UNIVERSITY OF WATERLOO

Faculty of Engineering

Department of Electrical and Computer Engineering

Runtime Analysis of XML Serialization in VB.NET

The CUMIS Group Limited

Burlington, Ontario

Prepared by

Michael A. Soares

ID [removed]

userid masoares

2A Computer Engineering

21 September 2009

[address removed]

21 September 2009

Dr. Manoj Sachdev, chair
Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario
N2L 3G1

Dear Sir:

This report, entitled "Runtime Analysis of XML Serialization in VB.NET", was prepared as my 2A Work Report for The CUMIS Group Limited. This report is in fulfillment of the course WKRPT 200. The main purpose of this report is to quantitatively analyse the different methods in which data, whether contained in a database or another digital storage medium, can be serialized into XML (i.e., be manipulated into a form in which another software application can further manipulate said data) and to offer several recommendations which may help in developing future software applications which may use said serialization.

I was employed as a Developer and Technical Analyst working with a group of developers and project leaders managed by Paul Koski; my role was overseen by Emily Beavis. My primary responsibilities included modifying and upgrading existing VB.NET applications, developing web applications using XML, ASP.NET and VB.NET and working with SQL databases.

I would like to thank Mrs. Emily Beavis for providing me with encouragement, valuable ideas and information over the work term which helped me in finalizing this report. I hereby confirm that I have received no further help other than what is mentioned above in writing this report. I also confirm this report has not been previously submitted for academic credit at this or any other academic institution.

Sincerely,

Michael A. Soares
ID [removed]

Contributions

For the past four months, I was employed at The CUMIS Group Limited ("CUMIS") in Burlington, Ontario as a Developer and Technical Analyst. I was part of the Information Technology department which consists of approximately 15 to 20 members, eight of which I worked and collaborated with on a daily basis, dealing with software development and database manipulation.

The team that I worked with is responsible for designing, developing, modifying and upgrading both software and web applications (code-wise) in use by CUMIS employees and credit unions across Canada. It consists mostly of Developers and Technical Analysts, such as myself during my employment at CUMIS, along with several Database Administrators (DBAs). Software is designed, developed and modified to meet CUMIS' needs by working collaboratively with project leaders and members from other departments and other Information Technology personnel at CUMIS. All software code developed at CUMIS is run through a thorough code review process with on-site supervisors and is compiled and then tested by project leaders before being put into the production environment.

While at CUMIS, my primary responsibilities involved working with new and existing software code, but more specifically encompassed:

- Developing web applications using tools such as VB.NET, ASP.NET, XML and HTML,
- Working with web services to interface differently-coded applications,
- Upgrading already-existing code to be fully compliant with Microsoft's .NET 2.0 Framework and with CUMIS' coding standards,
- Debugging and properly commenting already-existing code,
- Manipulating database objects within SQL database servers, and
- Researching new technologies to evaluate their use within CUMIS' existing environment.

For the majority of the term, I was diligent in upgrading already-existing VB.NET and ASP.NET code to be fully compliant with the standards being used by other developers at CUMIS. At different times throughout the term, I researched different ways of minimizing the amount of time needed for code reviews as, for the most part, they were unnecessarily time consuming; the result of my findings involved creating several filters for a third-party file comparison tool which ignore both line and block comments within code, something the tool included with Microsoft's Visual Studio 2005 is not capable of.

The research that was done on this topic was quite insufficient in scope to be suitable for a work term report. Towards the end of the term, however, Mrs. Emily Beavis, my supervisor, gave me the opportunity to start the initial development of a new web application that retrieves data from a SQL database and converts or serializes said data into Extensible Markup Language (XML) format, which can later be processed by another application's web service. However, after leaving CUMIS, I began to have some doubts in the way that I began to code the web application (VB.NET structures were used), most likely affecting runtime (the length needed to run the application) which may become quite lengthy by the end of the web application's development. Thus, I decided to challenge myself and took it upon myself to analyse the methods in which I began to code the aforementioned web application and offer a series of possible solutions that could be implemented to rectify any unwanted performance issues.

This is the main relationship between this report, the knowledge I gained and the tasks I performed while working at CUMIS. The data collected and the analyses performed in this work term report are beneficial to me in many different ways, primarily because it has given me the opportunity to learn well beyond what I thought I would in my first job as a Developer and Technical Analyst. This project and this subsequent report have also provided me with the ability to analyse and evaluate code and quantitative data from a completely different perspective, including both the code and data discussed in this report.

In the broader scheme of things, my research on this report topic should prove to be beneficial for CUMIS and its developers. Since technology is changing at such a fast rate at the present time, CUMIS has to constantly keep up with the development of said technologies, but more specifically, software as a whole. In this report, I provide the Information Technology department and its developers with several recommendations on how to properly serialize data and how to improve upon the performance of any application utilizing said XML serialization.

Executive Summary

The main purpose and scope of this report is to qualitatively and quantitatively analyse using both VB.NET classes and structures to serialize data into proper XML, which can be later be manipulated by a web service. This report will suggest to CUMIS ways of modifying a current web service client which may favour runtime, all while maintaining functionality. I have identified several recommendations in this report that will reduce the runtime needed for data to be serialized and which will, overall, allow CUMIS to import data into its new web application in as little time as possible.

The major points in this report are that each of the major sections in this report identify and summarize the use of XML serialization and some of the factors contributing to serialization runtime. The first section describes the application that has been initially developed and the problem being analysed. The second section analyses XML serialization and the different methods of serializing data (i.e., by using classes or structures). The third section quantifies the methods used to serialize the data, as well as several modified methods, analyses the resultant data, and provides justifications for the resultant runtimes. The final sections provide conclusions and recommendations based on the analyses in the preceding sections.

The major conclusion of this report will confirm that structures are slightly easier to implement and use when dealing with XML serialization due to only needing to declare a single instance of the main structure as opposed to multiple ones. In addition, it will also show that the average runtimes, when serializing the same number of elements and levels of nested elements, for data serialized using both classes and structures are, for the most part and up to a certain degree of accuracy, identical. Lastly, this report will also confirm that increasing the number of levels of nested elements needing to be serialized can have an effect on runtime in a production environment, such as CUMIS', by approximately 3.5% per new level.

Major recommendations in this report are also identified in that CUMIS' should use structures wherever possible when working with XML serialization. CUMIS' developers should also limit the number of levels of nested elements being used when serializing data to keep runtime to a minimum. If data needs to be split up into separate elements, developers should ensure that said data is split up into elements within the same level instead of being nested any further.

Table of Contents

List of Figures

List of Tables

1 Introduction

The CUMIS Group Limited ("CUMIS"), its principal companies, CUMIS Life Insurance Company and CUMIS General Insurance Company and its subsidiaries partner with credit unions across Canada to deliver both competitive insurance (e.g., home, auto, life, disability, etc.) and other financial and non-financial solutions [1]. Most of the financial tools and other software used by CUMIS employees and partnered credit unions are written and tested internally in order to offer said competitive, as well as unique, financial and non-financial solutions.

CUMIS is currently preparing to migrate some of its financial data from an older web application to one that is newer and better suited for doing business with its clients. Unfortunately, the data migration is not as simple as one might think. The Structured Query Language (SQL) database structures for both applications are not identical. In fact, none of the table or column names and data types are equivalent, thus directly copying over the old application's SQL database is not possible. One must either manually enter records into the new application or code together a web service and matching client which will map data from the old application's database to the new one.

In this section, the purpose and scope of the report are both set out and essential background information is presented on the topic.

1.1 XML Serialization & Web Service Client

In general, the purpose of Extensible Markup Language (XML) serialization in the .NET Framework is to convert objects created in one application into an open, standards-compliant language which can be easily transported to or consumed by any other application which accepts said compliant language as input for further manipulation [2]. For CUMIS' web service client, an example of which will be further discussed and analysed in this report, XML serialization is used to map data from the old finance web application's database into elements and attributes which can be later consumed and

further manipulated by its corresponding web service. In terms of the example that will be further discussed, however, generic data is simply mapped to a pre-formed XML structure in VB.NET and is then serialized into proper XML.

1.2 Purpose

Because the form or structure of the serialized data/corresponding XML can be coded together in more than one way, this report will focus on and analyse two such methods: using VB.NET classes and structures. The serialization of both entities, both using the same "dummy" or generic data, will be analysed in terms of written code and runtime; factors which may contribute to said length of time will also be identified. In addition, this report will suggest ways of modifying said code which may favour runtime, all while maintaining functionality.

1.3 Scope

This report will include qualitative and quantitative analysis of using both VB.NET classes and structures, serialized into proper XML, which can be consumed by the web service.

1.4 Outline

The sections in this report identify and summarize the use of both VB.NET classes and structures. This report also provides a qualitative and a quantitative analysis of using classes and structures as well as solutions on improving the runtime of using each entity all while maintaining the same functionality. A glossary has also been included for easy reference of technical terms used in this report. The first section analyses XML serialization and the different methods of serializing data (i.e., by using classes or structures). The second section quantifies the methods used to serialize the data, as well as several modified methods, analyses the resultant data, and provides justifications for

the resultant runtimes. Finally, conclusions and recommendations are outlined at the end of the report.

2 Working with XML Serialization

2.1 Introduction

As mentioned in 1.1, XML serialization is used primarily when data, or more specifically, objects within in an application, must be converted to a standard format for transport to another application or service; the resultant XML can also be stored temporarily in memory or permanently for future manipulation [2]. When said XML is deserialized by a secondary application, that is to say, when the XML is parsed and the original objects are reconstructed, the original data stored within the objects can be manipulated in the way the secondary application has been programmed to manipulate said data (e.g., the data can be run through a series of algorithms or they can be stored in a database). Both serialization and deserialization of objects can be done by creating an instance of the `XmlSerializer` class, included in the `System.Xml` assembly in VB.NET, followed by calling the `Serialize` and `Deserialize` methods, respectively; this report focuses primarily on the `Serialize` method.

Two of the simplest methods of defining the XML code's structure in VB.NET before serialization occurs are by creating a root class (as the root node of the XML structure) and multiple sub-classes or by creating a root structure and multiple sub-structures within whatever application is being developed. Serializing either classes or structures will result in the same final XML code, assuming the code to do so is properly written.

2.2 Defining and Serializing the XML Structure

The web service client that was developed for CUMIS uses structures in VB.NET in order to define the final structure of the serialized XML. Side-by-side with similar code written as classes used to hold some very simple generic data, however, one can tell that the code for the defined XML structure does not differ by very much compared to that developed with structures, apart from the obvious `Class` and `Structure` declarations. An example illustrating this has been provided in Figure 1; the code in said example has been structured to hold some data that would be present in a typical business memo,

whereas the structure defined in the web service client developed for CUMIS actually holds financial data. The same overall concept, however, is the same.

```
<XmlRoot(ElementName:="memo")> _
Public Class MemoClass
    <XmlElement(ElementName:="header")> _
    Public Header As HeaderClass

    <XmlElement(ElementName:="body")> _
    Public Body As String

    <XmlElement(ElementName:="signedby")> _
    Public SignedName As FirstLastClass
End Class

Public Class HeaderClass
    <XmlElement(ElementName:="to")> _
    Public ToName As FirstLastClass

    <XmlElement(ElementName:="from")> _
    Public FromName As FirstLastClass

    <XmlElement(ElementName:="date")> _
    Public DateMMDDYYYY As DateClass

    <XmlElement(ElementName:="subject")> _
    Public Subject As String
End Class

Public Class FirstLastClass
    <XmlElement(ElementName:="firstname")> _
    Public FirstName As String

    <XmlElement(ElementName:="lastname")> _
    Public LastName As String
End Class
```

```
<XmlRoot(ElementName:="memo")> _
Public Structure MemoStruct
    <XmlElement(ElementName:="header")> _
    Dim Header As HeaderStruct

    <XmlElement(ElementName:="body")> _
    Dim Body As String

    <XmlElement(ElementName:="signedby")> _
    Dim SignedName As FirstLastStruct
End Structure

Public Structure HeaderStruct
    <XmlElement(ElementName:="to")> _
    Dim ToName As FirstLastStruct

    <XmlElement(ElementName:="from")> _
    Dim FromName As FirstLastStruct

    <XmlElement(ElementName:="date")> _
    Dim DateMMDDYYYY As DateStruct

    <XmlElement(ElementName:="subject")> _
    Dim Subject As String
End Structure

Public Structure FirstLastStruct
    <XmlElement(ElementName:="firstname")> _
    Dim FirstName As String

    <XmlElement(ElementName:="lastname")> _
    Dim LastName As String
End Structure
```

Figure 1. Code for a typical XML structure as classes and structures in VB.NET.

As can be seen in Figure 1, there is very little that differs in both cases; both have one root node and the same amount of nodes and other nested elements. The member variables in the classes have been purposely declared as `Public` so they can be easily accessed and manipulated outside each of their respective classes. This avoids having to declare separate properties with `Get` and/or `Set` accessors/mutators. However, CUMIS may prefer to use said accessors/mutators if there is an absolute need to protect the member variables from being modified and/or accessed outside of the class or if the data needs to be manipulated within the class itself. As mentioned in 2.1, serializing each of the sets of code in Figure 1 using the respective serialization methods for classes and structures contained in the `CreateCase` class (see Appendix A) results in the generation of the exact same XML code, included in Figure 2 on the next page.

```
<?xml version="1.0"?>
<memo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <header>
                <to>
                        <firstname>John</firstname>
                        <lastname>Doe</lastname>
                </to>
                <from>
                        <firstname>Jane</firstname>
                        <lastname>Doe</lastname>
                </from>
                <date>
                        <month>12</month>
                        <day>31</day>
                        <year>9999</year>
                </date>
                <subject>Business</subject>
        </header>
        <body>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras purus nisi,
fringilla vitae pulvinar eget, malesuada vitae leo. Nullam eleifend quam ligula, ut
elementum nulla. Proin volutpat leo id ante suscipit sit amet imperdiet metus egestas.
Nullam turpis lectus, consectetur sit amet pharetra non, ullamcorper nec neque. In hac
habitasse platea dictumst. Proin dignissim orci sit amet nunc sollicitudin lacinia.
Sed bibendum tempor arcu vitae dapibus. Vestibulum nisi dolor, rhoncus vel aliquet ac,
porta in risus. Mauris sodales, lacus auctor porta adipiscing, magna sapien
sollicitudin erat, quis vulputate urna nisi et nulla. Ut et ipsum arcu. Nam ut quam
ipsum. Nunc a quam orci, eleifend vehicula velit. Phasellus malesuada, turpis
ullamcorper aliquet convallis, dui nisl lacinia nibh, non scelerisque nibh nibh quis
velit. Nam lectus enim, eleifend quis accumsan quis, lacinia eget eros. Donec
vestibulum leo at nunc tincidunt bibendum.</body>
        <signedname>
                <firstname>Jane</firstname>
                <lastname>Doe</lastname>
        </signedname>
</memo>
```

Figure 2. Generated XML code from the serialized VB.NET code in Figure 1.

Despite the same XML being generated above in Figure 2, there is slightly more of a difference in the code in Appendix A, used to serialize both `MemoClass` and `MemoStruct` in Figure 1 separately. More specifically, before the `Serializer` method is called to serialize the `MemoClass` class (the final XML's root node), multiple instances of each one of the other classes (depending on how many member variables are using those classes as types) will need to be declared in addition to the instance of `MemoClass`; only a single instance of the main structure `MemoStruct` needs to be declared. The latter is true because all structures in VB.NET have an implicit parameterless public constructor which initializes all member variables recursively, meaning if a member variable within a structure is declared as a another structure (e.g., `Header`), that other instance will be initialized as well, and so on and so forth [3].

3 Quantifying the VB.NET Code

3.1 Introduction

Before having quantified the code used to serialize both `MemoClass` and `MemoStruct` in Figure 1, it was initially believed that calling each of the methods used to serialize `MemoClass` and `MemoStruct`, `classSerialize` and `structSerialize` in the `CreateCase` class (see Appendix A), respectively, would result in a shorter runtime for `structSerialize`. This assumption was based on two facts:

1) In order to serialize `MemoClass`, instances of each of the other classes need to be declared, most likely adding on to the total runtime of `classSerialize`, and
2) Like structs in the C# programming language (also based on the .NET Framework), structures, unlike classes, do not require heap allocation; variables in structures contain the data, whereas a variable in a class contains a reference to said data, hence the need for the additional declarations and additional assignments mentioned in 3.1, also likely to increase runtime [3], [4].

3.2 Quantitative Runtime Analysis

As mentioned in 3.1, instances of each of the other classes need to be declared in order for `MemoClass` to be serialized, as well as a few additional assignments. Theoretically, this would mean that each additional declaration (five in total) and each additional data assignment (five in total) would be required to run in a total of $O(10)$ time (*order of 10* time), equivalent to simply $O(1)$. In other words, the difference in runtime of `classSerialize` versus that of `structSerialize` should not be significant.

The runtime of the `classSerialize` and `structSerialize` methods were measured separately by first creating two `DateTime` objects, storing the start and end times in said objects and then finding the elapsed time between the two by using a `TimeSpan` object and converting it into seconds. The result of this calculation was written to a console window and taken note of. The `Main()` method used has been included in Figure 3.

```
Sub Main()
    Static start_time As DateTime
    Static stop_time As DateTime
    Dim elapsed_time As TimeSpan

    Dim ms As New MemoryStream()
    Dim newCert As New CreateCase()

    'Start the timer
    start_time = Now

    'Serialize the data
    'Change to structSerialize for structures
    newCert.classSerialize(ms)

    'Stop the timer and calculate the difference
    stop_time = Now
    elapsed_time = stop_time.Subtract(start_time)

    'Close the memory stream
    ms.Flush()
    ms.Dispose()
    ms.Close()

    'Output the result
    Console.WriteLine(elapsed_time.TotalSeconds. _
        ToString("0.000000"))
End Sub
```

Figure 3. Method used to calculate the runtime of the serialization methods.

The method in Figure 3 was run exactly fourty times for each of the `classSerialize` and `structSerialize` methods, changing the number of XML elements and levels of nested XML elements generated through serialization every ten trials (i.e., moving data from one member variable and removing it from the other, as well as removing unneeded classes/structures). To understand what this means in terms of code, please refer to Appendix B.

After running the method in Figure 3 (as well as that modified to call `structSerialize`), the average runtimes in Table 1 were calculated using the complete set of data in Appendix C for each of the modified serialization methods listed in Appendix B.

Elements	**1**	**2**	**3**	**3**
Nested Elements	0	0	4	4
Further Nested Elements	0	0	0	7
Class Runtime (s)	0.221875	0.223438	0.237500	0.240625
Structure Runtime (s)	0.223438	0.225000	0.232813	0.246875

Table 1. Average runtimes of modified serialization methods in seconds.

For the most part, the average runtimes in Table 1 for both the `classSerialize` and the `structSerialize` methods are very similar, and are identical to one another up to two decimal places in every case. This similarity in runtime is more easily observed when the values are plotted on a graph, such as that in Figure 4.

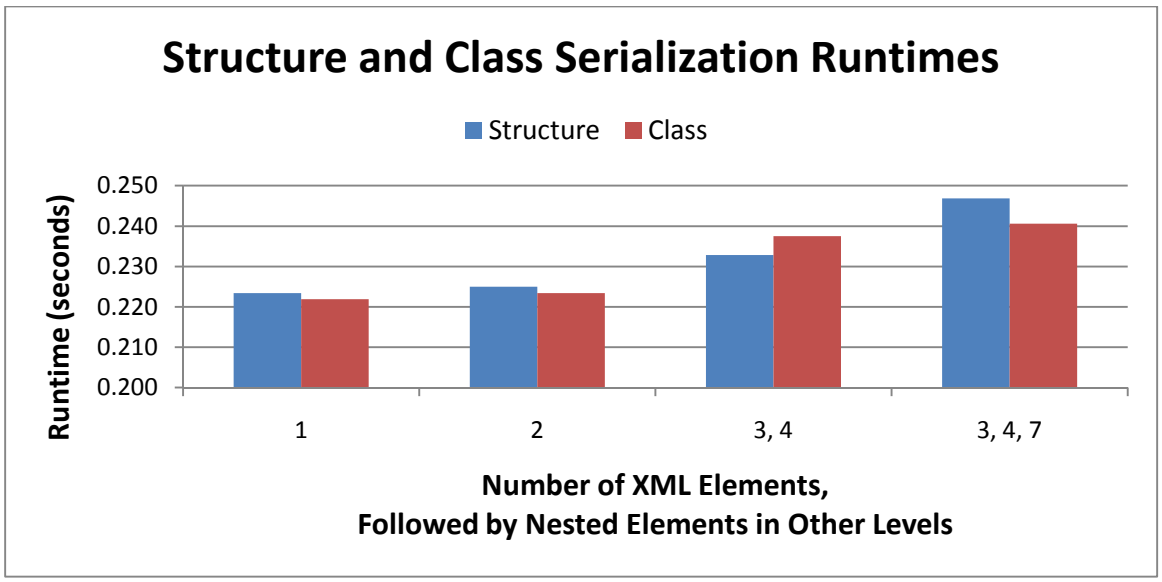

Figure 4. Average runtime values from Table 1 plotted on a bar graph.

From the data in Table 1, plotted in Figure 4, it is much easier to see that the largest difference in runtime occurs when the greatest number of nested elements is serialized. This behaviour was expected due to the `Serialize` method having to serialize objects nested within others, whereas with only one element (and only two as well), the data being serialized was contained within a single object, thus resulting in a shorter runtime for both classes and structures. The percent differences in runtime, calculated between classes and structures, have been included in Table 2.

Elements	**1**	**2**	**3**	**3**
Nested Elements	0	0	4	4
Further Nested Elements	0	0	0	7
Percent Difference (%)	0.7018	0.6969	1.9934	2.5641

Table 2. Percent differences in runtime between classes and structures.

3.3 Possible Solutions

With the largest difference in runtime between classes and structures in Table 2 being only approximately 2.5% (with classes serializing slightly faster in most cases), it should be up to the developer developing a serializing application, and perhaps even the project leader(s) as well, to decide on whether serialization should be implemented using classes or structures.

In a production environment such as CUMIS', where perhaps a thousand or so different records can be serialized, the 2.5% difference in the total serialization runtime would only represent approximately 6-7 seconds, which is quite minimal. With structures being slightly easier to implement, in other words, with structures only needing a single instantiation, it may be favourable for CUMIS to simply use them as opposed to classes. However, should CUMIS find the need to manipulate the data stored in said structures before being serialized or want to protect the data within said structures, then they may want to use classes in order to serialize their data instead since structures are not as customizable.

In addition to the above, CUMIS may want to limit the number of levels of nested elements being used when serializing data to keep runtime to a minimum. As can be easily observed in Table 1 and Figure 4, introducing a new level of nested elements into a class or a structure can easily increase serialization runtime by approximately 3.5% per new level introduced! Splitting data up into multiple elements in the same level has little to no noticeable effect on the serialization runtime of classes or structures and thus, CUMIS may want to consider this as well instead of creating multiple new levels of nested elements to store data.

These solutions will aid the Information Technology team in testing and rolling out the web service client and CUMIS' new web application promptly. Should CUMIS choose to leave the VB.NET structures that have already been implemented as-is, bugs within the web service client, if any, discovered by CUMIS employees, will be able to be

addressed quickly due to the simple nature of the code (written using structures, as mentioned in 2.2) that was already developed over this past work term.

4 Conclusions

From the analysis in the report body, it is concluded that compared to classes, structures are slightly easier to implement and use when dealing with XML serialization due to only needing to declare a single instance of the main structure as opposed to multiple ones. With classes specifically, this avoids having to assign other instances of classes to an object within the main class being serialized.

In addition, the average runtimes, when serializing the same data using the same number of elements and levels of nested elements, for both the `classSerialize` and the `structSerialize` methods are, for the most part and up to a certain degree of accuracy, identical. Using one or the other in a production environment such as CUMIS' will not bear very much of a noticeable effect when serializing thousands upon thousands of records of data.

Lastly, increasing the number of levels of nested elements needing to be serialized does have and can most certainly have an effect on runtime in a production environment, such as CUMIS', by approximately 3.5% per new level. Splitting data up into multiple elements in the same level, however, does not.

5 Recommendations

Based on the analysis and conclusions put forth in this report, it is recommended that CUMIS implement the following recommendations:

1) For ease of development when XML serialization is involved, CUMIS' developers should use structures. Classes should only be used instead in cases where data within a structure needs to be modified within the structure itself or if the data must be protected from either being accessed by or being modified by another class.
2) CUMIS' developers should limit the number of levels of nested elements being used when serializing data to keep runtime to a minimum.
3) If data needs to be split up into separate elements, developers should ensure that said data is split up into elements within the same level instead of being nested any further.

By implementing one or more of the above recommendations, development of the web service client will be completed in as little time as possible. CUMIS will also benefit in testing and rolling out the web service client needed to import data into their new web application promptly. It will also benefit them in keeping the overall length of time needed to actually import data into their new web application to a minimum.

The implementation of these recommendations will allow CUMIS employees to begin using its new web application as soon as possible. This will allow the Information Technology team to deal with bugs, discovered by CUMIS employees, in the web service application that may have been overlooked during initial development as quickly as possible so as to keep disruption of the application to a minimum.

Glossary

Accessor – A small method which is used to access objects from other parts of a program.

Assembly – Partially compiled code for use in development of applications.

Class – A construct used to create custom types within an application**.**

Heap – A tree-like data structure.

Method – A subroutine that is made up of programming statements used to perform an action on some data or to return data, usually associated with classes or objects.

Mutator – A small method which is used to change objects from other parts of a program.

.NET Framework – A software framework by Microsoft.

Object – An instance of a class.

Parameter – A piece of data passed into a program or a class on which is dependent on said data.

Property – In the context of this report, properties are implemented as a pair of accessor/mutator methods.

Runtime – The length of time in which an application runs from beginning to end.

SQL – Structured Query Language. A database language used for managing data in a database.

Structure – A data (value) type consisting of a number of other elements of many other types.

VB.NET – Visual Basic .NET. An object-oriented programming language based off of Visual Basic by Microsoft.

XML – Extensible Markup Language. A programming language used for encoding data.

References

[1] The CUMIS Group Limited, “About CUMIS,” 2009. [Online]. Available: http://www.cumis.com/cumis/freeFormDetail/0,2024,6190,00.html. [Accessed: Aug. 24, 2009].

[2] Microsoft Corporation, “XML Serialization in the .NET Framework,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms950721.aspx. [Accessed: Aug. 24, 2009].

[3] Microsoft Corporation, “Structures and Classes,” 2009. [Online]. Available: http://msdn.microsoft.com/en-us/library/2hkbth2a%28VS.71%29.aspx. [Accessed: Aug. 24, 2009].

[4] A. Hejlsberg, S. Wiltamuth, and P. Golde, *The C# Programming Language*. 2nd ed. Boston: Addison-Wesley, 2006, pp. 355

Appendix A – Serializer Method Differences

Class Serialization – Main differences (compared to that of a structure) have been highlighted in yellow.

```
Public Sub classSerialize(ByRef tempMemory As MemoryStream)
    'Declare instances of each class
    Dim businessMemo As New MemoClass
    Dim businessMemoHeader As New HeaderClass
    Dim businessMemoHeaderTo As New FirstLastClass
    Dim businessMemoHeaderFrom As New FirstLastClass
    Dim businessMemoHeaderDate As New DateClass
    Dim businessMemoSigned As New FirstLastClass

    'Declare an instance of XmlSerializer
    Dim XMLfrm As XmlSerializer = New _
      XmlSerializer(businessMemo.GetType())

    'Data assignments
    businessMemoHeaderTo.FirstName = "John"
    businessMemoHeaderTo.LastName = "Doe"
    businessMemoHeaderFrom.FirstName = "Jane"
    businessMemoHeaderFrom.LastName = "Doe"
    businessMemoHeaderDate.Month = "12"
    businessMemoHeaderDate.Day = "31"
    businessMemoHeaderDate.Year = "9999"
    businessMemoHeader.Subject = "Business"
    businessMemoSigned.FirstName = "Jane"
    businessMemoSigned.LastName = "Doe"

    With businessMemoHeader
        .ToName = businessMemoHeaderTo
        .FromName = businessMemoHeaderFrom
        .DateMMDDYYYY = businessMemoHeaderDate
    End With

    With businessMemo
        .Header = businessMemoHeader
        .Body = "Lorem ipsum dolor sit amet..." 'Body clipped
        .SignedName = businessMemoSigned
    End With

    'Serialize data to the memory stream
    XMLfrm.Serialize(tempMemory, businessMemo)
    tempMemory.Flush()
    tempMemory.Position = 0
End Sub
```

Structure Serialization

```
Public Sub structSerialize(ByRef tempMemory As MemoryStream)
    'Declare instances of the main structure
    Dim businessMemo As New MemoStruct

    'Declare an instance of XmlSerializer
    Dim XMLfrm As XmlSerializer = New _
      XmlSerializer(businessMemo.GetType())

    'Data assignments
    With businessMemo
        .Header.ToName.FirstName = "John"
        .Header.ToName.LastName = "Doe"
        .Header.FromName.FirstName = "Jane"
        .Header.FromName.LastName = "Doe"
        .Header.DateMMDDYYYY.Month = "12"
        .Header.DateMMDDYYYY.Day = "31"
        .Header.DateMMDDYYYY.Year = "9999"
        .Header.Subject = "Business"

        .Body = "Lorem ipsum dolor sit amet..." 'Body clipped
        .SignedName.FirstName = "Jane"
        .SignedName.LastName = "Doe"
    End With

    'Serialize data to the memory stream
    XMLfrm.Serialize(tempMemory, businessMemo)
    tempMemory.Flush()
    tempMemory.Position = 0
End Sub
```

Appendix B – Modified Serializer Methods and Results

One XML Element: <body>

Class Serializer

```
<XmlRoot(ElementName:="memo")> _
Public Class MemoClass1
    <XmlElement(ElementName:="body")> _
    Public Body As String
End Class
```

```
Public Sub classSerialize(ByRef tempMemory As MemoryStream)
    'Declare instances of each class
    Dim businessMemo As New MemoClass

    'Declare an instance of XmlSerializer
    Dim XMLfrm As XmlSerializer = New _
      XmlSerializer(businessMemo.GetType())

    With businessMemo
        .Body = "TO: John Doe\nFROM: Jane Doe\nDATE: 12319999\nSUBJECT:
Business\nLorem ipsum dolor sit amet...\n\nJane Doe" 'Body clipped
    End With

    'Serialize data to the memory stream
    XMLfrm.Serialize(tempMemory, businessMemo)
    tempMemory.Flush()
    tempMemory.Position = 0
End Sub
```

Structure Serializer

```
<XmlRoot(ElementName:="memo")> _
Public Structure MemoStruct
    <XmlElement(ElementName:="body")> _
    Dim Body As String
End Structure
```

```
Public Sub classSerialize(ByRef tempMemory As MemoryStream)
    'Declare instances of each class
    Dim businessMemo As New MemoStruct

    'Declare an instance of XmlSerializer
    Dim XMLfrm As XmlSerializer = New _
      XmlSerializer(businessMemo.GetType())

    'Continued on next page...
```

```
    With businessMemo
        .Body = "TO: John Doe\nFROM: Jane Doe\nDATE: 12319999\nSUBJECT:
Business\nLorem ipsum dolor sit amet...\n\nJane Doe" 'Body clipped
    End With


'Serialize data to the memory stream
    XMLfrm.Serialize(tempMemory, businessMemo)
    tempMemory.Flush()
    tempMemory.Position = 0
End Sub
```

Serialized XML

```
<?xml version="1.0"?>
<memo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <body>
            TO: John Doe\nFROM: Jane Doe\nDATE: 12319999\nSUBJECT:
Business\nLorem ipsum dolor sit amet, consectetur adipiscing elit. Cras
purus nisi, fringilla vitae pulvinar eget, malesuada vitae leo. Nullam
eleifend quam ligula, ut elementum nulla. Proin volutpat leo id ante
suscipit sit amet imperdiet metus egestas. Nullam turpis lectus,
consectetur sit amet pharetra non, ullamcorper nec neque. In hac
habitasse platea dictumst. Proin dignissim orci sit amet nunc
sollicitudin lacinia. Sed bibendum tempor arcu vitae dapibus.
Vestibulum nisi dolor, rhoncus vel aliquet ac, porta in risus. Mauris
sodales, lacus auctor porta adipiscing, magna sapien sollicitudin erat,
quis vulputate urna nisi et nulla. Ut et ipsum arcu. Nam ut quam ipsum.
Nunc a quam orci, eleifend vehicula velit. Phasellus malesuada, turpis
ullamcorper aliquet convallis, dui nisl lacinia nibh, non scelerisque
nibh nibh quis velit. Nam lectus enim, eleifend quis accumsan quis,
lacinia eget eros. Donec vestibulum leo at nunc tincidunt
bibendum.\n\nJane Doe
      </body>
</memo>
```

Two XML Elements: `<body>, <header>`

Class Serializer

```
<XmlRoot(ElementName:="memo")> _
Public Class MemoClass
    <XmlElement(ElementName:="header")> _
    Public Header As String

    <XmlElement(ElementName:="body")> _
    Public Body As String
End Class
```

```
Public Sub classSerialize(ByRef tempMemory As MemoryStream)
    'Declare instances of each class
    Dim businessMemo As New MemoClass

    'Declare an instance of XmlSerializer
    Dim XMLfrm As XmlSerializer = New _
      XmlSerializer(businessMemo.GetType())

    With businessMemo
        .Header = "TO: John Doe\nFROM: Jane Doe\nDATE:
12319999\nSUBJECT: Business"
        .Body = "Lorem ipsum dolor sit amet...\n\nJane Doe"
        'Body clipped
    End With

    'Serialize data to the memory stream
    XMLfrm.Serialize(tempMemory, businessMemo)
    tempMemory.Flush()
    tempMemory.Position = 0
End Sub
```

Structure Serializer

```
<XmlRoot(ElementName:="memo")> _
Public Structure MemoStruct
    <XmlElement(ElementName:="header")> _
    Dim Header As String

    <XmlElement(ElementName:="body")> _
    Dim Body As String
End Structure
```

```
Public Sub classSerialize(ByRef tempMemory As MemoryStream)
    'Declare instances of each class
    Dim businessMemo As New MemoStruct

    'Continued on next page...


    'Declare an instance of XmlSerializer
    Dim XMLfrm As XmlSerializer = New _
      XmlSerializer(businessMemo.GetType())

    With businessMemo
        .Header = "TO: John Doe\nFROM: Jane Doe\nDATE:
12319999\nSUBJECT: Business"
        .Body = "Lorem ipsum dolor sit amet...\n\nJane Doe"
        'Body clipped
    End With

    'Continued on next page...
```

```
    'Serialize data to the memory stream
    XMLfrm.Serialize(tempMemory, businessMemo)
    tempMemory.Flush()
    tempMemory.Position = 0
End Sub
```

Serialized XML

```
<?xml version="1.0"?>
<memo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <header>
            TO: John Doe\nFROM: Jane Doe\nDATE: 12319999\nSUBJECT:
Business
      </header>
      <body>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Cras purus nisi, fringilla vitae pulvinar eget, malesuada vitae leo.
Nullam eleifend quam ligula, ut elementum nulla. Proin volutpat leo id
ante suscipit sit amet imperdiet metus egestas. Nullam turpis lectus,
consectetur sit amet pharetra non, ullamcorper nec neque. In hac
habitasse platea dictumst. Proin dignissim orci sit amet nunc
sollicitudin lacinia. Sed bibendum tempor arcu vitae dapibus.
Vestibulum nisi dolor, rhoncus vel aliquet ac, porta in risus. Mauris
sodales, lacus auctor porta adipiscing, magna sapien sollicitudin erat,
quis vulputate urna nisi et nulla. Ut et ipsum arcu. Nam ut quam ipsum.
Nunc a quam orci, eleifend vehicula velit. Phasellus malesuada, turpis
ullamcorper aliquet convallis, dui nisl lacinia nibh, non scelerisque
nibh nibh quis velit. Nam lectus enim, eleifend quis accumsan quis,
lacinia eget eros. Donec vestibulum leo at nunc tincidunt
bibendum.\n\nJane Doe
      </body>
</memo>
```

Three XML Elements: <body>, <header>, <signedname>
Four Nested XML Elements: <to>, <from>, <date>, <subject>

Class Serializer

```
<XmlRoot(ElementName:="memo")> _
Public Class MemoClass
    <XmlElement(ElementName:="header")> _
    Public Header As HeaderClass

    <XmlElement(ElementName:="body")> _
    Public Body As String

    <XmlElement(ElementName:="signedname")> _
    Public SignedName As String
End Class

'Continued on next page...
```

```
Public Class HeaderClass
    <XmlElement(ElementName:="to")> _
    Public ToName As String

    <XmlElement(ElementName:="from")> _
    Public FromName As String

    <XmlElement(ElementName:="date")> _
    Public DateMMDDYYYY As String

    <XmlElement(ElementName:="subject")> _
    Public Subject As String
End Class
```

```
Public Sub classSerialize(ByRef tempMemory As MemoryStream)
    'Declare instances of each class
    Dim businessMemo As New MemoClass
    Dim businessMemoHeader As New HeaderClass

    'Declare an instance of XmlSerializer
    Dim XMLfrm As XmlSerializer = New _
      XmlSerializer(businessMemo.GetType())

    With businessMemoHeader
        .ToName = "John Doe"
        .FromName = "Jane Doe"
        .DateMMDDYYYY = "12319999"
        .Subject = "Business"
    End With

    With businessMemo
        .Header = businessMemoHeader
        .Body = "Lorem ipsum dolor sit amet..."
        .SignedName = "Jane Doe"
    End With

    'Continued on next page...
    'Serialize data to the memory stream
    XMLfrm.Serialize(tempMemory, businessMemo)
    tempMemory.Flush()
    tempMemory.Position = 0
End Sub
```

Structure Serializer

```
<XmlRoot(ElementName:="memo")> _
Public Structure MemoStruct
    <XmlElement(ElementName:="header")> _
    Dim Header As HeaderStruct

    <XmlElement(ElementName:="body")> _
    Dim Body As String

    'Continued on next page...
```

```
    <XmlElement(ElementName:="signedname")> _
    Dim SignedName As String
End Structure

Public Structure HeaderStruct
    <XmlElement(ElementName:="to")> _
    Dim ToName As String

    'Continued on next page...

    <XmlElement(ElementName:="from")> _
    Dim FromName As String

    <XmlElement(ElementName:="date")> _
    Dim DateMMDDYYYY As String

    <XmlElement(ElementName:="subject")> _
    Dim Subject As String
End Structure
```

```
Public Sub classSerialize(ByRef tempMemory As MemoryStream)
    'Declare instances of each class
    Dim businessMemo As New MemoStruct

    'Declare an instance of XmlSerializer
    Dim XMLfrm As XmlSerializer = New _
      XmlSerializer(businessMemo.GetType())

    With businessMemo
        .Header.ToName = "John Doe"
        .Header.FromName = "Jane Doe"
        .Header.DateMMDDYYYY = "12319999"
        .Header.Subject = "Business"
        .Body = "Lorem ipsum dolor sit amet..."
        .SignedName = "Jane Doe"
    End With

    'Serialize data to the memory stream
    XMLfrm.Serialize(tempMemory, businessMemo)
    tempMemory.Flush()
    tempMemory.Position = 0
End Sub
```

Serialized XML

```
<?xml version="1.0"?>
<memo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <header>
            <to>John Doe</to>
            <from>Jane Doe</from>
            <date>12319999</date>
            <subject>Business</subject>
      </header>
      <body>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Cras purus nisi, fringilla vitae pulvinar eget, malesuada vitae leo.
Nullam eleifend quam ligula, ut elementum nulla. Proin volutpat leo id
ante suscipit sit amet imperdiet metus egestas. Nullam turpis lectus,
consectetur sit amet pharetra non, ullamcorper nec neque. In hac
habitasse platea dictumst. Proin dignissim orci sit amet nunc
sollicitudin lacinia. Sed bibendum tempor arcu vitae dapibus.
Vestibulum nisi dolor, rhoncus vel aliquet ac, porta in risus. Mauris
sodales, lacus auctor porta adipiscing, magna sapien sollicitudin erat,
quis vulputate urna nisi et nulla. Ut et ipsum arcu. Nam ut quam ipsum.
Nunc a quam orci, eleifend vehicula velit. Phasellus malesuada, turpis
ullamcorper aliquet convallis, dui nisl lacinia nibh, non scelerisque
nibh nibh quis velit. Nam lectus enim, eleifend quis accumsan quis,
lacinia eget eros. Donec vestibulum leo at nunc tincidunt bibendum.
      </body>
      <signedname>Jane Doe</signedname>
</memo>
```

Three XML Elements:
<body>, <header>, <signedname>

Four Nested XML Elements:
<to>, <from>, <date>, <subject>, <firstname>, <lastname>

Seven Further Nested XML Elements:
<firstname> (x 2), <lastname> (x 2), <month>, <day>, <year>

See Figure 1, Figure 2, and Appendix A for code.

Appendix C – Runtime Test Data

Class Runtime Test
(All runtimes in seconds.)

Elements	**1**	**2**	**3**	**3**
Nested Elements	0	0	4	4
Further Nested Elements	0	0	0	7
Trial 1 Runtime (s)	0.234375	0.250000	0.265625	0.250000
Trial 2 Runtime (s)	0.218750	0.234375	0.250000	0.250000
Trial 3 Runtime (s)	0.218750	0.218750	0.234375	0.234375
Trial 4 Runtime (s)	0.218750	0.218750	0.234375	0.234375
Trial 5 Runtime (s)	0.218750	0.218750	0.281250	0.250000
Trial 6 Runtime (s)	0.234375	0.218750	0.203125	0.234375
Trial 7 Runtime (s)	0.218750	0.218750	0.234375	0.218750
Trial 8 Runtime (s)	0.218750	0.218750	0.218750	0.234375
Trial 9 Runtime (s)	0.234375	0.218750	0.218750	0.250000
Trial 10 Runtime (s)	0.218750	0.218750	0.234375	0.250000
Average Runtime (s)	0.221875	0.223438	0.237500	0.240625

Structure Runtime Test
(All runtimes in seconds.)

Elements	**1**	**2**	**3**	**3**
Nested Elements	0	0	4	4
Further Nested Elements	0	0	0	7
Trial 1 Runtime (s)	0.218750	0.234375	0.234375	0.296875
Trial 2 Runtime (s)	0.218750	0.234375	0.234375	0.281250
Trial 3 Runtime (s)	0.234375	0.218750	0.234375	0.234375
Trial 4 Runtime (s)	0.218750	0.218750	0.234375	0.234375
Trial 5 Runtime (s)	0.234375	0.234375	0.218750	0.250000
Trial 6 Runtime (s)	0.218750	0.218750	0.234375	0.218750
Trial 7 Runtime (s)	0.218750	0.218750	0.218750	0.234375
Trial 8 Runtime (s)	0.234375	0.234375	0.234375	0.250000
Trial 9 Runtime (s)	0.218750	0.218750	0.218750	0.234375
Trial 10 Runtime (s)	0.218750	0.218750	0.265625	0.234375
Average Runtime (s)	0.223438	0.225000	0.232813	0.246875