INTERNATIONAL POSTCODE SYSTEM (IPCS)

by

Bartosz Cieślicki

Rochdale 2018

System assumptions

The main concept of the system is to create a uniform system of postal codes for the whole earth. The most important aim is:

ENCODING GEOGRAPHICAL COORDINATES OF EACH POSTCODE IN ITS OWN RECORD.

The system so constructed will eliminate the need to have a large global database containing the location of each postcode, It will also reduce or completely eliminate the costs of geocoding and as well as greatly speed up computer systems which search for a geographical position for the requested address.

Assumptions:

1. The area which is identified by postcode should not exceed 1000m2. So that you can easily and accurately reach the requested place, using a map or satellite navigation.
2. The system must cover every part of the planet, regardless of population, density of development, shape or type of surface.
3. Postal codes must be calculated / created on the basis of an algorithm and not given**.**
4. You can calculate geographic coordinates from each postal code record that indicates center of the area.

International Postcode System (IPCS):

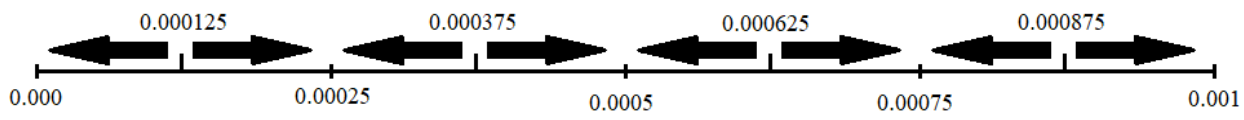
1. Should be based on the Latin alphabet as the most popular in the world.
2. It should have a relatively small number of characters that allows easy memorizing.
3. It should not contain special characters that are specific to a given country.
4. It should eliminate the most frequent mistakes in writing and reading (ie the letters O with the number 0 and the letter I with the number 1).

Construction of sectors

To calculate the International Postcode System(IPCS), we will use latitude and longitude in the decimal system. The longitude is between (-180, 180> while the latitude is between <-90, 90>. The first very important step to calculate the international postcode is to round the latitude and longitude to a quarter of 0.001.

Example:

Let's assume that the number t $\in$<0.000, 0.001>, the number x is our rounding.



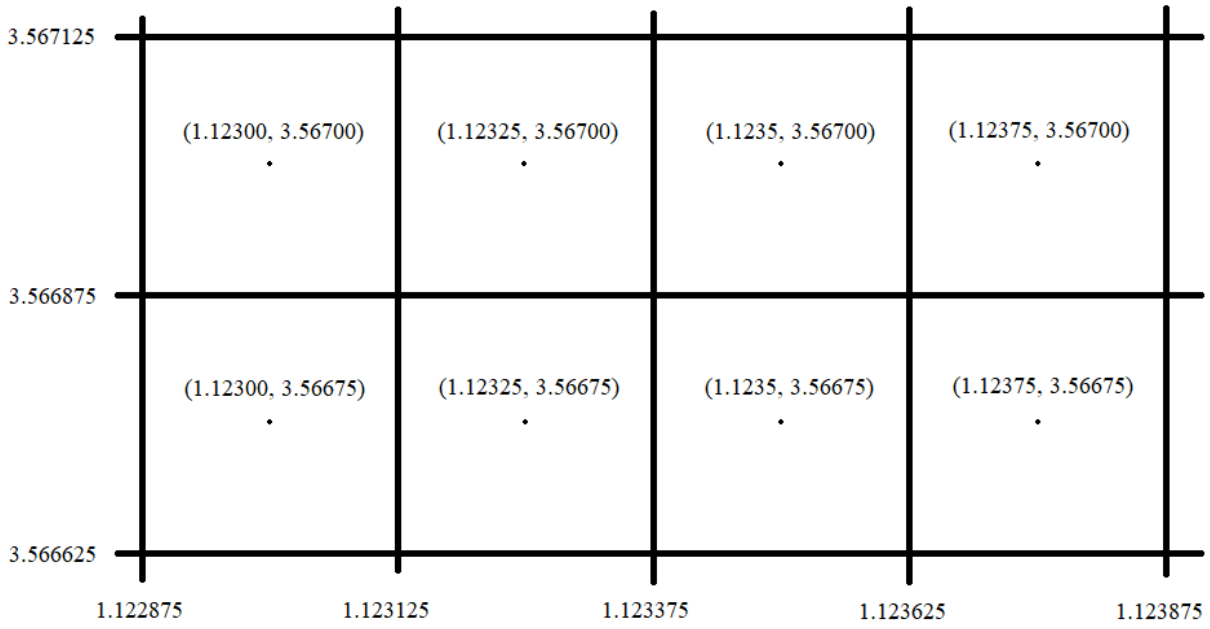
If:

t $\in$ <0.000, 0.000125) to x = 0.000
t $\in$ <0.000125, 0.000375) to x = 0.00025
t $\in$ <0.000375, 0.000625) to x = 0.0005

$t \in$ <0.000625, 0.000875) to x = 0.00075
$t \in$ <0.000875, 0.001 > to x = 0.001

Thanks to the use of roundings, the whole earth is divided into zones. Received coordinates identifies all created sector on earth and at the same time they are also its center. This fulfills the assumption we made at the beginning.

The grid extract looks as follows:



The structure of the sectors based on coordinates causes that the area of each sector, formed from rounding of latitude and longitude, decreases as moves to the poles of the earth. The largest areas will therefore be on the equator and will be around 728m2, which meets our assumption.

We can calculate the number of sectors created in this way. The numbers are rounded to the quarter of a thousand, so each thousandth is divided into 4.

For the longitude, we have a numerical range (-180, 180>. By x, we indicate the number of divisions. It is:

$$x = 360 * 1000 * 4 \qquad (\mathbf{1})$$

For latitude, we have a numerical range <-90, 90>. By y, we indicate the number of divisions. It is:

$$y = 181 * 1000 * 4 \qquad (\mathbf{2})$$

Using (**1**) and (**2**) we calculate the number of sectors for the whole earth:

$$\text{Number of sectors} = x * y = (360 * 1000 * 4) * (181 * 1000 * 4) = 1\,042\,560\,000\,000 \qquad (\mathbf{3})$$

All that remains is construction of the International Postcode (IPCS) for each of the formed/created sectors.

Construction of the International Postcode System(IPCS)

Using Latin alphabets and numbers, we have 10 digits and 26 letters. We remove the letters O and I from the alphabet, (to meet the assumptions of the system). So we have the following 34 chars: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, H, J, K, L, M, N, P, Q, R, S, T, U, V, W, X, Y, Z}. Based on 34 characters, we create a number system – a system of thirty-four. On the left side we have numbers in the system of thirty-four, on the right we have the equivalent in the decimal system.

0 = 0	8 = 8	G = 16	Q = 24	Y = 32	
1 = 1	9 = 9	H = 17	R = 25	Z = 33	
2 = 2	A = 10	J = 18	S = 26		
3 = 3	B = 11	K= 19	T = 27		(**4**)
4 = 4	C = 12	L = 20	U = 28		
5 = 5	D = 13	M = 21	V = 29		
6 = 6	E = 14	N = 22	W=30		
7 = 7	F = 15	P = 23	X = 31		

Each number based on 34 characters can be represented in the decimal system as follows:

... $dcba_{34}$ = f (a) x 34^0 + f (b) x 34^1 + f (c) x 34^2 + f (d) x 34^3 + ..., where f is a function that assigns each digit, from the thirty-four system, its equivalent in the decimal system, (see (4)). Whereas a, b, c, d are digits from the thirty-four systems.

Example:

$AGZ15Z_{34}$ = f(Z) x 34^0 + f(5) x 34^1 + f(1) x 34^2 + f(Z) x 34^3 + f(G) x 34^4 + f(A) x 34^5 =
33 x 34^0 + 5 x 34^1 + 1 x 34^2 + 33 x 34^3 + 16 x 34^4 + 10 x 34^5 =
33 x 1 + 5 x 34 + 1 x 1 156 + 33 x 39 304 + 16 x 1 336 336 + 10 x 45 435 424 =
33 + 170 + 1 156 + 1 297 032 + 21 381 376 + 454 354 240 = 477 034 007

As we can see, the thirty-four record is much shorter. We will need 8 characters to generate postcodes for the whole planet. The maximum number of characters in the current English and Irish postal code system is 7, in Japanese 7 digits, Dutch 6 characters, etc. Considering the length of postal codes in other countries and the fact that the International Postcode System is a global system, we assume that 8 characters are acceptable and meet the assumptions of our system.

The largest number that we can store, using 8 digits in our thirty-four system is (detailed calculations have been omitted):

$$ZZZZZZZZ_{34} = 1\ 785\ 793\ 904\ 895$$

It seems that the earth may have 1 785 793 904 896 postal codes (the number has been increased by 1 in relation to the largest number because 00000000 can also be a code). From the previous calculation (**3**), we know the number of sectors in our structure and it is equal to 1 042 560 000 000. This number is lesser than 1 785 793 904 895, therefore, 8 digits from the thirty-four system will allow to unambiguously define each sector. The sector areas are not equal in size due to the structure. The largest ones are located near the equator and is ≈ 728m2. The earth area of is about 510,100,000 km2 =

510,100,000,000,000 m2. To calculate the average area of a sector, we do a simple operation:

$$510\ 100\ 000\ 000\ 000\ m^2 / 1\ 042\ 560\ 000\ 000 \approx 489.2764\ m^2$$

So the number of $ZZZZZZZZ_{34}$ meets our assumptions, and the area of each sector is lesser than 1000m2.

Now what remains is the conversion of the sector centre to the International Postcode System(IPCS).

For our requirement, we will present integer part of the negative numbers in the following way:

For longitude, the negative integer part x will be converted to 360 + x.
For latitude, the negative integer y will be changed to 180 + y.

The fractional part remains unchanged.

Summarizing:

Longitude -XXX.xxxxx will be converted to (360 + (-XXX)). Xxxxx
Latitude -YYY.yyyyy will be converted to (180 + (-YYY)). Yyyyy

Examples of the conversion:

The number -75.43725 for longitude is 285.43725, because 360 + (-75) = 285.
The number -0.411250 for longitude is 360.411250, because 360 + 0 = 360.
The number -15.44375 for latitude is 165.44375, because 180 + (-15) = 165.

When creating the algorithm we should take into account the fact that in this record, the South Pole of 90 degrees of latitude and the North Pole of- 90 degrees of latitude are the same.

Taking into account the conversion of negative numbers and rounding, the general form of coordinates recorded in the decimal system is:

For longitude: XXX.xxxqq, where XXX $\in$ {x$\in$N: 000 <= x <= 360},
xxx $\in$ {x$\in$N: 000 <= x <= 999},
qq $\in$ {00, 25, 50, 75}

For latitude: YYY.yyygg, where YYY $\in$ {x$\in$N: 000 <= x <= 180},
yyy $\in$ {x$\in$N: 000 <= x <= 999},
gg $\in$ {00, 25, 50, 75}

In the next step, each element of the number XXX.xxxqq and the number YYY.yyygg are converted into a binary system.

XXX_{10} = $XXXXXXXXX_2$ - largest number is 360. We need 9 bits to present this number.
xxx_{10} = $xxxxxxxxxx_2$ - largest number is 999. We need 10 bits to present this number.
qq_{10} = qq_2 - the largest number is 75. However, our set has only 4 elements, that's why our conversion involves assigning 00_{10} => 00_2 , 25_{10} => 01_2 , 50_{10} => 10_2 ,75_{10} => 11_2

YYY_{10} = $YYYYYYYY_2$ - largest number is 180. We need 8 bits to present this number.
yyy_{10} = $yyyyyyyyyy_2$ - largest number is 999. We need 10 bits to present this number.
gg_{10} = gg_2 - the largest number is 75. However, our set has only 4 elements, that's why our conversion involves assigning 00_{10} => 00_2 , 25_{10} => 01_2 , 50_{10} => 10_2 ,75_{10} => 11_2

Then we set the numbers from the binary system in the following order:

$$XXXXXXXXX_2\ YYYYYYYY_2\ xxxxxxxxxx_2\ yyyyyyyyyy_2\ qq_2\ gg_2$$

We take a minimum 41-bit number, zero it and enter the above bits from right to left keeping their order. The number obtained is equal to:

$$XXXXXXXXXYYYYYYYYxxxxxxxxxxyyyyyyyyyyqqgg_2$$

In the last step, we change the binary system into a thirty-four system. The obtained thirty-four number is also our International Postcode (IPCS). Coordinates of the center of the area are encrypted in our record of International Postcode (IPCS).

The order in which the bits are written is very important because, rounded and presented in accordance with our convention, the coordinate with the largest numerical value is 360.99975, 180.99975. Converting it to a binary system we get the number:

$$10110100010110100111110011111111001111111_2 = 1\ 549\ 224\ 509\ 055_{10}$$

The number obtained is lesser than the maximum number of the thirty-four code which is $ZZZZZZZZ_{34}$ = 1 785 793 904 895_{10}. By changing the order of bits we can exceed the maximum value, which will make it impossible to create postal codes for all sectors.

In order to receive coordinates of the center of the sector (International Postcode), we do the operation in reverse order.

Examples:
The International Postcode (IPCS), for the coordinates -1.347120, 53.983489, we are constructing as follows:

1. We round, convert coordinates and get 359.34700, 53.98350
2. We change each element of coordinate, that is: numbers 359, 347, 00, 53, 983, 50, to a binary system as described above. And we get 101100111_2, 0101011011_2, 00_2, 00110101_2, 1111010111_2, 10_2
3. Set the numbers in the correct order 101100111_2, 00110101_2, 0101011011_2, 1111010111_2, 00_2, 10_2
4. We create the number $10110011100110101010101101111110101110010_2$

5. We convert the binary number to thirty-four and get $VCPM6TKY_{34}$
6. The number obtained is the International Postcode (IPCS): **VCPM 6TKY**

Coordinates for the International Postcode 1GL8 YT7F are calculated as follows:

1. We convert the number of thirty-four $1GL8YT7F_{34}$ to binary and we get $000010010001100101100000011010011011010012$
2. Extract the numbers 000010010_2, 00110010_2, 1100000011_2, 0100110110_2, 10_2, 01_2
3. We change the order of 000010010_2, 1100000011_2, 10_2, 00110010_2, 0100110110_2, 01_2
4. We change binary numbers to decimal 18, 771, 50, 50, 310, 25
5. From the numbers obtained, we get coorinate **18.77150, 50.31025**

Function that generates and calculates coordinates from the International Postcode:

Below, there are 2 functions written in C #. The first one constructs the International Postcode, and the second one calculates the coordinates from the International Postcode.

```
public string GeneratePostcode(double Lon, double Lat)
        {
            if ((Lon > 180) || (Lon < -180) || (Lat > 90) || (Lat < -90)) return "";
            if (Lat > 89.99975) Lat = 89.99975;
            if (Lat < -89.99975) Lat = -89.99975;

            string[] Coder = { "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B",
"C", "D", "E", "F", "G", "H", "J", "K", "L", "M", "N", "P", "Q", "R", "S", "T", "U", "V",
"W", "X", "Y", "Z" };

            if (Lat < 0)
            {
                Lat = ((long)Lat + 180) + Math.Abs(Lat - (long)Lat);
            }
            if (Lon < 0)
            {
                Lon = ((long)Lon + 360) + Math.Abs(Lon - (long)Lon);
            }

            string PartPostcodeLat = Lat.ToString("000.000000");
            string PartPostcodeLon = Lon.ToString("000.000000");

            try
            {
                string[] StrsLat = PartPostcodeLat.Split('.');
                string[] StrsLon = PartPostcodeLon.Split('.');

                ulong OptionsLat = 5;
                long Precision = Convert.ToInt32(StrsLat[1].Substring(3));
                if (Precision < 125)
                {
                    OptionsLat = 0;
                }
                else if (Precision < 375)
                {
                    OptionsLat = 1;
                }
                else if (Precision < 625)
                {
```

```
    OptionsLat = 2;
}
else if (Precision < 875)
{
    OptionsLat = 3;
}

ulong TempHiLat = Convert.ToUInt64(StrsLat[0]);
TempHiLat = TempHiLat << 24;
ulong TempLoLat = 0;
if (StrsLat.Length > 1)
{
    TempLoLat = Convert.ToUInt64(StrsLat[1].Substring(0, 3));
    if (OptionsLat == 5)
    {
        TempLoLat++;
    }
    else
    {
        TempHiLat = TempHiLat | OptionsLat;
    }

    TempLoLat = TempLoLat << 4;
}

ulong OptionsLon = 5;
Precision = Convert.ToInt32(StrsLon[1].Substring(3));
if (Precision < 125)
{
    OptionsLon = 0;
}
else if (Precision < 375)
{
    OptionsLon = 1;
}
else if (Precision < 625)
{
    OptionsLon = 2;
}
else if (Precision < 875)
{
    OptionsLon = 3;
}

ulong TempHiLon = Convert.ToUInt64(StrsLon[0]);
TempHiLon = TempHiLon << 32;
ulong TempLoLo = 0;
if (StrsLon.Length > 1)
{
    TempLoLo = Convert.ToUInt64(StrsLon[1].Substring(0, 3));
    if (OptionsLon == 5)
    {
        TempLoLo++;
    }
    else
    {
        TempHiLon = TempHiLon | (OptionsLon * 4);
    }

    TempLoLo = TempLoLo << 14;
}

ulong UlongResHi = TempHiLat | TempHiLon | TempLoLat | TempLoLo;

ulong Pos = 0;
```

```
            string Res = "";
            for (int i = 7; i >= 0; i--)
            {
                Pos = UlongResHi / (ulong)Math.Pow(34, i);
                UlongResHi -= (Pos * (ulong)Math.Pow(34, i));
                Res += Coder[Pos];
            }

            return Res;
        }
        catch
        {
            return "";
        }
    }

public bool ReversePostcode(string Postcode, out double Lat, out double Lon)
    {
        if (Postcode.Length == 8)
        {
            Postcode = Postcode.ToUpper();
            string Coder = "0123456789ABCDEFGHJKLMNPQRSTUVWXYZ";
            ulong Hi = 0;

            for (int i = 0; i < 8; i++)
            {
                Hi += (ulong)Coder.IndexOf(Postcode[i]) * (ulong)Math.Pow(34, (7 - i));
            }

            ulong OptionLat = (Hi & 3);
            double HalfLat = 0;
            switch (OptionLat)
            {
                case 1:
                    HalfLat = 0.00025;
                    break;
                case 2:
                    HalfLat = 0.0005;
                    break;
                case 3:
                    HalfLat = 0.00075;
                    break;
            }

            ulong OptionLon = (Hi & 12) >> 2;
            double HalfLon = 0;
            switch (OptionLon)
            {
                case 1:
                    HalfLon = 0.00025;
                    break;
                case 2:
                    HalfLon = 0.0005;
                    break;
                case 3:
                    HalfLon = 0.00075;
                    break;
            }

            long LoLat = ((long)Hi >> 4) & 1023;
            long HiLat = ((long)Hi >> 24) & 255;
            if (HiLat > 180)
            {
                Lat = -180;
                Lon = -180;
```

```
            return false;
        }
        if (HiLat > 90)
        {
            HiLat = HiLat - 180;
            Lat = HiLat - ((LoLat / 1000.0) + HalfLat);
        }
        else
        {
            Lat = HiLat + ((LoLat / 1000.0) + HalfLat);
        }

        long LoLon = ((long)Hi >> 14) & 1023;
        long HiLon = ((long)Hi >> 32);
        if (HiLon > 360)
        {
            Lat = -180;
            Lon = -180;
            return false;
        }
        if (HiLon > 180)
        {
            HiLon = HiLon - 360;
            Lon = HiLon - ((LoLon / 1000.0) + HalfLon);
        }
        else
        {
            Lon = HiLon + ((LoLon / 1000.0) + HalfLon);
        }

        return true;
    }
    else
    {
        Lat = 0;
        Lon = 0;
        return false;
    }
}
```