult = TM2Geo(565500, 5449250, UTM30_ED);
    // then convert ED50 geodetic co-ordinates to WGS84/ETRS89 geodetic co-ordinates
  var georesult2 = Geo2Geo(georesult.latitude, georesult.longitude, ED50, WGS84);
    // display result
  alert ('Lat. ' + rad2deg(georesult2.latitude) + '; Lon. ' + rad2deg(georesult2.longitude));
```

Code Example 8:

Converts OS National Grid Reference [SP 02294 02081] to OS County Series raw co-ordinates (Gloucestershire).

```
    // first convert OSGB National Grid Reference to OSGB National Grid co-ordinates
var cooresult = conv_GR2EN('SP', '02294', '02081', OSNG_GS);
    // then convert OSGB National Grid co-ordinates to OSGB36 geodetic co-ordinates
var georesult = TM2Geo(cooresult.eastings, cooresult.northings, OSNG);
    // then convert OSGB36 geodetic co-ordinates to County Series raw co-ordinates (county idx #21)
var cooresult2 = Geo2CtyRaw(georesult.latitude, georesult.longitude, 21);
    // retrieve name of projection origin (county index #21 - Gloucestershire)
var originname = MeridList[CtyList[21].Meridian].Name;
    // set display strings relevant to negative or positive easting/northing values
if (cooresult2.eastings < 0) {var eorw = ' ft W'; } else {var eorw = ' ft E'; }
if (cooresult2.northings < 0) {var nors = ' ft S'; } else {var nors = ' ft N'; }
    // set display easting/northing to their absolute values
var east = Math.abs(cooresult2.eastings); var north = Math.abs(cooresult2.northings);
    // display result
alert (east + eorw + ', ' + north + nors + ' of ' + originname + ' origin');
```

Code Example 9:

Converts County Series co-ordinates [24,540 ft. West; 141,620 ft. North of St Paul's Cathedral origin] to OSGB National Grid reference.

```
    // first determine a suitable county index number, based on St Paul's Cathedral (origin idx #44)
for (i=0; i<CtyList.length; i++) { if (CtyList[i].Meridian == 44) { var ctyidx = i; break; } }
    // then convert County Series raw co-ordinates to OSGB36 geodetic co-ordinates
var georesult = CtyRaw2Geo(-24540, 141620, ctyidx);
    // then convert OSGB36 geodetic co-ordinates to OSGB National Grid co-ordinates
var cooresult = Geo2TM(georesult.latitude, georesult.longitude, OSNG);
    // then convert OSGB National Grid co-ordinates to OSGB Grid Reference
var refresult = conv_EN2GR(cooresult.eastings, cooresult.northings, OSNG_GS);
    // display result
alert ('Grid ref. ' + refresult.square + ' ' + refresult.eastings + ' ' + refresult.northings);
```

Code Example 10:

Converts War Office Grid Co-ordinates [421180; 341030] to a list of OS County Series base sheets on which that point occurs and the internal co-ordinates of the point on each sheet.

```
    // first convert War Office Grid co-ordinates to OSGB36 geodetic co-ordinates
var georesult = CS2Geo(421180, 341030, WOCG);
    // then convert OSGB36 geodetic co-ordinates to array of County Series reference results
var countyresults = Geo2Cty(georesult.latitude, georesult.longitude);
    // initialise display string
var resultstring = '';
    // loop through all the County Series reference results
for (i=0; i<countyresults.length; i++) {
    // retrieve county name
  var countyname = CtyList[countyresults[i].cty].Name;
    // retrieve sheet number
  var sheetnumber = CtyList[countyresults[i].cty].Sheets[countyresults[i].sht].ShtNum;
    // add county name and sheet number to display string
  resultstring += countyname + ': Sheet ' + sheetnumber + ': ';
    // calculate internal co-ordinates of point from raw easting/northing and sheet co-ordinates
  var ei = countyresults[i].rwe – CtyList[countyresults[i].cty].Sheets[countyresults[i].sht].ShtW;
  var ni = countyresults[i].rwn – CtyList[countyresults[i].cty].Sheets[countyresults[i].sht].ShtS;
    // add sheet easting and northing to display string
  resultstring += RoundDec(ei,2) + ' ft from W edge, ' + RoundDec(ni,2) + ' ft from S edge\n'; }
  // display result
alert (resultstring);
```

Code Example 11:

Converts Bonne (Scotland) projection co-ordinates [23,520 ft. East; 216,100 ft. South] to a list of OS County Series base sheets on which that point occurs.

```
    // first convert Bonne (Scotland) co-ordinates to OSGB geodetic co-ordinates
var georesult = BN2Geo(23520, -216100, BONS);
    // then convert OSGB geodetic co-ordinates to array of County Series sheet numbers
var countyresults = Geo2CtyString(georesult.latitude, georesult.longitude);
    // initialise display string
var resultstring = '';
    // loop though all County Series sheet results, adding them to the display string
for (i=0; i<countyresults.length; i++) { resultstring += countyresults[i] + '\n'; }
    // display result
alert (resultstring);
```

Code Example 12:

Converts OS Irish Grid co-ordinates [325750; 386150] to WGS84/ETRS89 (GPS-compatible) geodetic co-ordinates.

```
    // first convert OS Irish Grid co-ordinates to OS Ireland 65 geodetic co-ordinates
var georesult = TM2Geo(325750, 386150, OSIG);
  // then convert OS Ireland 65 geodetic co-ordinates to WGS84/ETRS89 geodetic co-ordinates
var georesult2 = OSI652WGS84(georesult.latitude, georesult.longitude);
  // display result
alert ('Lat. ' + rad2deg(georesult2.latitude) + '; Lon. ' + rad2deg(georesult2.longitude));
```

Code Example 13:

Converts WGS84/ETRS89 (GPS-compatible) geodetic co-ordinates [53.5°N; 7.5°W] to OS Irish Grid Reference.

```
    // first convert WGS84/ETRS89 geodetic co-ordinates to OS Ireland 65 geodetic co-ordinates
var georesult = WGS842OSI65(deg2rad(53.5), deg2rad(-7.5));
  // then convert OS Ireland 65 geodetic co-ordinates to OS Irish Grid co-ordinates
var cooresult = Geo2TM(georesult.latitude, georesult.longitude, OSIG);
  // then convert OS Irish Grid co-ordinates to OS Irish Grid Reference
var refresult = conv_EN2GR(cooresult.eastings, cooresult.northings, OSIG_GS);
  // display result
alert ('Grid ref. ' + refresult.square + ' ' + refresult.eastings + ' ' + refresult.northings);
```

Code Example 14:

Converts OS Irish Grid Reference [N 428 136] to Bonne (Ireland) projection co-ordinates.

```
    // first convert OS Irish Grid Reference to OS Irish Grid co-ordinates
var cooresult = conv_GR2EN('N', '428', '136', OSIG_GS);
  // then convert OS Irish Grid co-ordinates to OS Ireland 65 geodetic co-ordinates
var georesult = TM2Geo(cooresult.eastings, cooresult.northings, OSIG);
  // then convert OS Ireland 65 geodetic co-ordinates to Bonne (Ireland) co-ordinates
var cooresult2 = Geo2BN(georesult.latitude, georesult.longitude, BONI);
  // set display strings relevant to negative or positive easting/northing values
if (cooresult2.eastings < 0) {var eorw = ' ft W'; } else {var eorw = ' ft E';}
if (cooresult2.northings < 0) {var nors = ' ft S'; } else {var nors = ' ft N';}
  // display result
alert (Math.abs(cooresult2.eastings) + eorw + '; ' + Math.abs(cooresult2.northings) + nors);
```

Code Example 15:

Converts War Office Irish Grid co-ordinates [309950; 274850] to Irish Transverse Mercator co-ordinates.

```
    // first convert War Office Irish Grid co-ordinates to OS Ireland 65 geodetic co-ordinates
var georesult = CS2Geo(309950, 274850, WOIG);
  // then convert OS Ireland 65 geodetic co-ordinates to WGS84/ETRS89 geodetic co-ordinates
var georesult2 = OSI652WGS84(georesult.latitude, georesult.longitude);
  // then convert WGS84/ETRS89 geodetic co-ordinates to Irish Transverse Mercator co-ordinates
var cooresult = Geo2TM(georesult2.latitude, georesult2.longitude, ITMG);
  // display result
alert ('Eastings = ' + cooresult.eastings + '; Northings = ' + cooresult.northings);
```

Code Example 16:

Converts Bonne (Ireland) co-ordinates [358,660 ft. West; 98,850 ft. North] to UTM Zone 29 co-ordinates based on ED50 datum.

```
    // first convert Bonne (Ireland) co-ordinates to OS Ireland 65 geodetic co-ordinates
var georesult = BN2Geo(-358660, 98850, BONI);
  // then convert OS Ireland 65 geodetic co-ordinates to WGS84/ETRS89 geodetic co-ordinates
var georesult2 = OSI652WGS84(georesult.latitude, georesult.longitude);
  // then convert WGS84/ETRS89 geodetic co-ordinates to ED50 geodetic co-ordinates
var georesult3 = Geo2Geo(georesult2.latitude, georesult2.longitude, WGS84, ED50);
  // then convert ED50 geodetic co-ordinates to UTM Zone 29 (ED50) co-ordinates
var cooresult = Geo2TM(georesult3.latitude, georesult3.longitude, UTM29_ED);
  // display result
alert ('Eastings = ' + cooresult.eastings + '; Northings = ' + cooresult.northings);
```

Reserved names

Names of variables, constants and functions utilised by the Co-ordinate Converter API

The following are complete lists, in alphabetical order, of the names of functions, variables and constants defined by the Co-ordinate Converter API. To ensure correct operation, you should avoid using these names for any other purpose in your code. Please note that names in JavaScript are case-sensitive.

Functions

arcsin
arsech
arsinh
artanh
BN2Geo
BN2Geosimple
BNgriddata
calc_M
conv_EN2GR
conv_GR2EN
coord
coordx
cosh
CountyData
CountySheet
CS2Geo
CS2Geosimple
CSgriddata
CtyRaw2Geo
ctyshtres
datum
deg2rad
dmsdata
ellipsoid
fivefig
Geo2BN
Geo2BNsimple
Geo2CS
Geo2CSsimple
Geo2Cty
Geo2CtyRaw
Geo2CtyString
Geo2Geo
Geo2TM
geodesic
geoextra
get_osgbshift
get_osishift
gridref
initctylist
initostn02
isvalidcoord
isvalidgeo
isvalidgeodms
isvalidgridref
isvalidigeo
isvalidigeodms
MeridianData
NG2YG
OSGB362WGS84
OSI652WGS84
rad2deg
rad2sec
RoundDec
Sec
sec2dms
sec2rad
sech
Sheet_WH
sinh
smallnum2string
stringmask
tanh
TM2Geo
TMgriddata
WGS842OSGB36
WGS842OSI65
YG2NG

Variables/constants

Airy
AiryMod
alphabetics
alphanumerics
BONI
BONS
CDEL
CtyList
ED50
FO1
GRS80
Int24
ITMG
MeridList
numerics
numerics_a
numerics_d
OSGB36
OSI65
osiA
osiB
OSIG
OSIG_GS
osik0
osilatm
osilonm
OSNG
OSNG_GS
OSNG_NS
OSNGgps
ostn02e
ostn02n
ostnres
OSYG
pi
s25in
s6in
UTM29_ED
UTM29_WGS
UTM30_ED
UTM30_WGS
UTM31_ED
UTM31_WGS
WGS84
WOCG
WOCG_GS
WOIG
WOIG_GS