Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Implement HTM Persistance

Nabeela Maham
line 2: email address

Kizito
line 2: email address

Muhammad Jasim
line 2: email address

Abstract—

The Aim of this project is to create a system which trains the system and uses the data from the previously applied experiments. For the next time, the system will use the previous data. The persistence is designed as implementation of a custom serializer / desterilizer. The serializer saves the instance of some HTM module to the stream and deserializer is responsible to create the instance from the stream.

Keywords— - SparseObjectMatrix<T>, AbstractMatric, AbstractFlatMatrix, AsbstractBinaryMatrix, SpatialPooler, TemporalMemory, Column, Synapse, Topology, DistalDendriteSegment, ProximalDendriteSegment, Segment, SegmentActivity, Pool, Connections, InMemoryDistributedDictionaryspatial Pooler,

I. INTRO (*HEADING 1*)

Hierarchical temporal memory (HTM) is a biologically constrained machine intelligence technology developed by Numenta. Originally described in the 2004 book. On Intelligence by Jeff Hawkins with Sandra Blakeslee, HTM is the distributed building and primarily used today for anomaly detection in streaming data. The technology is based on neuroscience and the physiology and interaction of pyramidal neurons in the neocortex of the mammalian (in particular, human) brain.

PERSISTING AN OBJECT USING C#

You can use serialization to persist an object's data between instances, which enables you to store values and retrieve them the next time that the object is instantiated.

Persist the Object using serialization.

In order to persist the values for the class, you must first mark the class with the **Serializable** attribute. Add the following code above the class definition:

```
[Serializable()]
```

The SerializableAttribute tells the compiler that everything in the class can be persisted to a file. Because the PropertyChanged event does not represent part of the object graph that should be stored, it should not be serialized. Doing so would serialize all objects that are attached to that event. You can add the NonSerializedAttribute to the field declaration for the PropertyChanged event handler.

```
[field: NonSerialized()]
public event System.ComponentModel.PropertyChangedEventHandler PropertyChanged;
```

Beginning with C# 7.3, you can attach attributes to the backing field of an auto-implemented property using the field target value. The following code adds a TimeLastLoaded property and marks it as not serializable:

```
[field:NonSerialized()]
public DateTime TimeLastLoaded { get; set; }
```

The next step is to add the serialization code to the LoanApp application. In order to serialize the class and write it to a file, you use the System.IO and System.Runtime.Serialization.Formatters.Binary namespaces. To avoid typing the fully qualified names, you can add references to the necessary namespaces as shown in the following code:

```
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
```

The next step is to add code to deserialize the object from the file when the object is created. Add a constant to the class for the serialized data's file name as shown in the following code:

```
const string FileName = @"../../../SavedLoan.bin";
```

Next, add the following code after the line that creates the TestLoan object:

```
if (File.Exists(FileName))
{
   Console.WriteLine("Reading saved file");
   Stream openFileStream = File.OpenRead(FileName);
   BinaryFormatter deserializer = new BinaryFormatter();
   TestLoan = (Loan)deserializer.Deserialize(openFileStream);
   TestLoan.TimeLastLoaded = DateTime.Now;
   openFileStream.Close();
}
```

You first must check that the file exists. If it exists, create a Stream class to read the binary file and a BinaryFormatter class to translate the file. You also need to convert from the stream type to the Loan object type.

Next you must add code to serialize the class to a file. Add the following code after the existing code in the Main method:

```
Stream SaveFileStream = File.Create(FileName);
BinaryFormatter serializer = new BinaryFormatter();
serializer.Serialize(SaveFileStream, TestLoan);
SaveFileStream.Close();
```

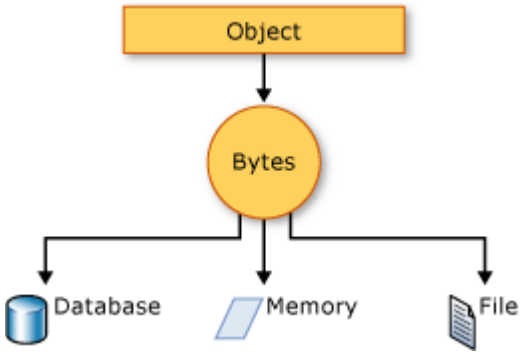
At this point, you can again build and run the application. The first time it runs, notice that the interest rates starts at 7.5, and then changes to 7.1. Close the application and then run it again. Now, the application prints the message that it has read the saved file, and the interest rate is 7.1 even before the code that changes it.

SERIALIZATION

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

A. How serialization works

This illustration shows the overall process of serialization:



The object is serialized to a stream that carries the data. The stream may also have information about the object's type, such as its version, culture, and assembly name. From that stream, the object can be stored in a database, a file, or memory.

1) Uses for serialization

Serialization allows the developer to save the state of an object and re-create it as needed, providing storage of objects as well as data exchange. Through serialization, a developer can perform actions such as:

- Sending the object to a remote application by using a web service
- Passing an object from one domain to another
- Passing an object through a firewall as a JSON or XML string
- Maintaining security or user-specific information across applications

B. JSON serialization

The System.Text.Json namespace contains classes for JavaScript Object Notation (JSON) serialization and deserialization. JSON is an open standard that is commonly used for sharing data across the web.

JSON serialization serializes the public properties of an object into a string, byte array, or stream that conforms to the RFC 8259 JSON specification. To control the way JsonSerializer serializes or deserializes an instance of the class:

- Use a JsonSerializerOptions object
- Apply attributes from the System.Text.Json.Serialization namespace to classes or properties
- Implement custom converters

C. Binary and XML serialization

The System.Runtime.Serialization namespace contains classes for binary and XML serialization and deserialization.

Binary serialization uses binary encoding to produce compact serialization for uses such as storage or socket-based network streams. In binary serialization, all members, even members that are read-only, are serialized, and performance is enhanced.

> WARNING: Binary serialization can be dangerous. For more information, see BinaryFormatter security guide.

XML serialization serializes the public fields and properties of an object, or the parameters and return values of methods, into an XML stream that conforms to a specific XML Schema definition language (XSD) document. XML serialization results in strongly typed classes with public properties and fields that are converted to XML. System.Xml.Serialization contains classes for serializing and deserializing XML. You apply attributes to classes and class members to control the way the XmlSerializer serializes or deserializes an instance of the class.

1) Making an object serializable

For binary or XML serialization, you need:

- The object to be serialized
- A stream to contain the serialized object
- A System.Runtime.Serialization.Formatter instance

Apply the SerializableAttribute attribute to a type to indicate that instances of the type can be serialized. An exception is thrown if you attempt to serialize but the type doesn't have the SerializableAttribute attribute.

To prevent a field from being serialized, apply the NonSerializedAttribute attribute. If a field of a serializable type contains a pointer, a handle, or some other data structure that is specific to a particular environment, and the field cannot be meaningfully reconstituted in a different environment, then you may want to make it nonserializable.

If a serialized class contains references to objects of other classes that are marked SerializableAttribute, those objects will also be serialized.

2) Basic and custom serialization

Binary and XML serialization can be performed in two ways, basic and custom.

Basic serialization uses .NET to automatically serialize the object. The only requirement is that the class has the SerializableAttribute attribute applied. The NonSerializedAttribute can be used to keep specific fields from being serialized.

When you use basic serialization, the versioning of objects may create problems. You would use custom serialization when versioning issues are important. Basic serialization is the easiest way to perform serialization, but it does not provide much control over the process.

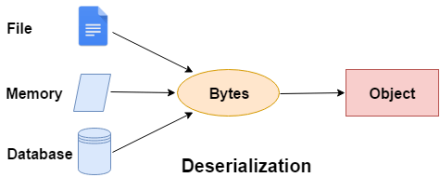
In custom serialization, you can specify exactly which objects will be serialized and how it will be done. The class must be marked SerializableAttribute and implement the ISerializable interface. If you want your object to be deserialized in a custom manner as well, use a custom constructor.

Serialization in HTM Persistance:

```
#region Serialization
2 references
public void Serialize(StreamWriter writer)
{
    HtmSerializer2 ser = new HtmSerializer2();

    ser.SerializeBegin(nameof(SegmentActivity), writer);

    ser.SerializeValue(this.ActiveSynapses, writer);
    ser.SerializeValue(this.PotentialSynapses, writer);

    ser.SerializeEnd(nameof(SegmentActivity), writer);
}
```

DESERIALIZATION

Deserialization is the reverse process of serialization. It means you can read the object from byte stream. Here, we are going to use **BinaryFormatter.Deserialize(stream)** method to deserialize the stream.

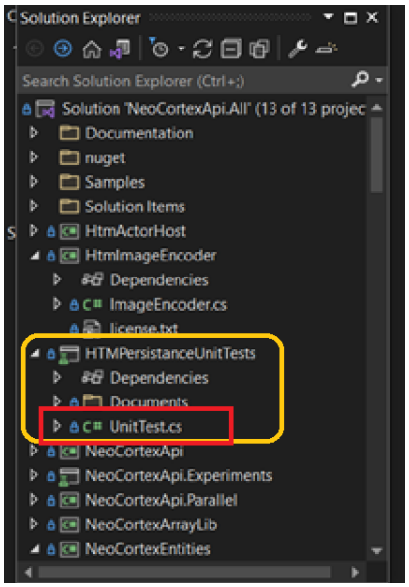


3) C# Deserialization Example

Derialization in HTM Persiatnce

```
2 references
public static SegmentActivity Deserialize(StreamReader sr)
{
    SegmentActivity segment = new SegmentActivity();
    HtmSerializer2 ser = new HtmSerializer2();
    while (sr.Peek() >= 0)
    {
        string data = sr.ReadLine();
        if (data == ser.LineDelimiter || data == ser.ReadBegin(nameof(SegmentActivity)) || data == ser.ReadEnd(nameof(SegmentActivity)))
        { }
        else
        {
            string[] str = data.Split(HtmSerializer2.ParameterDelimiter);
            for (int i = 0; i < str.Length; i++)
            {
                switch (i)
                {
                    case 0:
                        {

                            segment.ActiveSynapses = ser.ReadDictionaryIIValue(str[i]);
                            break;
                        }
                    case 1:
                        {
                            segment.PotentialSynapses = ser.ReadDictionaryIIValue(str[i]);
                            break;
                        }
                    default:
                        { break; }
                }
            }
        }
    }

    return segment;
}
#endregion
```

The project in the folder HTMPersistanceUnitTest



For Instance:
A person is perform an activity and uring that activity the user is unable to do anything else such chatting with family and friends, watching videos and listening sound. He stopped the activity and start doing the chatting with friends and performed his office work as well. Now, he restarted the activity. HTM persistence helps to resume the activity from where it was stopped. Serializer saves the data into TXT file and deserialize get that TXT file and give it to the system. And continue the work.
Serialization: The serializer saves the instance of some HTM module to the stream.
Deserialization: deserializer is responsible to create the instance from the stream.

We have done serialization and deserialization on SegmentActivity.cs, ProximalDEndrite.cs and DistelFDendrite.cs. We have Tested all the classes using UnitTest class.

In this paper test serialization and deserialization in the UnitTest.

The persistence is designed as implementation of a custom serializer/deserilizer. The serializer saves the instance of some HTM module to the stream and deserializer is responsible to create the instance from the stream.
Serialization class used for serialization and deserialization of primitive types.
HTM
Algorithm, biological
Traninng
Distributed building
Compute the traning process
Stop and on
Safe the resource
Importnace of serialization and deserialization
Htm Persistance
What we have done?
In the paper test serialization and deserialization

II. METHODS

Classes in this Project:

I have Tested three classes in this project. These classes are listed below.

1. SegmentActivity:

SegmentActivity.cs stores the calculus of a temporal cycle. It contains the index of segments with number of synapses with permanence higher than threshold. A
Dictionary, which holds the number of potential synapses of every segment.
Potential synspses are all established synapses between receptor cell and the segment's cell. Receprot cell was active cell in the previous cycle.
link to SegmentActivity (https://github.com/nabeelamaham/neocortexapi/blob/Nabeela-HTMPersistance/source/NeoCortexEntities/Entities/SegmentActivity.cs)

~~~csharp
Dictionary[segment index, number of active synapses].
Dictionary [segment index, number of potential synapses].
~~~

Serialization and deserialization have been applied on SegmentActivity.

2. ProximalDendrite:

It defines the eproximal dentritte segment. Note the segment is used during SP compute operation.
TM does not use this segment.
It uses the pool of synapses in the receptive field.

link to ProximalDendrite class : (https://github.com/nabeelamaham/neocortexapi/blob/Nabeela-HTMPersistance/source/NeoCortexEntities/Entities/ProximalDentrite.cs)

~~~csharp
public ProximalDendrite(int colIndx, double synapsePermConnected, int numInputs) : base(colIndx, synapsePermConnected, numInputs)
    {

    }
~~~

It creates and returns a newly created synapse with the specified source cell, permanence, and index.
This method is only called for Proximal Synapses. For ProximalDendrites, there are many synapses within a pool, and in that case, the index specifies the synapse's sequence order within the pool object, and may be referenced by that index.
It returns the instance of the new synapse.

~~~csharp
public Synapse CreateSynapse(int index, int inputIndex)
~~~

Indicies of Array of connected inputs defines RF(Potential Pool). It clear all the synapses from the segment. Sets the permanences for each linked Synapse specified by the indexes passed in which identify the input vector indexes associated with the permanences passed in are understood to be in "sparse" format and therefore require the int array identify their corresponding indexes.
This is the "sparse" version of this method.
Returns an array of synapse indexes as a dense binary array.
Returns an array of indexes of input neurons connected to this pool. It returns the indexes of connected input neurons.

~~~csharp
  public override void Serialize(StreamWriter writer)


      public static ProximalDendrite Deserialize(StreamReader sr)


~~~

3. DistalDendrite:

Implements a distal dendritic segment that is used for learning sequences. Segments are owned by Cells and in turn own Cells which are obversely connected to by a "source cell", which is the Cell that will activate a given Synapse owned by this Segment.

link to DistalDendriteDendrite (https://github.com/nabeelamaham/neocortexapi/blob/Nabeela-HTMPersistance/source/NeoCortexEntities/Entities/DistalDendrite.cs)

~~~csharp
  public class DistalDendrite : Segment,
IComparable<DistalDendrite>, IEquatable<DistalDendrite>
   {
~~~

Cell: The cell that owns (parent) the segment. the last iteration in which this segment was active.
The sequence number of the segment. Specifies the order of the segment of the Connections instance.

ParentCell:

The cell, which owns the segment.

flatIdx:

The flat index of the segment. If some segments are destroyed (synapses lost permanence)then the new segment will reuse the flat index. In contrast, the ordinal number will increas when new segments are created.
lastUsedIteration

ordinal:

The ordinal number of the segment. This number is incremented on each new segment.

If some segments are destroyed, this number is still increment.

synapsePermConnected

numInputs

~~~csharp
public DistalDendrite(Cell parentCell, int flatIdx, long lastUsedIteration, int ordinal, double synapsePermConnected, int numInputs) : base(flatIdx, synapsePermConnected, numInputs)
~~~

It compares this segment with the given one and Compares by index as well.

It Serialize method for DistalDendrite

~~~ csharp
internal void SerializeT(StreamWriter writer)
{
~~~

Then apply deserialization.

~~~csharp
public static DistalDendrite Deserialize(StreamReader sr)
{

~~~

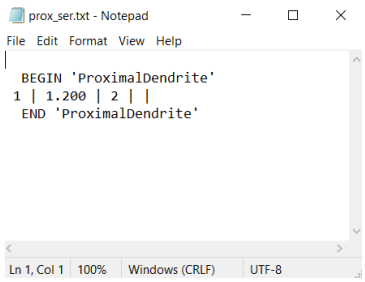
Unit Test Classes for the project.

the following code is applid for testing the classes(**SegmentActivity, DistelDendrite, pxorimalDendrite**)

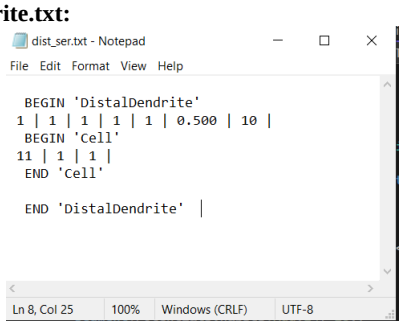
Here is the link to the unit Test class.

link to project (https://github.com/nabeelamaham/neocortexapi/blob/Nabeela-HTMPersistance/source/HTMPersistanceUnitTests/SerializeSegmentActivityTest.cs)

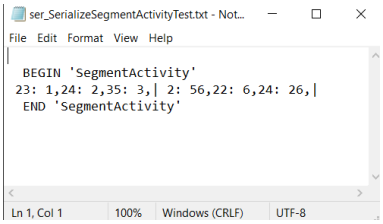
~~~csharp
public void TestSegmentActivitySErialization()
{
SegmentActivity segment = new SegmentActivity();
segment.ActiveSynapses = new Dictionary<int, int>();
segment.ActiveSynapses.Add(23, 1);
segment.PotentialSynapses = new Dictionary<int, int>();
segment.PotentialSynapses.Add(2, 56);
using (StreamWriter sw = new StreamWriter($"ser_{nameof(SerializeSegmentActivityTest)}.txt"))
{
segment.Serialize(sw);
}
using (StreamReader sr = new StreamReader($"ser_{nameof(SerializeSegmentActivityTest)}.txt"))
{
SegmentActivity segment1 = SegmentActivity.Deserialize(sr);
Assert.IsTrue(segment1.Equals(segment));
}
}


[TestClass]
public class DistalDendriteSerializationTest
{
[TestMethod]
public void TestDistalDendriteSerialization()
{
Cell c1 = new Cell(1, 1, 10, 1, NeoCortexEntities.NeuroVisualizer.CellActivity.ActiveCell);
DistalDendrite d1 = new DistalDendrite(c1, 1, 1, 1, 0.5, 10);
using (StreamWriter sw = new StreamWriter("dist_ser.txt"))
{
d1.Serialize(sw);
}
DistalDendrite d2;
using (StreamReader sr = new StreamReader("dist_ser.txt"))
{
d2 = DistalDendrite.Deserialize(sr);
}
var result = HtmSerializer2.IsEqual(d1, d2);
Assert.IsTrue(result);
}
}


[TestClass]
public class ProximalDendriteSerializationTest
{
[TestMethod]
public void TestProximalDendriteSerialization()
{
Cell c1 = new Cell(1, 1, 10, 1, NeoCortexEntities.NeuroVisualizer.CellActivity.ActiveCell);
ProximalDendrite p1 = new ProximalDendrite(1, 1.2, 2);
using (StreamWriter sw = new StreamWriter("prox_ser.txt"))
{
p1.Serialize(sw);
}
ProximalDendrite p2;
using (StreamReader sr = new StreamReader("prox_ser.txt"))
{
p2 = ProximalDendrite.Deserialize(sr);
}
var result = HtmSerializer2.IsEqual(p1, p2);
Assert.IsTrue(result);
```

**Serialization logic**

It is used for serialization and deserialization of primitive types. Such as Integer, Boolean, String, Array Int[], Double, Long. It work for non primitive type such as Synapses and cells.

Serializes the begin and end marker of the type.

**ProximalDendrite.txt:**
~~~



DistalDendrite.txt:



SegmentActivity.txt:

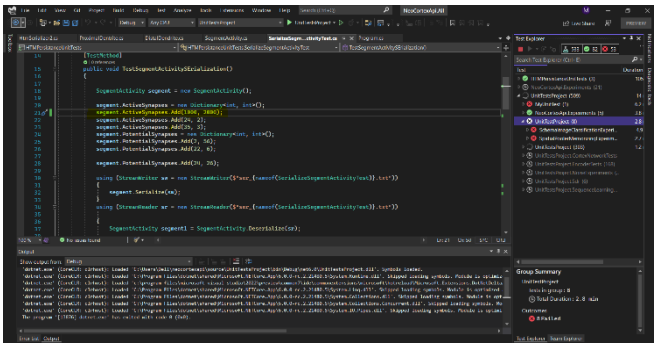


Htm persistance serialize the property of type Int. Read the property of type Int and return that integer. Deserializes from text file to DistalDendrite and return DistalDendrite. Serialize the property of type Double, String, Long and Bool. Read the property of type Double, String, Long, Bool and return these values. Serialize the array of type Double. Read the array of type Double and return double. Serialize the array of type Int. Read the array of type Int returns Int[]. Serialize and Deserialize the array of cells. Deserializes from text file to Cell and return cells. Serialize the dictionary with key:string and value:int. Read the dictionary with key:string and value:int and return Dictionary<String, int>. Serialize the List of DistalDendrite, Synapses and Integer and Dictionary<Segment, List<Synapses>>. Read the List of DistalDendrite and returns distal dendrite. Serialize the dictionary and Concurrentdictionary with key:int and value:Synapse and DistalDendrite. Read the dictionary with key:int and value:Synapse return Dictionary<int, Synapse>

III. Results

USED DIFFERENT VALUE AND SHOW THE RESUKLT

CONCLUSION OF YOUR I HAD USED DIFFERENT VALUES AND THE RESULT IS REACHED LIKE THIS:



WE ARE THREE MEMBERS IN THIS PROJECT. FAIZAN HAS NOT BEEN IMPLEMENTED HIS PART BECAUSE HE IS NOT PRESENT

IV. Discussion

Conclusion of your work should be precise and concise. How was the project, what is done, what is the result... There can be discussion on further work and direction.

Future scope

V. Ease of Use

VI. Prepare Your Paper Before Styling

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections A-D below for more information on proofreading, spelling and grammar.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

A. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

B. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive".
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: "Wb/m2" or "webers per square meter", not "webers/m2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".

may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a \square\square\square b \square\square\square\square\square \qquad \square\square\square$$

Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

C. Some Common Mistakes

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum $\square_0$, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is [7].

VII. Using the Template

After the text edit has been completed, the paper is ready for the template. Duplicate the template file by using the Save As command, and use the naming convention prescribed by your conference for the name of your paper. In this newly created file, highlight all of the contents and import your prepared text file. You are now ready to style your paper; use the scroll down window on the left of the MS Word Formatting toolbar.

A. Authors and Affiliations

The template is designed for, but not limited to, three authors. A minimum of one author is required for all report articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

1) For papers with more than three authors: Add author names horizontally, moving to a third row if needed for more than 8 authors.

2) For papers with less than three authors: To change the default, adjust the template as follows.

a) Selection: Highlight all author and affiliation lines.

b) Change number of columns: Select the Columns icon from the MS Word Standard toolbar and then select the correct number of columns from the selection palette.

c) Deletion: Delete the author and affiliation lines for the extra authors.

B. *Identify the Headings*

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1”, “Heading 2”, “Heading 3”, and “Heading 4” are prescribed.

C. *Figures and Tables*

For adding object other than text (tables, equations, graphs, figures, code…), there must be at least one cross reference to it. Figure 1 is an example

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

TABLE I. TABLE TYPE STYLES

Table Head	Table Column Head		
	Table column subhead	*Subhead*	*Subhead*
copy	More table copy[a]		

[a] Sample of a Table footnote. (*Table footnote*)

Figure 1 Example Figure Caption

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi TIFF or EPS file, with all fonts embedded) because, in an MSW document, this method is somewhat more stable than directly inserting a picture.

To have non-visible rules on your frame, use the MSWord “Format” pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.

D. *Code References:*

Referencing Code in your text should be avoided unless necessary. In such cases it can be inserted as a listing as shown in **Error! Reference source not found.**

Listing 1 Code Reference Example

```
Console.WriteLine(“Referencing code”, var);
  // using tab can be replaced with 4 spaces
```

Do not pass code as image. When referring to variable in **Error! Reference source not found.**, italics should be used for example *var*. Code flows and logic should be presented better as Graph or Diagram instead of words.
Code Block which is too big to put in the textbox can be reference as Listing 2.

Listing 2 Unit Test EncodeDateTimeTest

```
public void EncodeDateTimeTest(int w, double r, …)
  {
    …
    DateTimeEncoderExperimental encoder = new…
    var result = encoder.Encode(input);
    …
    Assert.IsTrue(result.SequenceEqual(expected…
    }
```


ACKNOWLEDGMENT *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

1. https://www.educba.com/deserialization-in-c-sharp/
2. https://www.guru99.com/c-sharp-serialization.html
3. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/
4. https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/walkthrough-persisting-an-object-in-visual-studio

This report template contains guidance text for composing and formatting technical reports. Please ensure that all template text is removed from your report prior to submission to the examination office. Failure to remove template text from your paper may result in your paper being degraded.