Convert between Latitude/Longitude & UTM coordinates / MGRS grid references

The *Universal Transverse Mercator* coordinate system is a global system of grid-based mapping references.

UTM is in fact a set of 60 separate 'transverse Mercator' projections,[1] covering 60 longitudinal zones each 6° wide. It covers latitudes spanning 80°S to 84°N (the poles are covered by the separate 'UPS' system). Latitude/longitude points are converted to eastings and northings measured in metres along (an ellipsoidal model of) the earth's surface.

Functional demo

Enter (WGS84) latitude/longitude, UTM coordinates, or MGRS/NATO grid reference into the test boxes to try out the calculations (values are updated automatically on entry):

Lat/Long	13° 24′ 45.0000″ N, 103° 52′ 00.0000″ E
UTM coordinate	48 N 377299 1483035
MGRS grid reference	48P UV 77298 83034
Grid convergence	-000° 15′ 46.5164″
Point scale factor	0.99978624

Lat/long as: [✓ deg/min/sec] decimal degrees;

UTM to: [✓ m] mm;

MGRS 10 digits.

UTM

A ***Universal Transverse Mercator* coordinate** comprises a zone number, a hemisphere (N/S), an easting and a northing. Eastings are referenced from the central meridian of each zone, & northings from the equator, both in metres. To avoid negative numbers, 'false eastings' and 'false northings' are used:

- Eastings are measured from 500,000 metres west of the central meridian. Eastings (at the equator) range from 166,021m to 833,978m (the range decreases moving away from the equator); a point *on* the the central meridian has the value 500,000m.
- In the northern hemisphere, northings are measured from the equator – ranging from 0 at the equator to 9,329,005m at 84°N). In the southern hemisphere they are measured from 10,000,000 metres south of the equator (close to the pole) – ranging from 1,116,915m at 80°S to 10,000,000m at the equator.

Geokov has a good explanation.

Norway/Svalbard: the designers of UTM made two exceptions to the rule. The part of zone 31 covering western Norway is transferred to zone 32, and the zones covering Svalbard are tweaked to keep Svalbard in two zones (it's easier to understand looking at a map). These widened zones are viable partly because zones are much narrower so far north, so little precision is lost in merging them.

Note that UTM coordinates get rounded, not truncated (unlike MGRS grid references).

MGRS

The ***Military Grid Reference System*** is an alternative way of representing UTM coordinates.

Instead of having monotonic eastings for a zone, and northings measured to/from the equator, zones are divided into latitude bands, then into 100km grid squares identified by letter-pairs, then eastings and northings within each 100km grid square.

- Each UTM zone is divided into 20 latitude bands, each 8° tall (except 'X' is 12° tall), lettered from 'C' at 80°S to 'X' at 84°N (omitting 'I' and 'O'), so that the zone and the latitude band together make a 'grid zone designator' (GZD) (see map)

- Each ‘GZD’ is divided into 100km squares, identified by a letter-pair (see illustration).
- Eastings and northings are then given in metres within each 100km square.

Hence the UTM coordinate *31 N 303760 5787415* is equivalent to an MGRS grid reference of *31U CT 03760 87415*.

Depending on the map scale or scope of interest the GZD, and even the 100km square identification, may be dropped. Similarly, depending on the accuracy required, the easting and northing may be given to 10 digits (specifying metres), to 8 digits, to 6 digits, or to just 4 digits (specifying kilometre squares).

Since UTM coordinates have to indicate which hemisphere they are in, it is important not to confuse the hemisphere indicator with a UTM ‘latitude band’ (since latitude bands also include ‘N’ and ‘S’). In these scripts, UTM coordinates have a space between the zone and the hemisphere indicator, and no 100km square indicator.

Note that MGRS grid references get truncated, not rounded (unlike UTM coordinates).

Accuracy

These scripts to calculate UTM eastings/northings from geodetic latitude/longitude and vice-versa implement Karney’s method, which (in the *order* n^6 version used) gives results “accurate to 5 nm for distances up to 3,900 km from the central meridian” (improving on more familiar earlier methods from Snyder/Thomas/USDMA).

Such accuracy is laudable, but does open further issues. The now-ubiquitous geocentric global datum WGS-84 (World Geodetic System 1984) has no ‘physical realisation’ – it is not tied to geodetic groundstations, just to satellites – and is defined to be accurate to no better than around ±1 metre (good enough for most of us!).

A central problem is that at an accuracy of better than ±1 metre, plate tectonic movements become significant. Simplifying somewhat (well, actually, a lot!), the ITRS was developed, with ‘epoch’-dependant ITRFs, where the latitude/longitude coordinate of a position will vary over time.

Various ‘static’ reference frames are also defined for various continents – NAD-83 for North America, ETRS89 for Europe, GDA94 for Australia, etc – within which latitude/longitude coordinates remain fixed (at least to centimetre or so accuracy, major earthquake events excepted). These reference frames have ‘epoch’-dependant mappings to ITRF datums. (Due to plate tectonics, ETRS89 shifts against ITRF by about 25mm/year; GDA94 by around 80mm/year).

So if you are using the calculations given here to convert between geodetic latitude/longitude coordinates and UTM grid references, you can assure your users they have no accuracy concerns – but you may have a major task explaining datums and reference frames to them.

Remarkably, this accuracy comes in a very simple & concise implementation; having entirely failed even to begin to understand the mathematics, I find it a source of wonder that such an involved derivation can result in such a simple implementation – just a few dozen lines of code (though for those with better maths skills than mine, it seems well explained in Karney’s paper).

Performance

Using Chrome on a middling Core i5 PC, a latitude-longitude / UTM conversion takes around 0.01 – 0.02 milliseconds (hence around 50,000 – 100,000 per second).

Example usage of UTM / MGRS libraries

To convert a UTM coordinate to a (WGS-84) latitude/longitude point:

```
<script type="module">
    import Utm from 'https://cdn.jsdelivr.net/npm/geodesy@2/utm.js';
    const utmCoord = Utm.parse('31 N 303189 5787193');
    const latLongP = utmCoord.toLatLon();
    console.log(latLongP.toString('d', 2)); // '52.20°N, 000.12°E'
</script>
```

To convert a (WGS-84) latitude/longitude point to a UTM coordinate:

```
<script type="module">
    import { LatLon } from 'https://cdn.jsdelivr.net/npm/geodesy@2/utm.js';
    const latLongP = new LatLon(52.2, 0.12);
    const utmCoord = latLongP.toUtm();
    console.log(utmCoord.toString()); // '31 N 303189 5787193'
</script>
```

To convert an MGRS grid reference to a (WGS-84) latitude/longitude point:

```
<script type="module">
    import Mgrs from 'https://cdn.jsdelivr.net/npm/geodesy@2/mgrs.js';
    const mgrsGrid = Mgrs.parse('31U CT 03189 87193');
    const utmCoord = mgrsGrid.toUtm();
    const latLongP = utmCoord.toLatLon();
    console.log(latLongP.toString('d', 2)); // '52.20°N, 000.12°E'
    </script>
```

To convert a (WGS-84) latitude/longitude point to an MGRS grid reference:

```
    <script type="module">
        import { LatLon } from 'https://cdn.jsdelivr.net/npm/geodesy@2/mgrs.js';
        const latLongP = new LatLon(52.2, 0.12);
        const utmCoord = latLongP.toUtm();
        const mgrsGRef = utmCoord.toMgrs();
        console.log(mgrsGRef.toString()); // '31U CT 03189 87193'
    </script>
```

See UTM and MGRS documentation for full details.

See below for the JavaScript source code, also available on GitHub. Full documentation is available, as well as a test suite

Note I use Greek letters in variables representing maths symbols conventionally presented as Greek letters (also primes ′ U+02B9 & ″ U+02BA): I value the great benefit in legibility over the minor inconvenience in typing (if you encounter any problems, ensure your `<head>` includes `<meta charset="utf-8">`, and/or use UTF-8 encoding when saving files).

With its untyped C-style syntax, JavaScript reads remarkably close to pseudo-code: exposing the algorithms with a minimum of syntactic distractions. These functions should be simple to translate into other languages if required, though can also be used as-is in browsers and Node.js.

For convenience & clarity, I have extended the base JavaScript `Number` object with `toRadians()` and `toDegrees()` methods: I don’t see great likelihood of conflicts, as these are ubiquitous operations.

If you would like to show your appreciation and support continued development of these scripts, I would most gratefully accept donations.

If you need any advice or development work done, I am available for consultancy.

If you have any queries or find any problems, contact me at scripts-geo@movable-type.co.uk.

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  */
-84 Conversion Functions                          (c) Chris Veness 2014-2019  */
                                                                 MIT Licence  */
le-type.co.uk/scripts/latlong-utm-mgrs.html                                   */
le-type.co.uk/scripts/geodesy-library.html#utm                                */
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  */

sable indent */

nEllipsoidal, { Dms } from './latlon-ellipsoidal-datum.js';



rsal Transverse Mercator (UTM) system is a 2-dimensional Cartesian coordinate system
 locations on the surface of the Earth.

set of 60 transverse Mercator projections, normally based on the WGS-84 ellipsoid.
ch zone, coordinates are represented as eastings and northings, measures in metres; e.g.
251 5411932’.

od based on Karney 2011 ‘Transverse Mercator with an accuracy of a few nanometers’,
on Krüger 1912 ‘Konforme Abbildung des Erdellipsoids in der Ebene’.

tm



- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */



inates, with functions to parse them and convert them to LatLon points.




es a Utm coordinate object comprising zone, hemisphere, easting, northing on a given
 (normally WGS84).

m  {number}        zone - UTM 6° longitudinal zone (1..60 covering 180°W..180°E).
m  {string}        hemisphere - N for northern hemisphere, S for southern hemisphere.
```

```
m  {number}        easting - Easting in metres from false easting (-500km from central meridian).
m  {number}        northing - Northing in metres from equator (N) or from false northing -10,000km (S).
m  {LatLon.datums} [datum=WGS84] - Datum UTM coordinate is based on.
m  {number}        [convergence=null] - Meridian convergence (bearing of grid north
                   clockwise from true north), in degrees.
m  {number}        [scale=null] - Grid scale factor.
ms {boolean=true}  verifyEN - Check easting/northing is within 'normal' values (may be
                   suppressed for extended coherent coordinates or alternative datums
                   e.g. ED50 (epsg.io/23029).
ws {TypeError} Invalid UTM coordinate.

ple
ort Utm from '/js/geodesy/utm.js';
st utmCoord = new Utm(31, 'N', 448251, 5411932);

tor(zone, hemisphere, easting, northing, datum=LatLonEllipsoidal.datums.WGS84, convergence=null, scale=null, verifyEN=true) {
!(1<=zone && zone<=60)) throw new RangeError(`invalid UTM zone ‘${zone}’`);
zone != parseInt(zone)) throw new RangeError(`invalid UTM zone ‘${zone}’`);
typeof hemisphere != 'string' || !hemisphere.match(/[NS]/i)) throw new RangeError(`invalid UTM hemisphere ‘${hemisphere}’`);
verifyEN) { // range-check E/N values
if (!(0<=easting && easting<=1000e3)) throw new RangeError(`invalid UTM easting ‘${easting}’`);
if (hemisphere.toUpperCase()=='N' && !(0<=northing && northing<9328094)) throw new RangeError(`invalid UTM northing ‘${northing}’`);
if (hemisphere.toUpperCase()=='S' && !(1118414<northing && northing<=10000e3)) throw new RangeError(`invalid UTM northing ‘${northing}’`);

!datum || datum.ellipsoid==undefined) throw new TypeError(`unrecognised datum ‘${datum}’`);

.zone = Number(zone);
.hemisphere = hemisphere.toUpperCase();
.easting = Number(easting);
.northing = Number(northing);
.datum = datum;
.convergence = convergence===null ? null : Number(convergence);
.scale = scale===null ? null : Number(scale);




rts UTM zone/easting/northing coordinate to latitude/longitude.

ments Karney’s method, using Krüger series to order n⁶, giving results accurate to 5nm
istances up to 3900km from the central meridian.

m   {Utm} utmCoord - UTM coordinate to be converted to latitude/longitude.
rns {LatLon} Latitude/longitude of supplied grid reference.

ple
st grid = new Utm(31, 'N', 448251.795, 5411932.678);
st latlong = grid.toLatLon(); // 48°51′29.52″N, 002°17′40.20″E

() {
t { zone: z, hemisphere: h } = this;

t falseEasting = 500e3, falseNorthing = 10000e3;

t { a, f } = this.datum.ellipsoid; // WGS-84: a = 6378137, f = 1/298.257223563;

t k0 = 0.9996; // UTM scale on the central meridian

t x = this.easting - falseEasting;                            // make x ± relative to central meridian
t y = h=='S' ? this.northing - falseNorthing : this.northing; // make y ± relative to equator

--- from Karney 2011 Eq 15-22, 36:

t e = Math.sqrt(f*(2-f)); // eccentricity
t n = f / (2 - f);        // 3rd flattening
t n2 = n*n, n3 = n*n2, n4 = n*n3, n5 = n*n4, n6 = n*n5;

t A = a/(1+n) * (1 + 1/4*n2 + 1/64*n4 + 1/256*n6); // 2πA is the circumference of a meridian

t η = x / (k0*A);
t ξ = y / (k0*A);

t β = [ null, // note β is one-based array (6th order Krüger expressions)
1/2*n - 2/3*n2 + 37/96*n3 -    1/360*n4 -   81/512*n5 +    96199/604800*n6,
       1/48*n2 +  1/15*n3 - 437/1440*n4 +   46/105*n5 - 1118711/3870720*n6,
                17/480*n3 -   37/840*n4 - 209/4480*n5 +      5569/90720*n6,
                         4397/161280*n4 -   11/504*n5 -  830251/7257600*n6,
                                     4583/161280*n5 -  108847/3991680*n6,
                                                   20648693/638668800*n6 ];

ξ′ = ξ;
(let j=1; j<=6; j++) ξ′ -= β[j] * Math.sin(2*j*ξ) * Math.cosh(2*j*η);

η′ = η;
(let j=1; j<=6; j++) η′ -= β[j] * Math.cos(2*j*ξ) * Math.sinh(2*j*η);

t sinhη′ = Math.sinh(η′);
t sinξ′ = Math.sin(ξ′), cosξ′ = Math.cos(ξ′);

t τ′ = sinξ′ / Math.sqrt(sinhη′*sinhη′ + cosξ′*cosξ′);
```

```
δτi = null;
τi = τ′;

const σi = Math.sinh(e*Math.atanh(e*τi/Math.sqrt(1+τi*τi)));
const τi′ = τi * Math.sqrt(1+σi*σi) - σi * Math.sqrt(1+τi*τi);
δτi = (τ′ - τi′)/Math.sqrt(1+τi′*τi′)
    * (1 + (1-e*e)*τi*τi) / ((1-e*e)*Math.sqrt(1+τi*τi));
τi += δτi;
ile (Math.abs(δτi) > 1e-12); // using IEEE 754 δτi -> 0 after 2-3 iterations
ote relatively large convergence test as δτi toggles on ±1.12e-16 for eg 31 N 400000 5000000
t τ = τi;

t ϕ = Math.atan(τ);

λ = Math.atan2(sinhη′, cosξ′);

--- convergence: Karney 2011 Eq 26, 27

p = 1;
(let j=1; j<=6; j++) p -= 2*j*β[j] * Math.cos(2*j*ξ) * Math.cosh(2*j*η);
q = 0;
(let j=1; j<=6; j++) q += 2*j*β[j] * Math.sin(2*j*ξ) * Math.sinh(2*j*η);

t γ′ = Math.atan(Math.tan(ξ′) * Math.tanh(η′));
t γ″ = Math.atan2(q, p);

t γ = γ′ + γ″;

--- scale: Karney 2011 Eq 28

t sinϕ = Math.sin(ϕ);
t k′ = Math.sqrt(1 - e*e*sinϕ*sinϕ) * Math.sqrt(1 + τ*τ) * Math.sqrt(sinhη′*sinhη′ + cosξ′*cosξ′);
t k″ = A / a / Math.sqrt(p*p + q*q);

t k = k0 * k′ * k″;

-----------

t λ0 = ((z-1)*6 - 180 + 3).toRadians(); // longitude of central meridian
 λ0; // move λ from zonal to global coordinates

ound to reasonable precision
t lat = Number(ϕ.toDegrees().toFixed(14)); // nm precision (1nm = 10^-14°)
t lon = Number(λ.toDegrees().toFixed(14)); // (strictly lat rounding should be ϕ⋅cosϕ!)
t convergence = Number(γ.toDegrees().toFixed(9));
t scale = Number(k.toFixed(12));

t latLong = new LatLon_Utm(lat, lon, 0, this.datum);
.. and add the convergence and scale into the LatLon object ... wonderful JavaScript!
ong.convergence = convergence;
ong.scale = scale;

rn latLong;




s string representation of UTM coordinate.

 coordinate comprises (space-separated)
ne
misphere
sting
rthing.

m   {string} utmCoord - UTM coordinate (WGS 84).
m   {Datum}  [datum=WGS84] - Datum coordinate is defined in (default WGS 84).
rns {Utm} Parsed UTM coordinate.
ws  {TypeError} Invalid UTM coordinate.

ple
st utmCoord = Utm.parse('31 N 448251 5411932');
utmCoord: {zone: 31, hemisphere: 'N', easting: 448251, northing: 5411932 }

arse(utmCoord, datum=LatLonEllipsoidal.datums.WGS84) {
atch separate elements (separated by whitespace)
oord = utmCoord.trim().match(/\S+/g);

utmCoord==null || utmCoord.length!=4) throw new Error(`invalid UTM coordinate ‘${utmCoord}’`);

t zone = utmCoord[0], hemisphere = utmCoord[1], easting = utmCoord[2], northing = utmCoord[3];

rn new this(zone, hemisphere, easting, northing, datum); // 'new this' as may return subclassed types




ns a string representation of a UTM coordinate.

stinguish from MGRS grid zone designators, a space is left between the zone and the
phere.
```

```
that UTM coordinates get rounded, not truncated (unlike MGRS grid references).

m   {number} [digits=0] - Number of digits to appear after the decimal point (3 ≡ mm).
rns {string} A string representation of the coordinate.

ple
st utm = new Utm('31', 'N', 448251, 5411932).toString(4);  // 31 N 448251.0000 5411932.0000

(digits=0) {

t z = this.zone.toString().padStart(2, '0');
t h = this.hemisphere;
t e = this.easting.toFixed(digits);
t n = this.northing.toFixed(digits);

rn `${z} ${h} ${e} ${n}`;





m - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */



atLon with method to convert LatLon points to UTM coordinates.

LatLon

_Utm extends LatLonEllipsoidal {


rts latitude/longitude to UTM coordinate.

ments Karney's method, using Krüger series to order n⁶, giving results accurate to 5nm
istances up to 3900km from the central meridian.

m   {number} [zoneOverride] - Use specified zone rather than zone within which point lies;
    note overriding the UTM zone has the potential to result in negative eastings, and
    perverse results within Norway/Svalbard exceptions.
rns {Utm} UTM coordinate.
ws  {TypeError} Latitude outside UTM limits.

ple
st latlong = new LatLon(48.8582, 2.2945);
st utmCoord = latlong.toUtm(); // 31 N 448252 5411933

neOverride=undefined) {
!(-80<=this.lat && this.lat<=84)) throw new RangeError(`latitude '${this.lat}' outside UTM limits`);

t falseEasting = 500e3, falseNorthing = 10000e3;

zone = zoneOverride || Math.floor((this.lon+180)/6) + 1; // longitudinal zone
λ0 = ((zone-1)*6 - 180 + 3).toRadians(); // longitude of central meridian

--- handle Norway/Svalbard exceptions
rid zones are 8° tall; 0°N is offset 10 into latitude bands array
t mgrsLatBands = 'CDEFGHJKLMNPQRSTUVWXX'; // X is repeated for 80-84°N
t latBand = mgrsLatBands.charAt(Math.floor(this.lat/8+10));
djust zone & central meridian for Norway
zone==31 && latBand=='V' && this.lon>= 3) { zone++; λ0 += (6).toRadians(); }
djust zone & central meridian for Svalbard
zone==32 && latBand=='X' && this.lon<  9) { zone--; λ0 -= (6).toRadians(); }
zone==32 && latBand=='X' && this.lon>= 9) { zone++; λ0 += (6).toRadians(); }
zone==34 && latBand=='X' && this.lon< 21) { zone--; λ0 -= (6).toRadians(); }
zone==34 && latBand=='X' && this.lon>=21) { zone++; λ0 += (6).toRadians(); }
zone==36 && latBand=='X' && this.lon< 33) { zone--; λ0 -= (6).toRadians(); }
zone==36 && latBand=='X' && this.lon>=33) { zone++; λ0 += (6).toRadians(); }

t φ = this.lat.toRadians();      // latitude ± from equator
t λ = this.lon.toRadians() - λ0; // longitude ± from central meridian

llow alternative ellipsoid to be specified
t ellipsoid = this.datum ? this.datum.ellipsoid : LatLonEllipsoidal.ellipsoids.WGS84;
t { a, f } = ellipsoid; // WGS-84: a = 6378137, f = 1/298.257223563;

t k0 = 0.9996; // UTM scale on the central meridian

--- easting, northing: Karney 2011 Eq 7-14, 29, 35:

t e = Math.sqrt(f*(2-f)); // eccentricity
t n = f / (2 - f);        // 3rd flattening
t n2 = n*n, n3 = n*n2, n4 = n*n3, n5 = n*n4, n6 = n*n5;

t cosλ = Math.cos(λ), sinλ = Math.sin(λ), tanλ = Math.tan(λ);

t τ = Math.tan(φ); // τ ≡ tanφ, τ' ≡ tanφ'; prime (') indicates angles on the conformal sphere
t σ = Math.sinh(e*Math.atanh(e*τ/Math.sqrt(1+τ*τ)));

t τ' = τ*Math.sqrt(1+σ*σ) - σ*Math.sqrt(1+τ*τ);
```

```
t ξ′ = Math.atan2(τ′, cosλ);
t η′ = Math.asinh(sinλ / Math.sqrt(τ′*τ′ + cosλ*cosλ));

t A = a/(1+n) * (1 + 1/4*n2 + 1/64*n4 + 1/256*n6); // 2πA is the circumference of a meridian

t α = [ null, // note α is one-based array (6th order Krüger expressions)
1/2*n - 2/3*n2 + 5/16*n3 +   41/180*n4 -     127/288*n5 +      7891/37800*n6,
      13/48*n2 -  3/5*n3 + 557/1440*n4 +     281/630*n5 - 1983433/1935360*n6,
               61/240*n3 -  103/140*n4 + 15061/26880*n5 +   167603/181440*n6,
                       49561/161280*n4 -     179/168*n5 + 6601661/7257600*n6,
                                         34729/80640*n5 - 3418889/1995840*n6,
                                                      212378941/319334400*n6 ];

ξ = ξ′;
(let j=1; j<=6; j++) ξ += α[j] * Math.sin(2*j*ξ′) * Math.cosh(2*j*η′);

η = η′;
(let j=1; j<=6; j++) η += α[j] * Math.cos(2*j*ξ′) * Math.sinh(2*j*η′);

x = k0 * A * η;
y = k0 * A * ξ;

--- convergence: Karney 2011 Eq 23, 24

p′ = 1;
(let j=1; j<=6; j++) p′ += 2*j*α[j] * Math.cos(2*j*ξ′) * Math.cosh(2*j*η′);
q′ = 0;
(let j=1; j<=6; j++) q′ += 2*j*α[j] * Math.sin(2*j*ξ′) * Math.sinh(2*j*η′);

t γ′ = Math.atan(τ′ / Math.sqrt(1+τ′*τ′)*tanλ);
t γ″ = Math.atan2(q′, p′);

t γ = γ′ + γ″;

--- scale: Karney 2011 Eq 25

t sinφ = Math.sin(φ);
t k′ = Math.sqrt(1 - e*e*sinφ*sinφ) * Math.sqrt(1 + τ*τ) / Math.sqrt(τ′*τ′ + cosλ*cosλ);
t k″ = A / a * Math.sqrt(p′*p′ + q′*q′);

t k = k0 * k′ * k″;

-----------

hift x/y to false origins
x + falseEasting;             // make x relative to false easting
y < 0) y = y + falseNorthing; // make y in southern hemisphere relative to false northing

ound to reasonable precision
Number(x.toFixed(9)); // nm precision
Number(y.toFixed(9)); // nm precision
t convergence = Number(γ.toDegrees().toFixed(9));
t scale = Number(k.toFixed(12));

t h = this.lat>=0 ? 'N' : 'S'; // hemisphere

rn new Utm(zone, h, x, y, this.datum, convergence, scale, !!zoneOverride);




 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  */

 as default, LatLon_Utm as LatLon, Dms };
```

```
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  */
M Conversion Functions                                    (c) Chris Veness 2014-2019  */
                                                                         MIT Licence  */
le-type.co.uk/scripts/latlong-utm-mgrs.html                                           */
le-type.co.uk/scripts/geodesy-library.html#mgrs                                       */
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  */

{ LatLon as LatLonEllipsoidal, Dms } from './utm.js';



Grid Reference System (MGRS/NATO) grid references provides geocoordinate references
the entire globe, based on UTM projections.

rences comprise a grid zone designator, a 100km square identification, and an easting
ing (in metres); e.g. ‘31U DQ 48251 11932’.

 on requirements, some parts of the reference may be omitted (implied), and
northings may be given to varying resolution.

dc.gov/standards/projects/FGDC-standards-projects/usng/fgdc_std_011_2001_usng.pdf

grs
```

```
bands C..X 8° each, covering 80°S to 84°N

ds = 'CDEFGHJKLMNPQRSTUVWXX'; // X is repeated for 80-84°N



d square column (‘e’) letters repeat every third zone

etters = [ 'ABCDEFGH', 'JKLMNPQR', 'STUVWXYZ' ];



d square row (‘n’) letters repeat every other zone

etters = [ 'ABCDEFGHJKLMNPQRSTUV', 'FGHJKLMNPQRSTUVABCDE' ];


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */



Grid Reference System (MGRS/NATO) grid references, with methods to parse references, and
t to UTM coordinates.




es an Mgrs grid reference object.

m  {number} zone - 6° longitudinal zone (1..60 covering 180°W..180°E).
m  {string} band - 8° latitudinal band (C..X covering 80°S..84°N).
m  {string} e100k - First letter (E) of 100km grid square.
m  {string} n100k - Second letter (N) of 100km grid square.
m  {number} easting - Easting in metres within 100km grid square.
m  {number} northing - Northing in metres within 100km grid square.
m  {LatLon.datums} [datum=WGS84] - Datum UTM coordinate is based on.
ws {RangeError}  Invalid MGRS grid reference.

ple
ort Mgrs from '/js/geodesy/mgrs.js';
st mgrsRef = new Mgrs(31, 'U', 'D', 'Q', 48251, 11932); // 31U DQ 48251 11932

tor(zone, band, e100k, n100k, easting, northing, datum=LatLonEllipsoidal.datums.WGS84) {
!(1<=zone && zone<=60)) throw new RangeError(`invalid MGRS zone ‘${zone}’`);
zone != parseInt(zone)) throw new RangeError(`invalid MGRS zone ‘${zone}’`);
t errors = []; // check & report all other possible errors rather than reporting one-by-one
band.length!=1 || latBands.indexOf(band) == -1) errors.push(`invalid MGRS band ‘${band}’`);
e100k.length!=1 || e100kLetters[(zone-1)%3].indexOf(e100k) == -1) errors.push(`invalid MGRS 100km grid square column ‘${e100k}’ for zone ${zone}`);
n100k.length!=1 || n100kLetters[0].indexOf(n100k) == -1) errors.push(`invalid MGRS 100km grid square row ‘${n100k}’`);
isNaN(Number(easting))) errors.push(`invalid MGRS easting ‘${easting}’`);
isNaN(Number(northing))) errors.push(`invalid MGRS northing ‘${northing}’`);
!datum || datum.ellipsoid==undefined) errors.push(`unrecognised datum ‘${datum}’`);
errors.length > 0) throw new RangeError(errors.join(', '));

.zone = Number(zone);
.band = band;
.e100k = e100k;
.n100k = n100k;
.easting = Number(easting);
.northing = Number(northing);
.datum = datum;





rts MGRS grid reference to UTM coordinate.

references refer to squares rather than points (with the size of the square indicated
e precision of the reference); this conversion will return the UTM coordinate of the SW
r of the grid reference square.

rns {Utm} UTM coordinate of SW corner of this MGRS grid reference.

ple
st mgrsRef = Mgrs.parse('31U DQ 48251 11932');
st utmCoord = mgrsRef.toUtm(); // 31 N 448251 5411932

{
t hemisphere = this.band>='N' ? 'N' : 'S';

et easting specified by e100k (note +1 because eastings start at 166e3 due to 500km false origin)
t col = e100kLetters[(this.zone-1)%3].indexOf(this.e100k) + 1;
t e100kNum = col * 100e3; // e100k in metres

et northing specified by n100k
t row = n100kLetters[(this.zone-1)%2].indexOf(this.n100k);
t n100kNum = row * 100e3; // n100k in metres
```

```
et latitude of (bottom of) band
t latBand = (latBands.indexOf(this.band)-10)*8;

et northing of bottom of band, extended to include entirety of bottom-most 100km square
t nBand = Math.floor(new LatLonEllipsoidal(latBand, 3).toUtm().northing/100e3)*100e3;

00km grid square row letters repeat every 2,000km north; add enough 2,000km blocks to
et into required band
n2M = 0; // northing of 2,000km block
e (n2M + n100kNum + this.northing < nBand) n2M += 2000e3;

rn new Utm_Mgrs(this.zone, hemisphere, e100kNum+this.easting, n2M+n100kNum+this.northing, this.datum);




s string representation of MGRS grid reference.

RS grid reference comprises (space-separated)
id zone designator (GZD)
0km grid square letter-pair
sting
rthing.

m   {string} mgrsGridRef - String representation of MGRS grid reference.
rns {Mgrs}   Mgrs grid reference object.
ws  {Error}  Invalid MGRS grid reference.

ple
st mgrsRef = Mgrs.parse('31U DQ 48251 11932');
st mgrsRef = Mgrs.parse('31UDQ4825111932');
 mgrsRef: { zone:31, band:'U', e100k:'D', n100k:'Q', easting:48251, northing:11932 }

arse(mgrsGridRef) {
!mgrsGridRef) throw new Error(`invalid MGRS grid reference ‘${mgrsGridRef}’`);

heck for military-style grid reference with no separators
!mgrsGridRef.trim().match(/\s/)) {
if (!Number(mgrsGridRef.slice(0, 2))) throw new Error(`invalid MGRS grid reference ‘${mgrsGridRef}’`);
let en = mgrsGridRef.trim().slice(5); // get easting/northing following zone/band/100ksq
en = en.slice(0, en.length/2)+' '+en.slice(-en.length/2); // separate easting/northing
mgrsGridRef = mgrsGridRef.slice(0, 3)+' '+mgrsGridRef.slice(3, 5)+' '+en; // insert spaces


atch separate elements (separated by whitespace)
t ref = mgrsGridRef.match(/\S+/g);

ref==null || ref.length!=4) throw new Error(`invalid MGRS grid reference ‘${mgrsGridRef}’`);

plit gzd into zone/band
t gzd = ref[0];
t zone = gzd.slice(0, 2);
t band = gzd.slice(2, 3);

plit 100km letter-pair into e/n
t en100k = ref[1];
t e100k = en100k.slice(0, 1);
t n100k = en100k.slice(1, 2);

e = ref[2], n = ref[3];

tandardise to 10-digit refs - ie metres) (but only if < 10-digit refs, to allow decimals)
e.length>=5 ?  e : (e+'00000').slice(0, 5);
n.length>=5 ?  n : (n+'00000').slice(0, 5);

rn new Mgrs(zone, band, e100k, n100k, e, n);




ns a string representation of an MGRS grid reference.

stinguish from civilian UTM coordinate representations, no space is included within the
band grid zone designator.

nents are separated by spaces: for a military-style unseparated string, use
s.toString().replace(/ /g, '');

that MGRS grid references get truncated, not rounded (unlike UTM coordinates); grid
ences indicate a bounding square, rather than a point, with the size of the square
ated by the precision - a precision of 10 indicates a 1-metre square, a precision of 4
ates a 1,000-metre square (hence 31U DQ 48 11 indicates a 1km square with SW corner at
448000 5411000, which would include the 1m square 31U DQ 48251 11932).

m   {number}     [digits=10] - Precision of returned grid reference (eg 4 = km, 10 = m).
rns {string}     This grid reference in standard format.
ws  {RangeError} Invalid precision.

ple
st mgrsStr = new Mgrs(31, 'U', 'D', 'Q', 48251, 11932).toString(); // 31U DQ 48251 11932
```

```
(digits=10) {
![ 2, 4, 6, 8, 10 ].includes(Number(digits))) throw new RangeError(`invalid precision ‘${digits}’`);

t { zone, band, e100k, n100k, easting, northing } = this;

runcate to required precision
t eRounded = Math.floor(easting/Math.pow(10, 5-digits/2));
t nRounded = Math.floor(northing/Math.pow(10, 5-digits/2));

nsure leading zeros
t zPadded = zone.toString().padStart(2, '0');
t ePadded = eRounded.toString().padStart(digits/2, '0');
t nPadded = nRounded.toString().padStart(digits/2, '0');

rn `${zPadded}${band} ${e100k}${n100k} ${ePadded} ${nPadded}`;




- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */



tm with method to convert UTM coordinate to MGRS reference.

Utm

rs extends Utm {


rts UTM coordinate to MGRS reference.

rns {Mgrs}
ws  {TypeError} Invalid UTM coordinate.

ple
st utmCoord = new Utm(31, 'N', 448251, 5411932);
st mgrsRef = utmCoord.toMgrs(); // 31U DQ 48251 11932

 {
GRS zone is same as UTM zone
t zone = this.zone;

onvert UTM to lat/long to get latitude to determine band
t latlong = this.toLatLon();
rid zones are 8° tall, 0°N is 10th band
t band = latBands.charAt(Math.floor(latlong.lat/8+10)); // latitude band

olumns in zone 1 are A-H, zone 2 J-R, zone 3 S-Z, then repeating every 3rd zone
t col = Math.floor(this.easting / 100e3);
note -1 because eastings start at 166e3 due to 500km false origin)
t e100k = e100kLetters[(zone-1)%3].charAt(col-1);

ows in even zones are A-V, in odd zones are F-E
t row = Math.floor(this.northing / 100e3) % 20;
t n100k = n100kLetters[(zone-1)%2].charAt(row);

runcate easting/northing to within 100km grid square
easting = this.easting % 100e3;
northing = this.northing % 100e3;

ound to nm precision
ing = Number(easting.toFixed(6));
hing = Number(northing.toFixed(6));

rn new Mgrs(zone, band, e100k, n100k, easting, northing);







atLonEllipsoidal adding toMgrs() method to the Utm object returned by LatLon.toUtm().

LatLonEllipsoidal

_Utm_Mgrs extends LatLonEllipsoidal {


rts latitude/longitude to UTM coordinate.

w of LatLon.toUtm, returning Utm augmented with toMgrs() method.

m   {number} [zoneOverride] - Use specified zone rather than zone within which point lies;
    note overriding the UTM zone has the potential to result in negative eastings, and
    perverse results within Norway/Svalbard exceptions (this is unlikely to be relevant
    for MGRS, but is needed as Mgrs passes through the Utm class).
rns {Utm}   UTM coordinate.
ws  {Error} If point not valid, if point outside latitude range.

ple
st latlong = new LatLon(48.8582, 2.2945);
st utmCoord = latlong.toUtm(); // 31 N 448252 5411933

neOverride=undefined) {
t utm = super.toUtm(zoneOverride);
rn new Utm_Mgrs(utm.zone, utm.hemisphere, utm.easting, utm.northing, utm.datum, utm.convergence, utm.scale);




 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -  */

s as default, Utm_Mgrs as Utm, Latlon_Utm_Mgrs as LatLon, Dms };
```