Implementation of DateTime Encoder For HTM

Ramesh Kumar
rkumar.panchani@gmail.com

***Abstract*—Based on the technique of Hierarchical Temporal Memory (HTM). HTM is a biologically inspired machine intelligence technology that mimics the architecture and processes of the NeoCortexApi (1.0.5-beta). Therefore, by using that NeoCortexApi (1.0.5-beta) NuGet package, which ease the implementation of the encoder. Since, it has built-in abstract class interface for the Datetime. In the algorithm, it is mentioned that how the encoding process has been done, and how the data and time has been Sparse Distributed Representations (SDRs) for the use in HTM systems, especially for the DateTime encoder. Lastly, I had performed the verifications of my results by performing several unit tests with the encoder.**

I. Introduction

An encoder converts a value to a Sparse Distributed Representation (SDM). SDM is a mathematical model of human long-term memory. According to recent findings in the neuroscience, the brain uses Sparse Distributed Representations (SDRs) to process information. This is true for all mammals, from mice to humans. The SDRs visualize the information processed by the brain at a given moment, each active cell bearing some semantic aspect of the overall message. Sparse means that only a few of the many (thousands of) neurons are active at the same time, in contrast to the typical "dense" representation, in computers, of a few bits of 0s and 1s. Distributed means that not only are the active cells spread across the representation, but the significance of the pattern is too. This makes the SDR resilient to the failure of single neurons and allows sub-sampling. As each bit or neuron has a meaning, if the same bit is active in two SDRs, it means that they are semantically similar: that is the key to our computational approach. Therefore, it is a generalized random-access memory (RAM) for long (e.g., 1,000 bit) binary words. These words serve as both addresses to and data for the memory. Therefore, SDRs are the foundation for all these functions, across all sensory modalities.

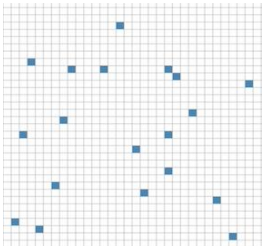

Fig. 1. Sparse distributed memory (SDM)

In this diagram represents a sparse distributed representation, which has couple of thousand cells with a small number of blue cells active. In SDRs, unlike in a dense representations, each bit has meaning. This means that if two vectors have 1s in the same position they are semantically similar in that attribute. SDRs are how brains solve the problem of knowledge representation that has plagued AI for decades. Following are SDR's advantages:

- SDR's are the common data structure in the cortex
- SDR's enable flexible recognition systems that have very high capacity, and are robust to a large amount of noise
- The union property allows a fixed representation to encode a dynamically changing set of patterns
- The analysis of SDR's provides a principled foundation for characterizing the behavior of the HTM learning algorithms and all cognitive functions

Since this project is based on Hierarchical temporal memory (HTM) – which is a biologically constrained theory (or model) of intelligence, originally described in the 2004 in the book of "On Intelligence" by "Jeff Hawkins." HTM is based on neuroscience and the physiology and interaction of pyramidal neurons in the NeoCortexApi (1.0.5-beta) of the mammalian (in particular, human) brain. At the core of HTM are learning algorithms that can store, learn, infer and recall high-order sequences. Unlike most other machine learning methods, HTM learns (in an unsupervised fashion) time-based patterns in unlabeled data on a continuous basis. HTM is robust to noise, and it has high capacity, meaning that it can learn multiple patterns simultaneously. When applied to computers, HTM is well suited for prediction, anomaly detection, classification and ultimately sensory motor applications. The theory has been tested and implemented in software through example applications from Numenta and a few commercial applications from Numenta's partners. Following is the diagrammatical representation of neuron model:

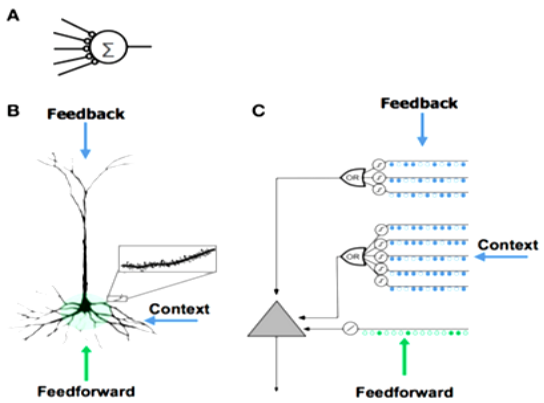

Fig. 2. Artificial Neural network (A), the biological neuron (B), and the HTM neuron (C)

II. METHODOLOGY

A DateTime encoder encodes a date and time according to the encoding parameters specified in its constructor. The input to a date encoder is a DateTime.DateTime object. In results, the output must have to be concatenation of several sub-endings, each of which encodes a different aspect of the date. Which sub-encodings are present, and details of those sub-encodings, are specified in the DateTime encoder constructor.

In DateTime encoder, I have passed two parameters i.e. width (W) and the radius in my code. For the DateTime encoder I had implemented the following formula:

$$n = width * (range / radius) \quad (1)$$

In code, n represents the total number of encoder array, the width represents the all number of 1's, the range represents the total number of specific event (e.g. in the case of Month the range would be from 01 to 12, so the range would be 12 in this case) and the radius represents occurrence of overlapping in the code at given instant.

For the specific given width and the radius values, the encoders will encode the n value. Likewise, I had made some cases of DateTime on consideration of width and radius factor:

Following are the ranges for the DateTime encoder:
Month = 12
Day = 31
Year = 10
Hour = 24
Min = 60
Sec = 60

Now, we are looking for the approaches to make DateTime encoder, therefore I am writing down 02 cases with an example to show how this encoder is working correctly, along with explanations and visual representations:

Case 01: When width = 3 and radius = 3

Let's assume we had to make an encoder for this date and time i.e. "11/04/2010 14:55:00". And its output would be look like after concatenation:

1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,1,1,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

Here yellow highlighter represents the bits of 11th month, green highlighter represents the bits of 04th day, sky blue highlighter represents the bits of 2010 year, orange highlighter represents the bits of 14 hour, blue highlighter represents the bits of 55th minute, and silver highlighter represents the bits of 00th second.

In order to make concatenation encoder for specific date and time, we have to check binary numbers for all conditions and one of the conditions must satisfy each sub encoder i.e. for month, day, year, hour, minute and second.

Let's considering the case, when width is 3 and radius = 3, then the encoder can be developed by following way:

Let's make encoder for "11/04/2010 14:55:00" date and time. I had highlighted the encoder with bold letters and used yellow highlighter to show how overlapping is done in encoder for every month, day, year, hour, minute and second:

Month: (3*(12/3)) = 12
01: 111000000000
02: 011100000000
03: 001110000000
04: 000111000000
05: 000011100000
06: 000001110000
07: 000000111000
08: 000000011100
09: 000000001110
10: 000000000111
11: 100000000011
12: 110000000001

Overlapping = radius – 1
Since, the radius = 3
Overlapping = 3 – 1 = 2

When the month is 11, we will choose this encoder

Day: (3*(31/3)) = 31
01: 1110000000000000000000000000000
02: 0111000000000000000000000000000
03: 0011100000000000000000000000000
04: 0001110000000000000000000000000
05: 0000111000000000000000000000000
06: 0000011100000000000000000000000
07: 0000001110000000000000000000000
08: 0000000111000000000000000000000
09: 0000000011100000000000000000000
10: 0000000001110000000000000000000
11: 0000000000111000000000000000000
12: 0000000000011100000000000000000
13: 0000000000001110000000000000000
14: 0000000000000111000000000000000
15: 0000000000000011100000000000000
16: 0000000000000001110000000000000
17: 0000000000000000111000000000000
18: 0000000000000000011100000000000
19: 0000000000000000001110000000000
20: 0000000000000000000111000000000
21: 0000000000000000000011100000000
22: 0000000000000000000001110000000
23: 0000000000000000000000111000000
24: 0000000000000000000000011100000
25: 0000000000000000000000001110000
26: 0000000000000000000000000111000
27: 0000000000000000000000000011100
28: 0000000000000000000000000001110
29: 0000000000000000000000000000111
30: 1000000000000000000000000000011
31: 1100000000000000000000000000001

When the day is 04, we will choose this encoder

Year: (3*(10/3)) = 10
00: 1110000000
01: 0111000000
02: 0011100000
03: 0001110000
04: 0000111000
05: 0000011100
06: 0000001110
07: 0000000111
08: 1000000011
09: 1100000001

When the year is 2010, we will choose this encoder

Hour: (3*(24/3)) = 24
00: 111000000000000000000000
01: 011100000000000000000000
02: 001110000000000000000000
03: 000111000000000000000000
04: 000011100000000000000000

05: 000001110000000000000000
06: 000000111000000000000000
07: 000000011100000000000000
08: 000000001110000000000000
09: 000000000111000000000000
10: 000000000011100000000000
11: 000000000001110000000000
12: 000000000000111000000000
13: 000000000000011100000000
14: 000000000000001110000000
15: 000000000000000111000000
16: 000000000000000011100000
17: 000000000000000001110000
18: 000000000000000000111000
19: 000000000000000000011100
20: 000000000000000000001110
21: 000000000000000000000111
22: 100000000000000000000011
23: 110000000000000000000001

When the hour is 14, we will choose this encoder

Minute: (3*(60/3)) = 60

00: 111000000000000000000000000000000000000000000000000000000000
01: 011100000000000000000000000000000000000000000000000000000000
02: 001110000000000000000000000000000000000000000000000000000000
03: 000111000000000000000000000000000000000000000000000000000000
04: 000011100000000000000000000000000000000000000000000000000000
05: 000001110000000000000000000000000000000000000000000000000000
06: 000000111000000000000000000000000000000000000000000000000000
07: 000000011100000000000000000000000000000000000000000000000000
08: 000000001110000000000000000000000000000000000000000000000000
09: 000000000111000000000000000000000000000000000000000000000000
10: 000000000011100000000000000000000000000000000000000000000000
11: 000000000001110000000000000000000000000000000000000000000000
12: 000000000000111000000000000000000000000000000000000000000000
13: 000000000000011100000000000000000000000000000000000000000000
14: 000000000000001110000000000000000000000000000000000000000000
15: 000000000000000111000000000000000000000000000000000000000000
16: 000000000000000011100000000000000000000000000000000000000000
17: 000000000000000001110000000000000000000000000000000000000000
18: 000000000000000000111000000000000000000000000000000000000000
19: 000000000000000000011100000000000000000000000000000000000000
20: 000000000000000000001110000000000000000000000000000000000000
21: 000000000000000000000111000000000000000000000000000000000000
22: 000000000000000000000011100000000000000000000000000000000000
23: 000000000000000000000001110000000000000000000000000000000000
24: 000000000000000000000000111000000000000000000000000000000000
25: 000000000000000000000000011100000000000000000000000000000000
26: 000000000000000000000000001110000000000000000000000000000000
27: 000000000000000000000000000111000000000000000000000000000000
28: 000000000000000000000000000011100000000000000000000000000000
29: 000000000000000000000000000001110000000000000000000000000000
30: 000000000000000000000000000000111000000000000000000000000000
31: 000000000000000000000000000000011100000000000000000000000000
32: 000000000000000000000000000000001110000000000000000000000000
33: 000000000000000000000000000000000111000000000000000000000000
34: 000000000000000000000000000000000011100000000000000000000000
35: 000000000000000000000000000000000001110000000000000000000000
36: 000000000000000000000000000000000000111000000000000000000000
37: 000000000000000000000000000000000000011100000000000000000000
38: 000000000000000000000000000000000000001110000000000000000000
39: 000000000000000000000000000000000000000111000000000000000000
40: 000000000000000000000000000000000000000011100000000000000000
41: 000000000000000000000000000000000000000001110000000000000000
42: 000000000000000000000000000000000000000000111000000000000000
43: 000000000000000000000000000000000000000000011100000000000000
44: 000000000000000000000000000000000000000000001110000000000000
45: 000000000000000000000000000000000000000000000111000000000000
46: 000000000000000000000000000000000000000000000011100000000000
47: 000000000000000000000000000000000000000000000001110000000000
48: 000000000000000000000000000000000000000000000000111000000000
49: 000000000000000000000000000000000000000000000000011100000000
50: 000000000000000000000000000000000000000000000000001110000000
51: 000000000000000000000000000000000000000000000000000111000000
52: 000000000000000000000000000000000000000000000000000011100000
53: 000000000000000000000000000000000000000000000000000001110000
54: 000000000000000000000000000000000000000000000000000000111000
55: 000000000000000000000000000000000000000000000000000000011100
56: 000000000000000000000000000000000000000000000000000000001110
57: 000000000000000000000000000000000000000000000000000000000111
58: 100000000000000000000000000000000000000000000000000000000011
59: 110000000000000000000000000000000000000000000000000000000001

When the minute is 55, we will choose

Second: (3*(60/3)) = 60

00: 111000000000000000000000000000000000000000000000000000000000
01: 011100000000000000000000000000000000000000000000000000000000
02: 001110000000000000000000000000000000000000000000000000000000
03: 000111000000000000000000000000000000000000000000000000000000
04: 000011100000000000000000000000000000000000000000000000000000

When the second is 00, we will choose this encoder

05: 000001110000000000000000000000000000000000000000000000000000
06: 000000111000000000000000000000000000000000000000000000000000
07: 000000011100000000000000000000000000000000000000000000000000
08: 000000001110000000000000000000000000000000000000000000000000
09: 000000000111000000000000000000000000000000000000000000000000
10: 000000000011100000000000000000000000000000000000000000000000
11: 000000000001110000000000000000000000000000000000000000000000
12: 000000000000111000000000000000000000000000000000000000000000
13: 000000000000011100000000000000000000000000000000000000000000
14: 000000000000001110000000000000000000000000000000000000000000
15: 000000000000000111000000000000000000000000000000000000000000
16: 000000000000000011100000000000000000000000000000000000000000
17: 000000000000000001110000000000000000000000000000000000000000
18: 000000000000000000111000000000000000000000000000000000000000
19: 000000000000000000011100000000000000000000000000000000000000
20: 000000000000000000001110000000000000000000000000000000000000
21: 000000000000000000000111000000000000000000000000000000000000
22: 000000000000000000000011100000000000000000000000000000000000
23: 000000000000000000000001110000000000000000000000000000000000
24: 000000000000000000000000111000000000000000000000000000000000
25: 000000000000000000000000011100000000000000000000000000000000
26: 000000000000000000000000001110000000000000000000000000000000
27: 000000000000000000000000000111000000000000000000000000000000
28: 000000000000000000000000000011100000000000000000000000000000
29: 000000000000000000000000000001110000000000000000000000000000
30: 000000000000000000000000000000111000000000000000000000000000
31: 000000000000000000000000000000011100000000000000000000000000
32: 000000000000000000000000000000001110000000000000000000000000
33: 000000000000000000000000000000000111000000000000000000000000
34: 000000000000000000000000000000000011100000000000000000000000
35: 000000000000000000000000000000000001110000000000000000000000
36: 000000000000000000000000000000000000111000000000000000000000
37: 000000000000000000000000000000000000011100000000000000000000
38: 000000000000000000000000000000000000001110000000000000000000
39: 000000000000000000000000000000000000000111000000000000000000
40: 000000000000000000000000000000000000000011100000000000000000
41: 000000000000000000000000000000000000000001110000000000000000
42: 000000000000000000000000000000000000000000111000000000000000
43: 000000000000000000000000000000000000000000011100000000000000
44: 000000000000000000000000000000000000000000001110000000000000
45: 000000000000000000000000000000000000000000000111000000000000
46: 000000000000000000000000000000000000000000000011100000000000
47: 000000000000000000000000000000000000000000000001110000000000
48: 000000000000000000000000000000000000000000000000111000000000
49: 000000000000000000000000000000000000000000000000011100000000
50: 000000000000000000000000000000000000000000000000001110000000
51: 000000000000000000000000000000000000000000000000000111000000
52: 000000000000000000000000000000000000000000000000000011100000
53: 000000000000000000000000000000000000000000000000000001110000
54: 000000000000000000000000000000000000000000000000000000111000
55: 000000000000000000000000000000000000000000000000000000011100
56: 000000000000000000000000000000000000000000000000000000001110
57: 000000000000000000000000000000000000000000000000000000000111
58: 100000000000000000000000000000000000000000000000000000000011
59: 110000000000000000000000000000000000000000000000000000000001

By using above way, we will encode all date time encoder. In this case we did it when our width and radius was equal to 3.

Case 02: When width = 6 and radius = 6

Let's assume we had to make an encoder for this date and time i.e. "03/13/2008 07:34:27". And it's output would be look like after concatenation:

0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,1,1,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

Here yellow highlighter represents the bits of 03rd month, green highlighter represents the bits of 13th day, sky blue highlighter represents the bits of 2008 year, orange highlighter represents the bits of 07 hour, blue highlighter

represents the bits of 34th minute, and silver highlighter represents the bits of 27th second.

Concatenation encoder output for this specific DateTime encoder event, when width = 6 and radius = 6 is:

Month: (6*(12/6)) = 12

01: 111111000000
02: 011111100000
03: 001111110000
04: 000111111000
05: 000011111100
06: 000001111110
07: 000000111111
08: 100000011111
09: 110000001111
10: 111000000111
11: 111100000011
12: 111110000001

Overlapping = radius – 1
Since, the radius = 6
Overlapping = 6 – 1 = 5

Day: (6*(31/6)) = 31

01: 11111100000000000000000000000000
02: 01111110000000000000000000000000
03: 00111111000000000000000000000000
04: 00011111100000000000000000000000
05: 00001111110000000000000000000000
06: 00000111111000000000000000000000
07: 00000011111100000000000000000000
08: 00000001111110000000000000000000
09: 00000000111111000000000000000000
10: 00000000011111100000000000000000
11: 00000000001111110000000000000000
12: 00000000000111111000000000000000
13: 00000000000011111100000000000000
14: 00000000000001111110000000000000
15: 00000000000000111111000000000000
16: 00000000000000011111100000000000
17: 00000000000000001111110000000000
18: 00000000000000000111111000000000
19: 00000000000000000011111100000000
20: 00000000000000000001111110000000
21: 00000000000000000000111111000000
22: 00000000000000000000011111110000
23: 00000000000000000000001111111000
24: 00000000000000000000000111111100
25: 00000000000000000000000011111110
26: 00000000000000000000000001111111
27: 10000000000000000000000000011111
28: 11000000000000000000000000001111
29: 11100000000000000000000000000111
30: 11110000000000000000000000000011
31: 11111000000000000000000000000001

Year: (6*(10/6)) = 10

00: 1111110000
01: 0111111000
02: 0011111100
03: 0001111110
04: 0000111111
05: 1000011111
06: 1100001111
07: 1110000111
08: 1111000011
09: 1111100001

Hour: (6*(24/6)) = 24

00: 11111100000000000000000
01: 01111110000000000000000
02: 00111111000000000000000
03: 00011111100000000000000
04: 00001111110000000000000
05: 00000111111000000000000
06: 00000011111100000000000
07: 00000001111110000000000
08: 00000000111111000000000
09: 00000000011111100000000
10: 00000000001111110000000
11: 00000000000111111000000
12: 00000000000011111100000
13: 00000000000001111110000
14: 00000000000000111111000
15: 00000000000000011111100
16: 00000000000000001111110
17: 00000000000000000111111
18: 00000000000000000111111
19: 10000000000000000011111
20: 11000000000000000001111
21: 11100000000000000000111
22: 11110000000000000000011
23: 11111000000000000000001

Minute: (6*(60/6)) = 60

00: 11111100000000000000000000000000000000000000000000000000000000
01: 01111110000000000000000000000000000000000000000000000000000000
02: 00111111000000000000000000000000000000000000000000000000000000
03: 00011111100000000000000000000000000000000000000000000000000000
04: 00001111110000000000000000000000000000000000000000000000000000
05: 00000111111000000000000000000000000000000000000000000000000000
06: 00000011111100000000000000000000000000000000000000000000000000
07: 00000001111110000000000000000000000000000000000000000000000000
08: 00000000111111000000000000000000000000000000000000000000000000
09: 00000000011111100000000000000000000000000000000000000000000000
10: 00000000001111110000000000000000000000000000000000000000000000
11: 00000000000111111000000000000000000000000000000000000000000000
12: 00000000000011111100000000000000000000000000000000000000000000
13: 00000000000001111110000000000000000000000000000000000000000000
14: 00000000000000111111000000000000000000000000000000000000000000
15: 00000000000000011111100000000000000000000000000000000000000000
16: 00000000000000001111110000000000000000000000000000000000000000
17: 00000000000000000111111000000000000000000000000000000000000000
18: 00000000000000000011111100000000000000000000000000000000000000
19: 00000000000000000001111110000000000000000000000000000000000000
20: 00000000000000000000111111000000000000000000000000000000000000
21: 00000000000000000000011111110000000000000000000000000000000000
22: 00000000000000000000001111111000000000000000000000000000000000
23: 00000000000000000000000111111100000000000000000000000000000000
24: 00000000000000000000000011111110000000000000000000000000000000
25: 00000000000000000000000001111111000000000000000000000000000000
26: 00000000000000000000000000111111100000000000000000000000000000
27: 00000000000000000000000000011111110000000000000000000000000000
28: 00000000000000000000000000001111111000000000000000000000000000
29: 00000000000000000000000000000111111100000000000000000000000000
30: 00000000000000000000000000000011111110000000000000000000000000
31: 00000000000000000000000000000001111111000000000000000000000000
32: 00000000000000000000000000000000111111100000000000000000000000
33: 00000000000000000000000000000000011111110000000000000000000000
34: 00000000000000000000000000000000001111111000000000000000000000
35: 00000000000000000000000000000000000111111100000000000000000000
36: 00000000000000000000000000000000000011111110000000000000000000
37: 00000000000000000000000000000000000001111111000000000000000000
38: 00000000000000000000000000000000000000111111100000000000000000
39: 00000000000000000000000000000000000000011111110000000000000000
40: 00000000000000000000000000000000000000001111111000000000000000
41: 00000000000000000000000000000000000000000111111100000000000000
42: 00000000000000000000000000000000000000000011111110000000000000
43: 00000000000000000000000000000000000000000001111111000000000000
44: 00000000000000000000000000000000000000000000111111100000000000
45: 00000000000000000000000000000000000000000000011111110000000000
46: 00000000000000000000000000000000000000000000001111111000000000
47: 00000000000000000000000000000000000000000000000111111100000000
48: 00000000000000000000000000000000000000000000000011111110000000
49: 00000000000000000000000000000000000000000000000001111111000000
50: 00000000000000000000000000000000000000000000000000111111100000
51: 00000000000000000000000000000000000000000000000000011111110000
52: 00000000000000000000000000000000000000000000000000001111111000
53: 00000000000000000000000000000000000000000000000000000111111100
54: 00000000000000000000000000000000000000000000000000000011111110
55: 00000000000000000000000000000000000000000000000000000001111111
56: 10000000000000000000000000000000000000000000000000000000111111
57: 11000000000000000000000000000000000000000000000000000000011111
58: 11100000000000000000000000000000000000000000000000000000001111
59: 11110000000000000000000000000000000000000000000000000000000111
60: 11111000000000000000000000000000000000000000000000000000000011
61: 11111100000000000000000000000000000000000000000000000000000001

Second: (6*(60/6)) = 60

00: 11111100000000000000000000000000000000000000000000000000000000
01: 01111110000000000000000000000000000000000000000000000000000000
02: 00111111000000000000000000000000000000000000000000000000000000
03: 00011111100000000000000000000000000000000000000000000000000000
04: 00001111110000000000000000000000000000000000000000000000000000
05: 00000111111000000000000000000000000000000000000000000000000000
06: 00000011111100000000000000000000000000000000000000000000000000
07: 00000001111110000000000000000000000000000000000000000000000000
08: 00000000111111000000000000000000000000000000000000000000000000
09: 00000000011111100000000000000000000000000000000000000000000000
10: 00000000001111110000000000000000000000000000000000000000000000
11: 00000000000111111000000000000000000000000000000000000000000000
12: 00000000000011111100000000000000000000000000000000000000000000
13: 00000000000001111110000000000000000000000000000000000000000000
14: 00000000000000111111000000000000000000000000000000000000000000
15: 00000000000000011111100000000000000000000000000000000000000000
16: 00000000000000001111110000000000000000000000000000000000000000
17: 00000000000000000111111000000000000000000000000000000000000000

18: 000000000000000000111111000000000000000000000000000000000000
19: 000000000000000000011111100000000000000000000000000000000000
20: 000000000000000000001111110000000000000000000000000000000000
21: 000000000000000000000111111000000000000000000000000000000000
22: 000000000000000000000011111100000000000000000000000000000000
23: 000000000000000000000001111110000000000000000000000000000000
24: 000000000000000000000000111111000000000000000000000000000000
25: 000000000000000000000000011111100000000000000000000000000000
26: 000000000000000000000000001111110000000000000000000000000000
27: 000000000000000000000000000111111000000000000000000000000000
28: 000000000000000000000000000011111100000000000000000000000000
29: 000000000000000000000000000001111110000000000000000000000000
30: 000000000000000000000000000000111111000000000000000000000000
31: 000000000000000000000000000000011111100000000000000000000000
32: 000000000000000000000000000000001111110000000000000000000000
33: 000000000000000000000000000000000111111000000000000000000000
34: 000000000000000000000000000000000011111100000000000000000000
35: 000000000000000000000000000000000001111110000000000000000000
36: 000000000000000000000000000000000000111111000000000000000000
37: 000000000000000000000000000000000000011111100000000000000000
38: 000000000000000000000000000000000000001111110000000000000000
39: 000000000000000000000000000000000000000111111000000000000000
40: 000000000000000000000000000000000000000011111100000000000000
41: 000000000000000000000000000000000000000001111110000000000000
42: 000000000000000000000000000000000000000000111111000000000000
43: 000000000000000000000000000000000000000000011111100000000000
44: 000000000000000000000000000000000000000000001111110000000000
45: 000000000000000000000000000000000000000000000111111000000000
46: 000000000000000000000000000000000000000000000011111100000000
47: 000000000000000000000000000000000000000000000001111110000000
48: 000000000000000000000000000000000000000000000000111111000000
49: 000000000000000000000000000000000000000000000000011111100000
50: 000000000000000000000000000000000000000000000000001111110000
51: 000000000000000000000000000000000000000000000000000111111000
52: 000000000000000000000000000000000000000000000000000011111100
53: 000000000000000000000000000000000000000000000000000001111110
54: 000000000000000000000000000000000000000000000000000000111111
55: 100000000000000000000000000000000000000000000000000000011111
56: 110000000000000000000000000000000000000000000000000000001111
57: 111000000000000000000000000000000000000000000000000000000111
58: 111100000000000000000000000000000000000000000000000000000011
59: 111110000000000000000000000000000000000000000000000000000001

By using above way, we will encode all date time encoder. In this case we did it when our width and radius was equal to 6.

Example 01:

1) Width = 3;

Radius = 3; (means 2 overlapping will occur)

Since we have *n = width * (range / radius)*, Therefore, *n = 3 * (range/ 3).*
Considering case for the specific month, we have month starting from January till December, since the range is 12 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, and 12, Therefore for the month *n = (3 * (12/3)) = 12*.

For the specific date, we have date which will start from the 1st to 31st, since the range is 31 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, and 31. Therefore for the date *n = (3*(31/3)) =31.*

For the specific year, we have year which will start from the e.g. 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, and 2009, since rest are the repetitions of two last bits, therefore, we can conclude for the year we have range of 10 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, and 09. Therefore for the year *n = (3*(10/3)) = 10.*

For the specific hour, we have hour which will start from the 00 to 23, since the range is 24 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, and 23. Therefore for the hour *n = (3*(24/3)) = 24.*

For the specific minute, we have minute which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the minute *n = (3*(60/3)) = 60.*

For the specific second, we have second which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the second *n = (3*(60/3)) = 60.*

Concatenation in this case would be, encoder = n = (month + date + year + hour + minute + second) = (12 + 31 + 10 + 24 + 60 + 60) = 197, this means that the final encoder would consist total of 197 bits for the case when width = 3 and the radius = 3.

Following are the examples how would encoder look like under the above mentioned results:

"11/04/2010 14:55:00"
Concatenation Encoder output for this specific DateTime encoder event, when width = 3 and radius = 3 is:
1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

"04/25/2013 08:10:58"
Concatenation Encoder output for this specific DateTime encoder event, when width = 3 and radius = 3 is:
0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,1,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,1

"12/12/2012 12:12:12"
Concatenation Encoder output for this specific DateTime encoder event, when width = 3 and radius = 3 is:
1,1,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

"01/10/2022 21:13:43"
Concatenation Encoder output for this specific DateTime encoder event, when width = 3 and radius = 3 is:
1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,
0,0,0,0,0

"07/18/2016 20:04:15"
Concatenation Encoder output for this specific DateTime encoder event, when width = 3 and radius = 3 is:
0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

"03/13/2008 07:34:27"
Concatenation Encoder output for this specific DateTime encoder event, when width = 3 and radius = 3 is:
0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,1,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

Example 02:
2) Width = 2;
Radius = 1; (means 0 overlapping will occur)

Since we have *n = width * (range / radius)*, Therefore, *n = 2 * (range/ 1).*
Considering case for the specific month, we have month starting from January till December, since the range is 12 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, and 12, Therefore for the month *n = (2 * (12/1)) = 24*.

For the specific date, we have date which will start from the 1st to 31st, since the range is 31 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, and 31. Therefore for the date *n = (2*(31/1)) = 62.*

For the specific year, we have year which will start from the e.g. 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, and 2009, since rest are the repetitions of two last bits, therefore, we can conclude for the year we have range of 10 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, and 09. Therefore for the year *n = (2*(10/1)) = 20.*

For the specific hour, we have hour which will start from the 00 to 23, since the range is 24 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, and 23. Therefore for the hour *n = (2*(24/1)) = 48.*

For the specific minute, we have minute which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the minute *n = (2*(60/1)) = 120.*

For the specific second, we have second which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the second *n = (2*(60/1)) = 120.*

Concatenation in this case would be, encoder = n = (month + date + year + hour + minute + second) = (24 + 62 + 20 + 48 + 120 + 120) = 394, this means that the final encoder would consist total of 394 bits for the case when width = 2 and the radius = 1.

Following are the examples how would encoder look like under the above mentioned results:

"11/04/2010 14:55:00"
Concatenation Encoder output for this specific DateTime encoder event, when width = 2 and radius = 1 is:
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0

"02/21/2019 13:51:43"
Concatenation Encoder output for this specific DateTime encoder event, when width = 2 and radius = 1 is:
0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0

"07/12/2009 01:38:23"
Concatenation Encoder output for this specific DateTime encoder event, when width = 2 and radius = 1 is:

0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,1,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0

"12/05/2002 21:09:35"
Concatenation Encoder output for this specific DateTime encoder event, when width = 2 and radius = 1 is:
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0

"10/03/2005 15:11:56"
Concatenation Encoder output for this specific DateTime encoder event, when width = 2 and radius = 1 is:
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,1,1,0,0,0,0,0,0

"01/27/2012 19:01:54"
Concatenation Encoder output for this specific DateTime encoder event, when width = 2 and radius = 1 is:
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,1,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,
0,0,0,0,0,0,0,0,0,0

"03/24/2015 10:06:34"
Concatenation Encoder output for this specific DateTime encoder event, when width = 2 and radius = 1 is:
0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0

Example 03:

3) Width = 6;

Radius = 6; (means 5 overlapping will occur)

Since we have *n = width * (range / radius)*, Therefore, *n = 6 * (range/ 6).*
Considering case for the specific month, we have month starting from January till December, since the range is 12 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, and 12, Therefore for the month *n = (6 * (12/6)) = 12.*

For the specific date, we have date which will start from the 1^{st} to 31^{st}, since the range is 31 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, and 31. Therefore for the date *n = (6*(31/6)) =31.*

For the specific year, we have year which will start from the e.g. 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, and 2009, since rest are the repetitions of two last bits, therefore, we can conclude for the year we have range of 10 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, and 09. Therefore for the year *n = (6*(10/6)) = 10.*

For the specific hour, we have hour which will start from the 00 to 23, since the range is 24 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, and 23. Therefore for the hour *n = (6*(24/6)) = 24.*

For the specific minute, we have minute which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the minute *n = (6*(60/6)) = 60.*

For the specific second, we have second which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,

52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the second *n = (6*(60/6)) = 60*.

Concatenation in this case would be, encoder = n = (month + date + year + hour + minute + second) = (12 + 31 + 10 + 24 + 60 + 60) = 197, this means that the final encoder would consist total of 197 bits for the case when width = 6 and the radius = 6.

Following are the examples how would encoder look like under the above mentioned results:

"11/04/2010 14:55:00"
Concatenation Encoder output for this specific DateTime encoder event, when width = 6 and radius = 6 is:
1,1,1,1,0,0,0,0,0,0,1,1,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

"03/13/2008 07:34:27"
Concatenation Encoder output for this specific DateTime encoder event, when width = 6 and radius = 6 is:
0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,1,1,0,0,0,0,0,0,0,1,1,1,1,
1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

"07/17/2002 22:46:50"
Concatenation Encoder output for this specific DateTime encoder event, when width = 6 and radius = 6 is:
0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,
1,1,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,1,1,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,
1,0,0,0,0

"09/05/2001 18:06:59"
Concatenation Encoder output for this specific DateTime encoder event, when width = 6 and radius = 6 is:
1,1,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1

"01/31/2014 05:19:33"
Concatenation Encoder output for this specific DateTime encoder event, when width = 6 and radius = 6 is:
1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,1,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
1,1,1,1,1,1,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

"12/25/2030 19:29:04"
Concatenation Encoder output for this specific DateTime encoder event, when width = 6 and radius = 6 is:
1,1,1,1,1,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,1,1,1,1,1,0,1,1,1,1,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

Example 04:
4) Width = 1;
Radius = 1; (means 0 overlapping will occur)

Since we have *n = width * (range / radius)*, Therefore, *n = 1 * (range/ 1).*
Considering case for the specific month, we have month starting from January till December, since the range is 12 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, and 12, Therefore for the month *n = (1 * (12/1)) = 12*.

For the specific date, we have date which will start from the 1st to 31st, since the range is 31 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, and 31. Therefore for the date *n = (1*(31/1)) =31*.

For the specific year, we have year which will start from the e.g. 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, and 2009, since rest are the repetitions of two last bits, therefore, we can conclude for the year we have range of 10 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, and 09. Therefore for the year *n = (1*(10/1)) = 10*.

For the specific hour, we have hour which will start from the 00 to 23, since the range is 24 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, and 23. Therefore for the hour *n = (1*(24/1)) = 24*.

For the specific minute, we have minute which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the minute *n = (1*(60/1)) = 60*.

For the specific second, we have second which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the second *n = (1*(60/1)) = 60*.

Concatenation in this case would be, encoder = n = (month + date + year + hour + minute + second) = (12 + 31 + 10 + 24 + 60 + 60) = 197, this means that the final encoder would consist total of 197 bits for the case when width = 1 and the radius = 1.

Following are the examples how would encoder look like under the above mentioned results:

"05/01/2011 20:55:06"
Concatenation Encoder output for this specific DateTime encoder event, when width = 1 and radius = 1 is:
0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

"06/02/2012 21:56:07"
Concatenation Encoder output for this specific DateTime encoder event, when width = 1 and radius = 1 is:
0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

"07/03/2013 19:57:08"
Concatenation Encoder output for this specific DateTime encoder event, when width = 1 and radius = 1 is:
0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0
"07/03/2013 19:57:08"
Concatenation Encoder output for this specific DateTime encoder event, when width = 1 and radius = 1 is:
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

"07/03/2013 19:57:09"
Concatenation Encoder output for this specific DateTime encoder event, when width = 1 and radius = 1 is:
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0

Likewise, one can create any type of encoder, by following the above mentioned conditions. There are certain violations but still one can make most of the DateTime encoders.

Methods of code implementation for the datetime encoder:

I have used following methods for the making of date time encoder:

This is the constructor for the width and the radius:

```
/// <summary>
/// For the setting of radius and the width
/// </summary>
/// <param name="settings"></param>
public DateTimeEncoder(Dictionary<string, object> settings)
{
    if (settings.TryGetValue("Radius", out object radius) && (double)radius > 0)
    {
        this.Radius = (double)radius;
    }
    else
    {
        this.Radius = 1;
    }

    if (settings.TryGetValue("W", out object width) && (int)width > 0)
    {
        this.W = (int)width;
    }
    else
    {
        this.W = 1;
    }
}
```

In public override method, I had implemented the formula of n = (width * range/radius). Since, for the monthArray, we have range of 12, for the dayArray, we have the range of 31, for the yearArray, we have range of 10, for the hourArray, we have the range of 24, for the minuteArray and secondArray, we have the range of 60. I had divided the range with radius and then multiplied it with the width to get final answer in form of array: Following is the implementation:

```
/// <summary>
/// Implementation of n = width * (radius / range) forumla.
/// </summary>
/// <param name="inputData"></param>
/// <returns></returns>
int[] monthArray = encoding(inputDateTime.Month - 1,
(int)(W * 12 / Radius));

int[] dayArray = encoding(inputDateTime.Day - 1, (int)(W *
31 / Radius));

int[] yearArray = encoding(inputDateTime.Year, (int)(W * 10
/ Radius));

int[] hourArray = encoding(inputDateTime.Hour, (int)(W * 24
/ Radius));

int[] minuteArray = encoding(inputDateTime.Minute, (int)(W
* 60 / Radius));

int[] secondArray = encoding(inputDateTime.Second, (int)(W
* 60 / Radius));
```

For the concatenation I had added all the arrays (i.e. monthArray, dayArray, yearArray, hourArray, minuteArray, and secondArray) together and then finally returned the outArray:

```
int[] outArray = new int[monthArray.Length +
dayArray.Length + yearArray.Length + hourArray.Length +
minuteArray.Length + secondArray.Length];

monthArray.CopyTo(outArray, 0);
```

```
dayArray.CopyTo(outArray, monthArray.Length);

yearArray.CopyTo(outArray, monthArray.Length +
dayArray.Length);

hourArray.CopyTo(outArray, monthArray.Length +
dayArray.Length + yearArray.Length);

minuteArray.CopyTo(outArray, monthArray.Length +
dayArray.Length + yearArray.Length + hourArray.Length);

secondArray.CopyTo(outArray, monthArray.Length +
dayArray.Length + yearArray.Length + hourArray.Length +
minuteArray.Length);
```

In the encoding method, I had implemented the condition of correct execution of the encoder using a for loop:

```
/// <summary>
/// Implemented logical condition for encoding bits of date and time.
/// </summary>
/// <param name="element"></param>
/// <param name="numberOfBits"></param>
/// <returns></returns>
private int[] Encoding(int element, int numberOfBits)
{
    int[] encoderArray = new int[numberOfBits];

    foreach (var item in encoderArray)
    {
        encoderArray[item] = 0;
    }

    for (int i = element * (W - (int)Radius + 1); i < element * (W - (int)Radius + 1) + W; i++)
    {
        int j = i % encoderArray.Length;
        encoderArray[j] = 1;
    }

    return encoderArray;
}
```

Here, I have implemented the date encoder only and I had performed multiple unit tests for date encoder as well:

```
/// <summary>
/// Implementation of only Date encoder.
/// </summary>
public int[] EncodeDateOnly(object inputData)
{
    DateTime inputDate = DateTime.Parse((String)inputData);
    int[] monthArray = encoding(inputDate.Month - 1, (int)(W * 12 / Radius));
    int[] dayArray = encoding(inputDate.Day - 1, (int)(W * 31 / Radius));
    int[] yearArray = encoding(inputDate.Year, (int)(W * 10 / Radius));

    int[] dateArray = new int[monthArray.Length + dayArray.Length + yearArray.Length];
    foreach (var item in dateArray)
    {
        dateArray[item] = 0;
    }
    monthArray.CopyTo(dateArray, 0);
    dayArray.CopyTo(dateArray, monthArray.Length);
    yearArray.CopyTo(dateArray, monthArray.Length + dayArray.Length);
    return dateArray;
}
```

For the compilation of unit test part of date encoder only following is the code:

```
/// <summary>
/// performing number of unit tests to verify the date encoder only.
/// </summary>
/// <param name="input"></param>
/// <param name="expectedOutput"></param>
[TestMethod]
public void EncodeDateOnlyTest(Object input, int[] expectedOutput)
{
    CortexNetworkContext ctx = new CortexNetworkContext();

    Dictionary<string, object> encoderSettings = getDefaultSettings();

    DateTimeEncoder encoder = new DateTimeEncoder(encoderSettings);

    var result = encoder.EncodeDateOnly(input);

    foreach (var item in result)
    {
        Debug.Write(item + ",");
    }
    Assert.IsTrue(result.SequenceEqual(expectedOutput));
}
```

Here, I have implemented the time encoder only and I had performed multiple unit tests for time encoder as well:

```
/// <summary>
/// Implementation of only Time encoder.
/// </summary>
public int[] EncodeTimeOnly(object inputData)
{
    DateTime inputTime = DateTime.Parse((String)inputData);
    int[] hourArray = encoding(inputTime.Hour, (int)(W * 24 / Radius));
    int[] minuteArray = encoding(inputTime.Minute, (int)(W * 60 / Radius));
    int[] secondArray = encoding(inputTime.Second, (int)(W * 60 / Radius));

    int[] timeArray = new int[hourArray.Length + minuteArray.Length + secondArray.Length];
    foreach (var item in timeArray)
    {
        timeArray[item] = 0;
    }
    hourArray.CopyTo(timeArray, 0);
    minuteArray.CopyTo(timeArray, hourArray.Length);
    secondArray.CopyTo(timeArray, hourArray.Length + minuteArray.Length);
    return timeArray;
}
```

For the compilation of unit test part of time encoder only following is the code:

```
/// <summary>
/// performing number of unit tests to verify the time encoder only.
/// </summary>
/// <param name="input"></param>
/// <param name="expectedOutput"></param>
[TestMethod]
public void EncodeTimeOnlyTest(Object input, int[] expectedOutput)
{
    CortexNetworkContext ctx = new CortexNetworkContext();

    Dictionary<string, object> encoderSettings = getDefaultSettings();

    DateTimeEncoder encoder = new DateTimeEncoder(encoderSettings);

    var result = encoder.EncodeTimeOnly(input);

    foreach (var item in result)
    {
        Debug.Write(item + ",");
    }
    Assert.IsTrue(result.SequenceEqual(expectedOutput));
}
```

For the unit tests of the encoder, following are the codes:

1. *Demonstrates how to create an encoder by explicate invoke of initialization:*

```
/// <summary>
/// Demonstratses how to create an encoder by explicite invoke of initialization.
/// </summary>
[TestMethod]

public void InitTest1()
{
    Dictionary<string, object> encoderSettings = getDefaultSettings();

    // We change here value of Name property.
    encoderSettings["Name"] = "hello";

    // We add here new property.
    encoderSettings.Add("TestProp1", "hello");

    var encoder = new DateTimeEncoder();

    // Settings can also be passed by invoking Initialize(sett)
    encoder.Initialize(encoderSettings);

    // Property can also be set this way.
    encoder["abc"] = "1";

    Assert.IsTrue((string)encoder["TestProp1"] == "hello");

    Assert.IsTrue((string)encoder["Name"] == "hello");

    Assert.IsTrue((string)encoder["abc"] == "1");
}
```

2. *Initializes encoder and sets mandatory properties:*

```
/// <summary>
/// Initializes encoder and sets mandatory properties.
/// </summary>
[TestMethod]
public void InitTest2()
{
    CortexNetworkContext ctx = new CortexNetworkContext();

    Dictionary<string, object> encoderSettings = getDefaultSettings();

    var encoder = ctx.CreateEncoder(typeof(DateTimeEncoder).FullName, encoderSettings);

    foreach (var item in encoderSettings)
    {
        Assert.IsTrue(item.Value == encoder[item.Key]);
    }
}
```

3. *Demonstrates how to create an encoder and how to set encoder properties by using of context:*

```
/// <summary>
/// Demonstratses how to create an encoder and how to set encoder properties by using of context.
/// </summary>
[TestMethod]
public void InitTest3()
{
    CortexNetworkContext ctx = new CortexNetworkContext();

    // Gets set of default properties, which more or less every encoder requires.
    Dictionary<string, object> encoderSettings = getDefaultSettings();

    // We change here value of Name property.
    encoderSettings["Name"] = "hello";

    // We add here new property.
    encoderSettings.Add("TestProp1", "hello");

    var encoder = ctx.CreateEncoder(typeof(DateTimeEncoder).FullName, encoderSettings);

    // Property can also be set this way .
    encoder["abc"] = "1";

    Assert.IsTrue((string)encoder["TestProp1"] == "hello");

    Assert.IsTrue((string)encoder["Name"] == "hello");

    Assert.IsTrue((string)encoder["abc"] == "1");
}
```

4. *Demonstrates how to create an encoder by explicateinvoke of initialization:*

```
/// <summary>
/// Demonstratses how to create an encoder by explicite invoke of initialization.
/// </summary>
[TestMethod]
public void InitTest4()
{
    Dictionary<string, object> encoderSettings = getDefaultSettings();

    // We change here value of Name property.
    encoderSettings["Name"] = "hello";

    // We add here new property.
    encoderSettings.Add("TestProp1", "hello");

    var encoder = new DateTimeEncoder();

    // Settings can also be passed by invoking Initialize(sett)
    encoder.Initialize(encoderSettings);

    // Property can also be set this way.
    encoder["abc"] = "1";

    Assert.IsTrue((string)encoder["TestProp1"] == "hello");

    Assert.IsTrue((string)encoder["Name"] == "hello");

    Assert.IsTrue((string)encoder["abc"] == "1");
}
```

5. *Initialize all encoders*

```
/// <summary>
/// Initializes all encoders.
/// </summary>
[TestMethod]
public void InitializeAllEncodersTest()
{
    CortexNetworkContext ctx = new CortexNetworkContext();

    Assert.IsTrue(ctx.Encoders != null && ctx.Encoders.Count > 0);

    Dictionary<string, object> encoderSettings = getDefaultSettings();

    foreach (var item in ctx.Encoders)
    {
        var encoder = ctx.CreateEncoder(typeof(DateTimeEncoder).FullName, encoderSettings);

        foreach (var sett in encoderSettings)
        {
            Assert.IsTrue(sett.Value == encoder[sett.Key]);
        }
    }
}
```

For the datetime encoder, date encoder only and time encoder only, I am giving the values of width and the radius to the encoder by following code:

```
/// <summary>
/// Values for the radius and width will be passed here.
/// </summary>
/// <returns></returns>
private static Dictionary<string, object> getDefaultSettings()
{
    Dictionary<String, Object> encoderSettings = new Dictionary<string, object>();
    //encoderSettings.Add("N", 0);
    encoderSettings.Add("W", 1);
    //encoderSettings.Add("MinVal", (double)0);
    //encoderSettings.Add("Max2Val", (double)0);
    encoderSettings.Add("Radius", (double)1);
    //encoderSettings.Add("Resolution", (double)0);
    //encoderSettings.Add("Periodic", (bool)false);
    //encoderSettings.Add("ClipInput", (bool)true);
    return encoderSettings;
}
```

III. RESULTS

Basically, the DateTime encoder is implemented on by using above mentioned equation 1.1. As we had observed by varies the parameters of equation 1.1 one can write encoder for the specific desired date and time. Since, the code has been implemented in the DateTime library and unit test for the execution purpose. As it will take 2 parameters from user in shape of W and radius, then in encoding method the code follows the given instructions for correct execution.

Following are the implementation and their results from the unit tests:

1. *Case 1: (When W = 1 and Radius = 1)*

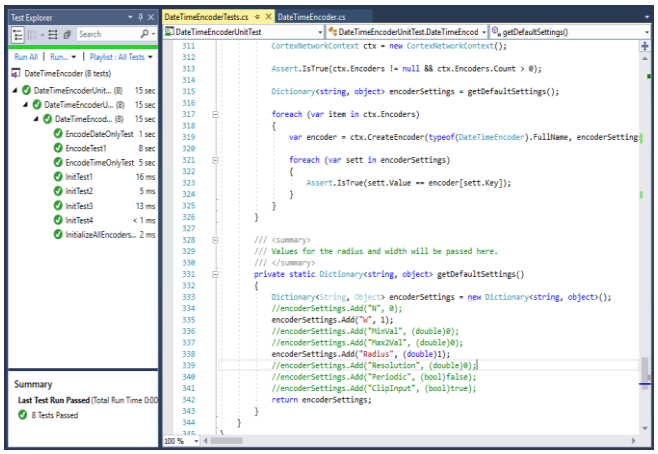

Fig. 3. Unit test when Width = 1 and Radius = 1

In the following unit test we had set width to 1 and the radius to 1 and we had performed unit test for different date and time slot i.e. "05/01/2011 20:55:06", "06/02/2012 21:56:07", "07/03/2013 22:57:08", "08/04/2014 23:58:09", and "08/04/2014 23:58:10". From the output of the unit test it is evident for width = 1 and radius = 1, the encoder is working correctly.

2. *Case 1: (When W = 2 and Radius = 1)*

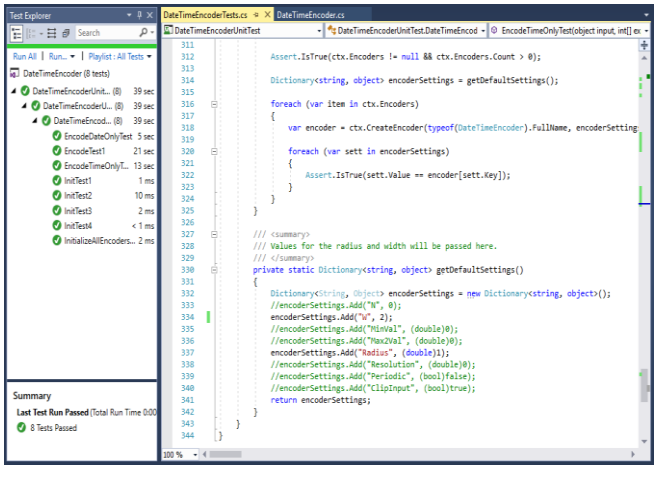

Fig. 4. Unit test when Width = 2 and Radius = 1

In the following unit test we had set width to 2 and the radius to 1 and we had performed unit test for different date and time slot i.e. "11/04/2010 14:55:01", "02/21/2019 13:51:43", "07/12/2009 01:38:23", "12/05/2002 21:09:35", "10/03/2005 15:11:56", "01/27/2012 19:01:54", and "03/24/2015 10:06:34". From the output of the unit test it is evident for width = 2 and radius = 1, the encoder is working correctly.

3. *Case 1: (When W = 3 and Radius = 3)*

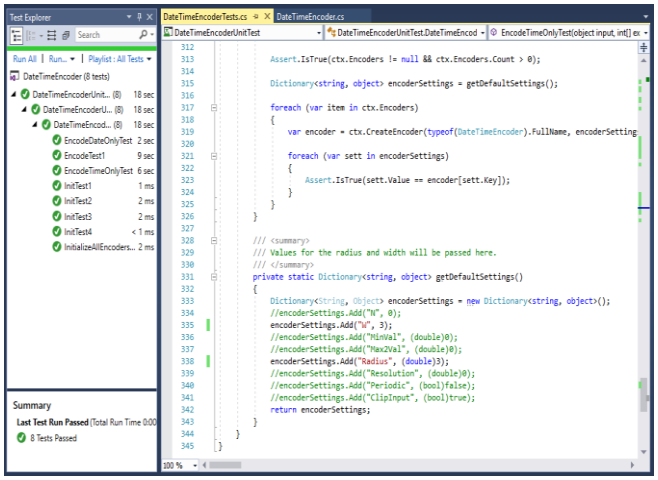

Fig. 5. Unit test when Width = 3 and Radius = 3

In the following unit test we had set width to 3 and the radius to 3 and we had performed unit test for different date and time slot i.e. "11/04/2010 14:55:00", "04/25/2013 08:10:58", "12/12/2012 12:12:12", "01/10/2022 21:13:43", "07/18/2016 20:04:15", and "03/13/2008 07:34:27". From the output of the unit test it is evident for width = 3 and radius = 3, the encoder is working correctly.

4. *Case 1: (When W = 6 and Radius = 6)*

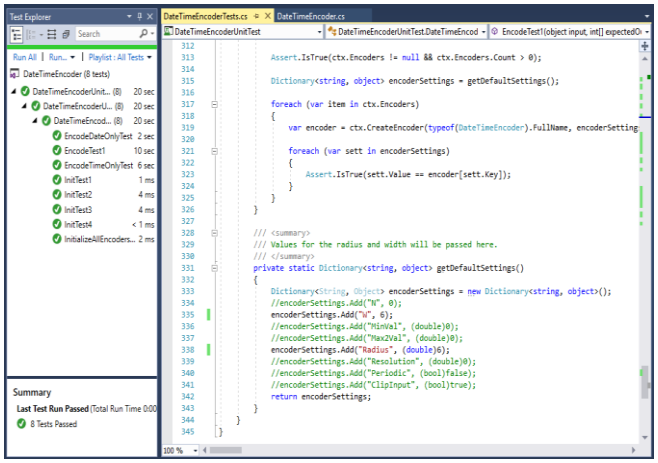

Fig. 6. Unit test when Width = 6 and Radius = 6

In the following unit test we had set width to 6 and the radius to 6 and we had performed unit test for different date and time slot i.e. "11/04/2010 14:55:00", "03/13/2008 07:34:27", "07/17/2002 22:46:50", "09/05/2001 18:06:59", "01/31/2014 05:19:33", "12/25/2030 19:29:04", and "05/01/2055 23:47:03". From the output of the unit test it is evident for width = 6 and radius = 6, the encoder is working correctly.

Finally, I had compiled all the unit tests individually using unit test method for the specific width and the radius of the encoder. Following is the output of 32 tests passed for this encoder:

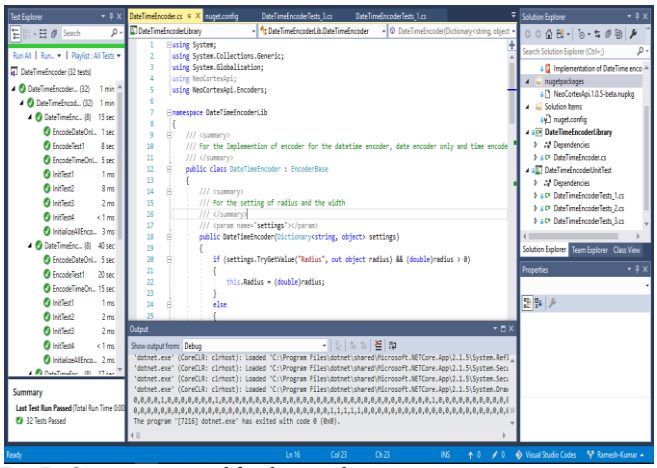

Fig. 7. 32 unit test passed for the encoder

IV. Discussion

1. *Conclusion*

Limitations: If the value of n in the encoder is some decimals number (e.g. 36.5, 40.75, etc.), then the encoder will not encode the data in correct order. For example,
If n = 50.5 and in this case, we can't encode the last available bit of the encoder. Therefore, under this condition the encoder is prone to error.

Explanation for the prone to error condition:
Since we have, *n = width * (range / radius)*, now assume we have to make an encoder of width = 3 and the radius = 4.

Let's consider the case for the specific month, we have month starting from January till December, since the range is 12 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, and 12, Therefore for the month *n = (3 * (12/4)) = 9*. From this we can conclude that we will have an encoder of 9 numbers, since n = 9 and there would not be any error for this case.

For the specific date, we have date which will start from the 1st to 31st, since the range is 31 i.e. 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, and 31. Therefore for the date *n = (3*(31/4)) =23.25.* From this we can conclude that we will have an encoder of 23.25 numbers, since n = 23.25 and assume we will have an encoder e.g. 00011100000000000000000*0* but this last bit can't be divided into subpart i.e. 0.25 as we must have this last bit in whole number for correct encoding.

For the specific year, we have year which will start from the e.g. 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, and 2009, since rest are the repetitions of two last bits, therefore, we can conclude for the year we have range of 10 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, and 09. Therefore for the year *n = (3*(10/4)) = 7.5*. Again, we can't encode 0.5 last bit here as well.

For the specific hour, we have hour which will start from the 00 to 23, since the range is 24 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, and 23. Therefore for the hour *n = (3*(24/4)) = 18.* Again, here the encoder will work correctly.

For the specific minute, we have minute which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the minute *n = (3*(60/4)) = 45*. Again, here the encoder will work correctly.

For the specific second, we have second which will start from the 00 to 59, since the range is 60 i.e. 00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, and 59. Therefore for the second *n = (3*(60/4)) = 45*. Again, here the encoder will work correctly.

Concatenation in this case would be, encoder = n = (month + date + year + hour + minute + second) = (9 + 23.25 + 7.5 + 18 + 45 + 45) = 147.75, this means that the final encoder would consist total of 147.75 bits, which will obviously not encode correctly. Since, at the output we have some error. Therefore, for non-decimal number the encoder will give accurate answers and for the non-decimal numbers there might be some error associate with the final answer.

- **Pros**
  - The DateTime encoder works for all values of width and radius, when n is some non-decimal number.
  - Since the rules of the calendar aren't that complex, especially when the encoding difference between Feb 28 and Jan 1 is negligible. Because sometimes it doesn't matter if Nov 23 & Nov 28 are encoded as "the end of a month". And the DateEncoder's semantics are configurable, so if you want to give more meaning to "day of week" than "time of day" it's easy. Therefore, one can implement real world ideas using encoders.

- **Cons**
  - The DateTime gives error when n is some decimal number.

2. *Future Challenges*

I. In future, one can work to reduce such errors to make it workable under all conditions of width and radius. It would be a serious challenge for someone to implement this encoder for decimal n numbers for the formula of (1).

II. For rogue behavior detection, it really depends on human activity. A human might go home over the weekend, so there would be no activity for over 48 hours, but then all during the week there could be constant activity during work hours. Therefore, one can work to include this time difference in the encoder, so that it can also detect static moment. Since in case of DateTime, there is no static thing.

III. Someone can also implement the decoder of this encoder

REFERENCES

[1] F. D. S Webber, "Semantic Folding Theory And its Application in Semantic Fingerprinting, 57 Artificial Intelligence; Computation and Language; Neurons and Cognition. Retrieved from http://arxiv.org/abs/1511.08855," 2015.

[2] Rhyolight, "Predictions on irregular time series data," discourse.numenta.org, 2017.

[3] Scott Purdy, "Encoding Data for HTM Systems," Numenta, Inc, Redwood City, California, USA,.

[4] Olshausen, "Sparse coding with an overcomplete basis set: A strategy employed by V1," Vision Research, 37:3311-3325, Field, D.J., 1997.

[5] Numenta, "Encoders - NuPIC 1.0.3 documentation," Sphinx 1.5.3 , 2017.

[6] Pentti Kanerva, "Sparse distributed memory (SDM)," NASA Ames Research Center, United States, 1988.

[7] Jeff Hawkins, "Hierarchical temporal memory (HTM)," Numenta, Huntington, New York, United States, 2004.

[8] J. & Ahmad, S Hawkins, "Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex. arXiv, 1511.00083," http://arxiv.org/abs/1511.00083 2015.

[9] "NeoCortexApi.1.0.5-beta," Frankfurt University of Apllied Sciences, Frankfurt, Germany, [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/se-dystsys-2018-2019-softwareengineering/tree/master/Source/HTM/nuget 2019.

[10] "HTM," Frankfurt University of Applied Sciences, Frankfurt, Germany, [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/se-dystsys-2018-2019-softwareengineering/tree/master/Source/HTM 2019.

[11] Matthew Winder, "Adding nuget package to project ," stackoverflow.com, To set a relative path for a file-based package source with NuGet? 2015.