



# Movable Type Scripts

## 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,<sup>1</sup> covering 60 latitudinal 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	<input type="text"/>
UTM coordinate	<input type="text" value="48 N 377298.745 1483034.794"/>
MGRS grid reference	<input type="text"/>


Grid convergence      -000°15'46.5164"

Point scale factor      0.99978624

Lat/long as: ☐ deg/min/sec ☐ decimal degrees;

UTM to: ☐ m ☐ mm;

UTM  digits.



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<sup>6</sup>* 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).

See below for the JavaScript source code, also available on *GitHub utm/mgrs*. 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.

I have extended the base JavaScript `Number` object with `toRadians()` and `toDegrees()`; I don't see great likelihood of conflicts.

I offer these scripts for free use and adaptation to balance my debt to the open-source info-verse. You are welcome to re-use these scripts [under an *MIT* licence, without any warranty express or implied] provided solely that you retain my copyright notice and a link to this page.



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`.

© 2014–2016 Chris Veness

```
/* ----- */
/* UTM / WGS-84 Conversion Functions (c) Chris Veness 2014-2016 */
/* MIT Licence */
/* www.movable-type.co.uk/scripts/latlong-utm-mgrs.html */
/* www.movable-type.co.uk/scripts/geodesy/docs/module-utm.html */
/* ----- */
```

```
'use strict';
if (typeof module !== 'undefined' && module.exports) var LatLon = require('./latlon-ellipsoidal.js'); // ≡ import LatLon from 'latlon-ellipsoidal.js'

/**
 * Convert between Universal Transverse Mercator coordinates and WGS 84 latitude/longitude points.
 *
 * Method based on Karney 2011 'Transverse Mercator with an accuracy of a few nanometers',
 * building on Krüger 1912 'Konforme Abbildung des Erdellipsoids in der Ebene'.
 *
 * @module utm
 * @requires latlon-ellipsoidal
 */

/**
 * Creates a Utm coordinate object.
 *
 * @constructor
 * @param {number} zone - UTM 6° longitudinal zone (1..60 covering 180°W..180°E).
 * @param {string} hemisphere - N for northern hemisphere, S for southern hemisphere.
 * @param {number} easting - Easting in metres from false easting (-500km from central meridian).
 * @param {number} northing - Northing in metres from equator (N) or from false northing -10,000km (S).
 * @param {LatLon.datum} [datum=WGS84] - Datum UTM coordinate is based on.
 * @param {number} [convergence] - Meridian convergence (bearing of grid north clockwise from true
 * north), in degrees
 * @param {number} [scale] - Grid scale factor
 * @throws {Error} Invalid UTM coordinate
 *
 * @example
 * var utmCoord = new Utm(31, 'N', 448251, 5411932);
 */
function Utm(zone, hemisphere, easting, northing, datum, convergence, scale) {
  if (!(this instanceof Utm)) { // allow instantiation without 'new'
    return new Utm(zone, hemisphere, easting, northing, datum, convergence, scale);
  }

  if (datum === undefined) datum = LatLon.datum.WGS84; // default if not supplied
  if (convergence === undefined) convergence = null; // default if not supplied
  if (scale === undefined) scale = null; // default if not supplied

  if (!(1 <= zone && zone <= 60)) throw new Error('Invalid UTM zone '+zone);
  if (!hemisphere.match(/[NS]/i)) throw new Error('Invalid UTM hemisphere '+hemisphere);
  // range-check easting/northing (with 40km overlap between zones) - is this worthwhile?
  //if (!(120e3 <= easting && easting <= 880e3)) throw new Error('Invalid UTM easting '+ easting);
  //if (!(0 <= northing && northing <= 10000e3)) throw new Error('Invalid UTM northing '+ northing);

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

/**
 * Converts latitude/longitude to UTM coordinate.
 *
 * Implements Karney's method, using Krüger series to order n^6, giving results accurate to 5nm for
 * distances up to 3900km from the central meridian.
 *
 * @returns {Utm} UTM coordinate.
 * @throws {Error} If point not valid, if point outside latitude range.
 *
 * @example
 * var latlong = new LatLon(48.8582, 2.2945);
 * var utmCoord = latlong.toUtm(); // utmCoord.toString(): '31 N 448252 5411933'
 */
LatLon.prototype.toUtm = function() {
  if (isNaN(this.lat) || isNaN(this.lon)) throw new Error('Invalid point');
  if (!(-80 <= this.lat && this.lat <= 84)) throw new Error('Outside UTM limits');

  var falseEasting = 500e3, falseNorthing = 10000e3;

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

  // ---- handle Norway/Svalbard exceptions
  // grid zones are 8° tall; 0°N is offset 10 into latitude bands array
  var mgrsLatBands = 'CDEFGHJKLMNPQRSTUUVWXX'; // X is repeated for 80-84°N
  var latBand = mgrsLatBands.charAt(Math.floor(this.lat/8+10));
  // adjust zone & central meridian for Norway
  if (zone==31 && latBand=='V' && this.lon>= 3) { zone++; λ0 += (6).toRadians(); }
  // adjust zone & central meridian for Svalbard
  if (zone==32 && latBand=='X' && this.lon< 9) { zone--; λ0 -= (6).toRadians(); }
  if (zone==32 && latBand=='X' && this.lon>= 9) { zone++; λ0 += (6).toRadians(); }
  if (zone==34 && latBand=='X' && this.lon< 21) { zone--; λ0 -= (6).toRadians(); }
  if (zone==34 && latBand=='X' && this.lon>=21) { zone++; λ0 += (6).toRadians(); }
  if (zone==36 && latBand=='X' && this.lon< 33) { zone--; λ0 -= (6).toRadians(); }
  if (zone==36 && latBand=='X' && this.lon>=33) { zone++; λ0 += (6).toRadians(); }

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



```
var a = this.datum.ellipsoid.a, f = this.datum.ellipsoid.f;
// WGS 84: a = 6378137, b = 6356752.314245, f = 1/298.257223563;

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

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

var e = Math.sqrt(f*(2-f)); // eccentricity
var n = f / (2 - f); // 3rd flattening
var n2 = n*n, n3 = n*n2, n4 = n*n3, n5 = n*n4, n6 = n*n5; // TODO: compare Horner-form accuracy?

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

var τ = Math.tan(φ); // τ ≡ tanφ, τ' ≡ tanφ'; prime (') indicates angles on the conformal sphere
var σ = Math.sinh(e*Math.atanh(e*τ/Math.sqrt(1+τ*τ)));

var τ' = τ*Math.sqrt(1+σ*σ) - σ*Math.sqrt(1+τ*τ);

var ξ' = Math.atan2(τ', cosλ);
var η' = Math.asinh(sinλ / Math.sqrt(τ'*τ' + cosλ*cosλ));

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

var 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 ];

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

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

var x = k0 * A * η;
var y = k0 * A * ξ;

// ---- convergence: Karney 2011 Eq 23, 24

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

var γ' = Math.atan(τ' / Math.sqrt(1+τ'*τ'))*tanλ;
var γ'' = Math.atan2(q', p');

var γ = γ' + γ'';

// ---- scale: Karney 2011 Eq 25

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

var k = k0 * k' * k'';

// -----

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

// round to reasonable precision
x = Number(x.toFixed(6)); // nm precision
y = Number(y.toFixed(6)); // nm precision
var convergence = Number(γ.toDegrees().toFixed(9));
var scale = Number(k.toFixed(12));

var h = this.lat>=0 ? 'N' : 'S'; // hemisphere

return new Utm(zone, h, x, y, this.datum, convergence, scale);
};

/**
 * Converts UTM zone/easting/northing coordinate to latitude/longitude
 *
 * @param {Utm} utmCoord - UTM coordinate to be converted to latitude/longitude.
 * @returns {LatLon} Latitude/longitude of supplied grid reference.
 *
 * @example
 * var grid = new Utm(31, 'N', 448251.795, 5411932.678);
 * var latlong = grid.toLatLonE(); // latlong.toString(): 48°51'29.52"N, 002°17'40.20"E
 */
Utm.prototype.toLatLonE = function() {
  var z = this.zone;
  var h = this.hemisphere;
  var x = this.easting;
  var y = this.northing;

  if (isNaN(z) || isNaN(x) || isNaN(y)) throw new Error('Invalid coordinate');
```

```
var falseEasting = 500e3, falseNorthing = 10000e3;

var a = this.datum.ellipsoid.a, f = this.datum.ellipsoid.f;
// WGS 84: a = 6378137, b = 6356752.314245, f = 1/298.257223563;

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

x = x - falseEasting; // make x ± relative to central meridian
y = h=='S' ? y - falseNorthing : y; // make y ± relative to equator

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

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

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

var η = x / (k0*A);
var ξ = y / (k0*A);

var 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 ];

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

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

var sinhη' = Math.sinh(η');
var sinξ' = Math.sin(ξ'), cosξ' = Math.cos(ξ');

var τ' = sinξ' / Math.sqrt(sinhη'*sinhη' + cosξ'*cosξ');

var τi = τ';
do {
  var σi = Math.sinh(e*Math.atanh(e*τi/Math.sqrt(1+τi*τi)));
  var τi' = τi * Math.sqrt(1+σi*σi) - σi * Math.sqrt(1+τi*τi);
  var δτi = (τ' - τi')/Math.sqrt(1+τi'*τi')
    * (1 + (1-e*e)*τi*τi) / ((1-e*e)*Math.sqrt(1+τi*τi));
  τi += δτi;
} while (Math.abs(δτi) > 1e-12); // using IEEE 754 δτi -> 0 after 2-3 iterations
// note relatively large convergence test as δτi toggles on ±1.12e-16 for eg 31 N 400000 5000000
var τ = τi;

var φ = Math.atan(τ);

var λ = Math.atan2(sinhη', cosξ');

// ---- convergence: Karney 2011 Eq 26, 27

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

var γ' = Math.atan(Math.tan(ξ') * Math.tanh(η'));
var γ'' = Math.atan2(q, p);

var γ = γ' + γ'';

// ---- scale: Karney 2011 Eq 28

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

var k = k0 * k' * k'';

// -----

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

// round to reasonable precision
var lat = Number(φ.toDegrees().toFixed(11)); // nm precision (1nm = 10^-11°)
var lon = Number(λ.toDegrees().toFixed(11)); // (strictly lat rounding should be φ*cosφ!)
var convergence = Number(γ.toDegrees().toFixed(9));
var scale = Number(k.toFixed(12));

var latLong = new LatLon(lat, lon, this.datum);
// ... and add the convergence and scale into the LatLon object ... wonderful JavaScript!
latLong.convergence = convergence;
latLong.scale = scale;

return latLong;
};
```

```

/**
 * Parses string representation of UTM coordinate.
 *
 * A UTM coordinate comprises (space-separated)
 * - zone
 * - hemisphere
 * - easting
 * - northing.
 *
 * @param {string} utmCoord - UTM coordinate (WGS 84).
 * @param {Datum} [datum=WGS84] - Datum coordinate is defined in (default WGS 84).
 * @returns {Utm}
 * @throws {Error} Invalid UTM coordinate.
 *
 * @example
 * var utmCoord = Utm.parse('31 N 448251 5411932');
 * // utmCoord: {zone: 31, hemisphere: 'N', easting: 448251, northing: 5411932 }
 */
Utm.parse = function(utmCoord, datum) {
    if (datum === undefined) datum = LatLon.datum.WGS84; // default if not supplied

    // match separate elements (separated by whitespace)
    utmCoord = utmCoord.trim().match(/\S+/g);

    if (utmCoord==null || utmCoord.length!=4) throw new Error('Invalid UTM coordinate \''+utmCoord+'');

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

    return new Utm(zone, hemisphere, easting, northing, datum);
};

/**
 * Returns a string representation of a UTM coordinate.
 *
 * To distinguish from MGRS grid zone designators, a space is left between the zone and the
 * hemisphere.
 *
 * Note that UTM coordinates get rounded, not truncated (unlike MGRS grid references).
 *
 * @param {number} [digits=0] - Number of digits to appear after the decimal point (3 ≡ mm).
 * @returns {string} A string representation of the coordinate.
 *
 * @example
 * var utm = Utm.parse('31 N 448251 5411932').toString(4); // 31 N 448251.0000 5411932.0000
 */
Utm.prototype.toString = function(digits) {
    digits = Number(digits||0); // default 0 if not supplied

    var z = this.zone<10 ? '0'+this.zone : this.zone; // leading zero
    var h = this.hemisphere;
    var e = this.easting;
    var n = this.northing;
    if (isNaN(z) || !h.match(/[NS]/) || isNaN(e) || isNaN(n)) return '';

    return z+' '+h+' '+e.toFixed(digits)+' '+n.toFixed(digits);
};

/* ----- */

/** Polyfill Math.sinh for old browsers / IE */
if (Math.sinh === undefined) {
    Math.sinh = function(x) {
        return (Math.exp(x) - Math.exp(-x)) / 2;
    };
}

/** Polyfill Math.cosh for old browsers / IE */
if (Math.cosh === undefined) {
    Math.cosh = function(x) {
        return (Math.exp(x) + Math.exp(-x)) / 2;
    };
}

/** Polyfill Math.tanh for old browsers / IE */
if (Math.tanh === undefined) {
    Math.tanh = function(x) {
        return (Math.exp(x) - Math.exp(-x)) / (Math.exp(x) + Math.exp(-x));
    };
}

/** Polyfill Math.asinh for old browsers / IE */
if (Math.asinh === undefined) {
    Math.asinh = function(x) {
        return Math.log(x + Math.sqrt(1 + x*x));
    };
}

/** Polyfill Math.atanh for old browsers / IE */
if (Math.atanh === undefined) {
    Math.atanh = function(x) {
        return Math.log((1+x) / (1-x)) / 2;
    };
}

```

```
/** Polyfill String.trim for old browsers
 * (q.v. blog.stevenlevithan.com/archives/faster-trim-javascript) */
if (String.prototype.trim === undefined) {
    String.prototype.trim = function() {
        return String(this).replace(/^\s\s*/, '').replace(/\s\s*$/, '');
    };
}

/* ----- */
if (typeof module !== 'undefined' && module.exports) module.exports = Utm; // ≡ export default Utm
```

```
/* ----- */
/*  MGRS / UTM Conversion Functions (c) Chris Veness 2014-2016 */
/*                                     MIT Licence */
/* www.movable-type.co.uk/scripts/latlong-utm-mgrs.html */
/* www.movable-type.co.uk/scripts/geodesy/docs/module-mgrs.html */
/* ----- */

'use strict';
if (typeof module !== 'undefined' && module.exports) var Utm = require('./utm.js'); // ≡ import Utm from 'utm.js'
if (typeof module !== 'undefined' && module.exports) var LatLon = require('./latlon-ellipsoidal.js'); // ≡ import LatLon from 'latlon-ellipsoidal.js'
```

```
/**
 * Convert between Universal Transverse Mercator (UTM) coordinates and Military Grid Reference
 * System (MGRS/NATO) grid references.
 *
 * @module mgrs
 * @requires utm
 * @requires latlon-ellipsoidal
 */
```

```
/* qv www.fgdc.gov/standards/projects/FGDC-standards-projects/usng/fgdc_std_011_2001_usng.pdf p10 */
```

```
/*
 * Latitude bands C..X 8° each, covering 80°S to 84°N
 */
Mgrs.latBands = 'CDEFGHJKLMNPQRSTUVWXYZ'; // X is repeated for 80-84°N
```

```
/*
 * 100km grid square column ('e') letters repeat every third zone
 */
Mgrs.e100kLetters = [ 'ABCDEFGH', 'JKLMNPQR', 'STUVWXYZ' ];
```

```
/*
 * 100km grid square row ('n') letters repeat every other zone
 */
Mgrs.n100kLetters = [ 'ABCDEFGHJKLMNPQRSTUV', 'FGHJKLMNPQRSTUVABCDE' ];
```

```
/**
 * Creates an Mgrs grid reference object.
 *
 * @constructor
 * @param {number} zone - 6° longitudinal zone (1..60 covering 180°W..180°E).
 * @param {string} band - 8° latitudinal band (C..X covering 80°S..84°N).
 * @param {string} e100k - First letter (E) of 100km grid square.
 * @param {string} n100k - Second letter (N) of 100km grid square.
 * @param {number} easting - Easting in metres within 100km grid square.
 * @param {number} northing - Northing in metres within 100km grid square.
 * @param {LatLon.datum} [datum=WGS84] - Datum UTM coordinate is based on.
 * @throws {Error} Invalid MGRS grid reference.
 *
 * @example
 * var mgrsRef = new Mgrs(31, 'U', 'D', 'Q', 48251, 11932); // 31U DQ 48251 11932
 */
function Mgrs(zone, band, e100k, n100k, easting, northing, datum) {
    // allow instantiation without 'new'
    if (!(this instanceof Mgrs)) return new Mgrs(zone, band, e100k, n100k, easting, northing, datum);

    if (datum === undefined) datum = LatLon.datum.WGS84; // default if not supplied

    if (!(1<=zone && zone<=60)) throw new Error('Invalid MGRS grid reference (zone \''+zone+'\')');
    if (band.length !== 1) throw new Error('Invalid MGRS grid reference (band \''+band+'\')');
    if (Mgrs.latBands.indexOf(band) == -1) throw new Error('Invalid MGRS grid reference (band \''+band+'\')');
    if (e100k.length!=1) throw new Error('Invalid MGRS grid reference (e100k \''+e100k+'\')');
    if (n100k.length!=1) throw new Error('Invalid MGRS grid reference (n100k \''+n100k+'\')');

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

```
/**
 * Converts UTM coordinate to MGRS reference.
 *
```

```
* @returns {Mgrs}
* @throws {Error} Invalid UTM coordinate.
*
* @example
*   var utmCoord = new Utm(31, 'N', 448251, 5411932);
*   var mgrsRef = utmCoord.toMgrs(); // 31U DQ 48251 11932
*/
Utm.prototype.toMgrs = function() {
    if (isNaN(this.zone + this.easting + this.northing)) throw new Error('Invalid UTM coordinate ''+this.toString()+''');

    // MGRS zone is same as UTM zone
    var zone = this.zone;

    // convert UTM to lat/long to get latitude to determine band
    var latlong = this.toLatLonE();
    // grid zones are 8° tall, 0°N is 10th band
    var band = Mgrs.latBands.charAt(Math.floor(latlong.lat/8+10)); // latitude band

    // columns in zone 1 are A-H, zone 2 J-R, zone 3 S-Z, then repeating every 3rd zone
    var col = Math.floor(this.easting / 100e3);
    var e100k = Mgrs.e100kLetters[(zone-1)%3].charAt(col-1); // col-1 since 1*100e3 -> A (index 0), 2*100e3 -> B (index 1), etc.

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

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

    // round to nm precision
    easting = Number(easting.toFixed(6));
    northing = Number(northing.toFixed(6));

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

/**
 * Converts MGRS grid reference to UTM coordinate.
 *
 * @returns {Utm}
 *
 * @example
 *   var utmCoord = Mgrs.parse('31U DQ 448251 11932').toUtm(); // 31 N 448251 5411932
 */
Mgrs.prototype.toUtm = function() {
    var zone = this.zone;
    var band = this.band;
    var e100k = this.e100k;
    var n100k = this.n100k;
    var easting = this.easting;
    var northing = this.northing;

    var hemisphere = band>='N' ? 'N' : 'S';

    // get easting specified by e100k
    var col = Mgrs.e100kLetters[(zone-1)%3].indexOf(e100k) + 1; // index+1 since A (index 0) -> 1*100e3, B (index 1) -> 2*100e3, etc.
    var e100kNum = col * 100e3; // e100k in metres

    // get northing specified by n100k
    var row = Mgrs.n100kLetters[(zone-1)%2].indexOf(n100k);
    var n100kNum = row * 100e3; // n100k in metres

    // get latitude of (bottom of) band
    var latBand = (Mgrs.latBands.indexOf(band)-10)*8;

    // northing of bottom of band, extended to include entirety of bottommost 100km square
    // (100km square boundaries are aligned with 100km UTM northing intervals)
    var nBand = Math.floor(new LatLon(latBand, 0).toUtm().northing/100e3)*100e3;
    // 100km grid square row letters repeat every 2,000km north; add enough 2,000km blocks to get
    // into required band
    var n2M = 0; // northing of 2,000km block
    while (n2M + n100kNum + northing < nBand) n2M += 2000e3;

    return new Utm(zone, hemisphere, e100kNum+easting, n2M+n100kNum+northing, this.datum);
};

/**
 * Parses string representation of MGRS grid reference.
 *
 * An MGRS grid reference comprises (space-separated)
 * - grid zone designator (GZD)
 * - 100km grid square letter-pair
 * - easting
 * - northing.
 *
 * @param {string} mgrsGridRef - String representation of MGRS grid reference.
 * @returns {Mgrs} Mgrs grid reference object.
 * @throws {Error} Invalid MGRS grid reference.
 *
 * @example
 *   var mgrsRef = Mgrs.parse('31U DQ 48251 11932');
 *   var mgrsRef = Mgrs.parse('31UDQ4825111932');
 *   // mgrsRef: {zone:31, band:'U', e100k:'D', n100k:'Q', easting:48251, northing:11932 }
 */
```



```
Mgrs.parse = function(mgrsGridRef) {
  mgrsGridRef = mgrsGridRef.trim();

  // check for military-style grid reference with no separators
  if (!mgrsGridRef.match(/\s/)) {
    var en = mgrsGridRef.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
  }

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

  if (mgrsGridRef==null || mgrsGridRef.length!=4) throw new Error('Invalid MGRS grid reference \''+mgrsGridRef+'');

  // split gzd into zone/band
  var gzd = mgrsGridRef[0];
  var zone = gzd.slice(0, 2);
  var band = gzd.slice(2, 3);

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

  var e = mgrsGridRef[2], n = mgrsGridRef[3];

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

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

/**
 * Returns a string representation of an MGRS grid reference.
 *
 * To distinguish from civilian UTM coordinate representations, no space is included within the
 * zone/band grid zone designator.
 *
 * Components are separated by spaces: for a military-style unseparated string, use
 * Mgrs.toString().replace(/ /g, '');
 *
 * Note that MGRS grid references get truncated, not rounded (unlike UTM coordinates).
 *
 * @param {number} [digits=10] - Precision of returned grid reference (eg 4 = km, 10 = m).
 * @returns {string} This grid reference in standard format.
 * @throws {Error} Invalid precision.
 *
 * @example
 * var mgrsStr = new Mgrs(31, 'U', 'D', 'Q', 48251, 11932).toString(); // '31U DQ 48251 11932'
 */
Mgrs.prototype.toString = function(digits) {
  digits = (digits === undefined) ? 10 : Number(digits);
  if ([2,4,6,8,10].indexOf(digits) == -1) throw new Error('Invalid precision \''+digits+'');

  var zone = this.zone.pad(2); // ensure leading zero
  var band = this.band;

  var e100k = this.e100k;
  var n100k = this.n100k;

  // set required precision
  var easting = Math.floor(this.easting/Math.pow(10, 5-digits/2));
  var northing = Math.floor(this.northing/Math.pow(10, 5-digits/2));

  // ensure leading zeros
  easting = easting.pad(digits/2);
  northing = northing.pad(digits/2);

  return zone+band + ' ' + e100k+n100k + ' ' + easting + ' ' + northing;
};

/* ----- */

/** Extend Number object with method to pad with leading zeros to make it w chars wide
 * (q.v. stackoverflow.com/questions/2998784 */
if (Number.prototype.pad === undefined) {
  Number.prototype.pad = function(w) {
    var n = this.toString();
    while (n.length < w) n = '0' + n;
    return n;
  };
}

/* ----- */
if (typeof module != 'undefined' && module.exports) module.exports = Mgrs; // ≡ export default Mgrs
```