Assignment #1 – v2

Introduction

This practical assignment covers several topics taught in the first part of this course, such as: **functional programming**, LINQ, LINQ to XML, XPATH, REST/OData.

Overview

You are required to build a command-line program, **A1.exe**, which offers uniform access to several data sources:

1) **Files** containing data in the **XML** format (only schema-less documents without namespaces).
2) **Files** containing data in the **JSON** format.
3) **Web services** accessible via **REST/OData** protocols and delivering data in the **JSON** format.

This program must be developed (from scratch) as a single **C#** source file, named **A1.cs**, and must only use **.NET API** installed by **VS.2013** (which is available in the COMPSCI labs).

There is only one exception: you can also use the open source library **JSON.NET**, named **Newtonsoft.Json.dll** – this is given in the assignment test folder and offers fast conversions between the JSON and XML formats.

This assignment is strictly individual; please note that code sharing will be detected. Further information on academic misconduct can be found here https://www.auckland.ac.nz/en/about/learning-and-teaching/policies-guidelines-and-procedures/academic-integrity-info-for-staff/register-of-deliberate-academic-misconduct.html.

Input Specs

Program **A1.exe** expects 10 positional command-line parameters, logically grouped in **two** groups of 5, each group detailing all parameters is required to build a **sequence of XML elements**, further referred to as **seq_i** (where i = 1, 2):

A1.exe loc_1 fname-or-url_1 $xpath_1$ $kpath_1$ $spath_1$ loc_2 fname-or-url_2 $xpath_2$ $kpath_2$ $spath_2$

The following is a description of command-line parameters for i=1,2.

- **loc_i** : one of **/FILE-XML**, **/FILE-JSON**, **/URL-JSON**, indicating the data source location and format
  - any of the 3*3 combinations is possible
    - **/FILE-XML /FILE-XML**
    - **/URL-JSON /FILE-XML**
    - **/URL-JSON /URL-JSON**
    - **…**
- **fname-or-url_i** :
  - an absolute or relative **file name**, if **loc_i** is one of **/FILE-XML** or **/FILE-JSON**
  - an absolute **REST URL** to an **OData** service, if **loc_i** is **/URL-JSON**
- **$xpath_i$:** an XPATH query which returns **seq_i**
  - data in JSON format must first be converted to XML, using the "standard" JSON.NET conversion and wrapped into a an XML tag named **root. Use the api provided by Newtonsoft.Json.dll to do the conversion**
- **$kpath_i$** : an XPATH query, which returns a single subelement or attribute, relative to the root of each element of **seq_i**; its string value is used as a **joining key** in further operations
  - $kpath_1$ is **unique** for each element of **seq_1**
  - $kpath_2$ may return **duplicate** values when evaluated against seq_2
- **$spath_i$** : an XPATH query, which returns a single subelement or attribute, relative to the root of each element of **seq_i**; its string value is used as a **sorting key** in further operations. The results should be sorted in **ascending** order by the value of the sorting key

Note: All comparison operations should involve the **Ordinal string comparer (which defines culture independent case sensitive sorting strictly based on ordinal numbers of characters)**. Please look at System.StringComparer class. Furthermore, note that all values used here are strings, even if they look like numbers. For example, the following comparisons hold:

"A" < "a" (65 < 97)

"10" < "5" ("1" < "5", 49 < 53)

Output Specs

Each successful run of A1.exe creates five text files with predefined names and XML contents:

- **_LeftSeq.xml** : sequence **seq_1**, sorted by **$spath_1$**, and wrapped into an XML tag named **LeftSeq**
- **_RightSeq.xml** : sequence **seq_2**, sorted by **$spath_2$**, and wrapped into an XML tag named **RightSeq**
- **_InnerJoin.xml** : the inner join of **seq_1** and **seq_2**, wrapped into an XML tag named **InnerJoin**; each joined pair is grouped under an XML tag named **Join**
- **_GroupJoin.xml** : the group join of **seq_1** and **seq_2**, wrapped into an XML tag named **GroupJoin**; each joined pair is grouped under an XML tag named **Join**; and each paired right subsequence of size **n** is grouped under an XML tag named **Group** having an attribute **Count=n**
- **_LeftOuterJoin.xml** : the left outer join of **seq_1** and **seq_2**, wrapped into an XML tag named **LeftOuterJoin**; each joined pair is grouped under an XML tag named **Join**

For more details about the required output, contents and formats, see the **following sections** and the **Test folder**!

Additional Specs

The indicated **name spelling** is mandatory (both for input and output), including the indicated **case** (recall that XML is **case sensitive**).

Important note: the demo **OData** service **truncates long responses** (more specifically, responses which have more than 200 elements). However, in these cases, your program **must be able** to follow with new requests, until it gets all required data.

The XML sequences created by the program must **exactly** match the corresponding outputs created by the model solution, after discarding not significant white space.

All program **errors** must be caught and the error messages must be written to standard **Error**. Errors may appear as a result of incorrect command-line parameters or incorrect data.

Last, but not least, we provide a **Test folder** containing:

- typical **test files**
- typical **test batches**
- **expected outputs**
- a running compiled **model solution**

This test folder is an integral part of the assignment specs. Please consult it and experiment with it to clarify the required contents and formats.

Assessment

There are several ways to implement the required program. However, in this assignment, your solution will be mainly assessed according to the following criteria:

- **full exact XML correctness**
- **high-level functional code**
- **runtime** (compared with the given running model)

In particular, the following checks will also be used to assess the ”**high-level functional code**” criterion:

- no **class** definitions, except the main class which contains the entry point to the program (static void Main(string[] args)) and all other methods
- no **for, foreach** or **while** loops

For your information, the model program, which follows these guidelines, has about 65 semicolons (i.e. statements) and less than 100 lines (including a few blank lines, for readability).

Example

Consider the following two XML files, **MyCustomers.xml** and **MyOrders.xml**, which are also given in the test folder.

File **MyCustomers.xml**, where **Customer** elements (highlighted in yellow) are uniquely identified by their attribute **CustomerID**

```
<?xml version="1.0" encoding="utf-8"?>
<Customers>
  <Customer CustomerID="ggim">Georgy (Георгй)</Customer>
  <Customer CustomerID="rnic">Radu</Customer>
  <Customer CustomerID="sman">Mano</Customer>
  <Customer CustomerID="pdel">Patrice</Customer>
</Customers>
```

The XPATH query **Customers/Customer** returns a sequence consisting of all the **Customer** elements, in their file order (not sorted)

File **MyOrders.xml**, where **Order** elements (highlighted in yellow) are uniquely identified by their attribute **OrderID**

```
<?xml version="1.0" encoding="utf-8"?>
<Orders>
  <Order OrderID="4010" CID="pdel">cuda card</Order>
  <Order OrderID="1020" CID="rnic">optical mouse</Order>
  <Order OrderID="1010" CID="rnic">flash memory</Order>
  <Order OrderID="2030" CID="sman">digital camera</Order>
  <Order OrderID="2020" CID="sman">pocket pc</Order>
  <Order OrderID="2010" CID="sman">iphone</Order>
</Order >
```

The XPATH query **Orders/Order** returns a sequence consisting of all the **Order** elements, in their file order (not sorted)

Consider the following operation:

```
A1
   /FILE-XML MyCustomers.xml Customers/Customer @CustomerID @CustomerID
   /FILE-XML MyOrders.xml Orders/Order @CID @OrderID
```

Expected **_LeftSeq.xml** (note **Customer** elements are sorted on their CustomerID attribute)

```
<?xml version="1.0" encoding="utf-8"?>
<LeftSeq>
  <Customer CustomerID="ggim">Georgy (Георгй)</Customer>
  <Customer CustomerID="pdel">Patrice</Customer>
  <Customer CustomerID="rnic">Radu</Customer>
  <Customer CustomerID="sman">Mano</Customer>
</LeftSeq>
```

Expected **_RightSeq.xml** (note **Order** elements are sorted on their OrderID attribute)

```
<?xml version="1.0" encoding="utf-8"?>
<RightSeq>
  <Order OrderID="1010" CID="rnic">flash memory</Order>
  <Order OrderID ="1020" CID="rnic">optical mouse</Order>
  <Order OrderID ="2010" CID="sman">iphone</Order>
  <Order OrderID ="2020" CID="sman">pocket pc</Order>
  <Order OrderID ="2030" CID="sman">digital camera</Order>
  <Order OrderID ="4010" CID="pdel">cuda card</Order>
</RightSeq>
```

Expected **_InnerJoin.xml** (note that joined pairs are enclosed in an extra **Join** tag and are sorted as required by the specs)

```
<?xml version="1.0" encoding="utf-8"?>
<InnerJoin>
 <Join>
  <Customer CustomerID="pdel">Patrice</Customer>
  <Order OrderID="4010" CID="pdel">cuda card</Order>
 </Join>
 <Join>
  <Customer CustomerID="rnic">Radu</Customer>
  <Order OrderID="1010" CID="rnic">flash memory</Order>
 </Join>
 <Join>
  <Customer CustomerID="rnic">Radu</Customer>
  <Order OrderID="1020" CID="rnic">optical mouse</Order>
 </Join>
 <Join>
  <Customer CustomerID="sman">Mano</Customer>
  <Order OrderID="2010" CID="sman">iphone</Order>
 </Join>
 <Join>
  <Customer CustomerID="sman">Mano</Customer>
  <Order OrderID="2020" CID="sman">pocket pc</Order>
 </Join>
 <Join>
  <Customer CustomerID="sman">Mano</Customer>
  <Order OrderID="2030" CID="sman">digital camera</Order>
 </Join>
</InnerJoin>
```

Expected **_LeftOuterJoin.xml** (similar to **_InnerJoin.xml**, with one additional **Customer** element which does not have any associated **Order**)

```
<?xml version="1.0" encoding="utf-8"?>
<LeftOuterJoin>
 <Join>
  <Customer CustomerID="ggim">Georgy (Георгй)</Customer>
 </Join>
 <Join>
  <Customer CustomerID="pdel">Patrice</Customer>
  <Order OrderID="4010" CID="pdel">cuda card</Order>
 </Join>
 <Join>
  <Customer CustomerID="rnic">Radu</Customer>
  <Order OrderID="1010" CID="rnic">flash memory</Order>
 </Join>
 <Join>
  <Customer CustomerID="rnic">Radu</Customer>
  <Order OrderID="1020" CID="rnic">optical mouse</Order>
 </Join>
 <Join>
  <Customer CustomerID="sman">Mano</Customer>
  <Order OrderID="2010" CID="sman">iphone</Order>
 </Join>
 <Join>
  <Customer CustomerID="sman">Mano</Customer>
  <Order OrderID="2020" CID="sman">pocket pc</Order>
 </Join>
 <Join>
  <Customer CustomerID="sman">Mano</Customer>
  <Order OrderID="2030" CID="sman">digital camera</Order>
 </Join>
</LeftOuterJoin>
```

Expected **_GroupJoin.xml** (logically similar to **_LeftOuterJoin.xml**, where each **Customer** is followed by the group of all its associated **Order** elements, with a **Count** attribute per each **Group** element)

```
<?xml version="1.0" encoding="utf-8"?>
<GroupJoin>
 <Join>
  <Customer CustomerID="ggim">Georgy (Георгй)</Customer>
  <Group Count="0" />
 </Join>
 <Join>
  <Customer CustomerID ="pdel">Patrice</Customer>
  <Group Count="1">
   <Order OrderID="4010" CID="pdel">cuda card</Order>
  </Group>
 </Join>
 <Join>
  <Customer CustomerID ="rnic">Radu</Customer>
  <Group Count="2">
   <Order OrderID ="1010" CID="rnic">flash memory</Order>
   <Order OrderID ="1020" CID="rnic">optical mouse</Order>
  </Group>
 </Join>
 <Join>
  <Customer CustomerID ="sman">Mano</Customer>
  <Group Count="3">
   <Order OrderID ="2010" CID="sman">iphone</Order>
   <Order OrderID ="2020" CID="sman">pocket pc</Order>
   <Order OrderID ="2030" CID="sman">digital camera</Order>
  </Group>
 </Join>
</GroupJoin>
```

Submission

Submit electronically, to the COMPSCI web assignment dropbox (ADB), a **7z** archive, with a folder containing your C# source file, named **A1.cs** – and nothing else. This file must be compilable into a command-line executable named **A1.exe**, with this compiler invocation:

```
CSC /R:Newtonsoft.Json.dll /OUT:A1.exe A1.cs
```

You must not add the required **Newtonsoft.Json.dll** library and the path to CSC.exe; the marker will provide these.

Please include a README file (text or Word doc), which outlines the completed parts that need to be assessed by the marker (items from the basic requirements or bonuses).

Please recheck it and ensure that it compile and works in the labs (not only on your home machine)!

Please keep your receipt, and, just in case, keep also a copy of the submitted archive!

Deadline

Monday 17 August, 2015, 18:00. Do not leave it for the last minute, please. Remember that you can resubmit and, by default, we only consider your last submission.

This assignment carries 7% towards the final mark for the paper. Late submissions will incur penalties, 0.5% off for each hour late, for up to three days.

Appendices

Appendices 1, 2, 3 describe the starting contexts of tests **T1**, **T2**, and **T3** (from the **Test folder**). In all these three cases, their **second sequences** are **identical** and come from the **same** XML file. However, their **first sequences** are **slightly different** and come from three **different** sources: an XML file, a JSON file, an OData service. Their final outputs are **similar** but **not** identical.

Appendix 1: Test T1 (Test folder)

```
A1.EXE
   /FILE-XML  Orders.xml  Orders/Order  OrderID  OrderID
   /FILE-XML  OrderDetails.xml  OrderDetails/OrderDetail  OrderID  OrderDetailID
```

Both input sequences come from **XML** files. The first file is Orders.**xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<Orders>
 <Order>
  <OrderID>10000</OrderID>
  <CustomerID>VOID</CustomerID>
  <EmployeeID>0</EmployeeID>
 </Order>
 <Order>
  <OrderID>10249</OrderID>
  <CustomerID>TOMSP</CustomerID>
  <EmployeeID>6</EmployeeID>
 </Order>
 <Order>
  <OrderID>10248</OrderID>
  <CustomerID>VINET</CustomerID>
  <EmployeeID>5</EmployeeID>
 </Order>
  …
 <Order>
  <OrderID>11077</OrderID>
  <CustomerID>RATTC</CustomerID>
  <EmployeeID>1</EmployeeID>
 </Order>
</Orders>
```

The second file is OrderDetails.**xml:**

```
<?xml version="1.0" encoding="utf-8"?>
<OrderDetails>
 <OrderDetail>
  <OrderDetailID>000000</OrderDetailID>
  <OrderID>0</OrderID>
  <Quantity>0</Quantity>
  <UnitPrice>0.0000</UnitPrice>
 </OrderDetail>
 <OrderDetail>
  <OrderDetailID>000005</OrderDetailID>
  <OrderID>10249</OrderID>
  <Quantity>9</Quantity>
  <UnitPrice>18.6000</UnitPrice>
 </OrderDetail>
 …
 <OrderDetail>
  <OrderDetailID>002155</OrderDetailID>
  <OrderID>11077</OrderID>
  <Quantity>2</Quantity>
  <UnitPrice>13.0000</UnitPrice>
 </OrderDetail>
</OrderDetails>
```

The results of this test, **T1**, are given in the **Test folder** – please check this!

Appendix 2: Test T2 (Test folder)

```
A1.EXE
   /FILE-JSON  Orders.json  root/value  OrderID  OrderID
   /FILE-XML  OrderDetails.xml  OrderDetails/OrderDetail  OrderID  OrderDetailID
```

This test is similar to Test **T1**, except that the data for the first sequence comes in **JSON** format. The first file is Orders.**json**:

```
{"value":[
   {"OrderID":10000,"CustomerID":"VOID","EmployeeID":0},
   {"OrderID":10249,"CustomerID":"TOMSP","EmployeeID":6},
   {"OrderID":10248,"CustomerID":"VINET","EmployeeID":5},
  …
   {"OrderID":11077,"CustomerID":"RATTC","EmployeeID":1}
]}
```

This data is internally converted to XML (using JSON.NET) and wrapped – as **required by the specs** - under a single tag named **root**.

```
<root>
  <value>
    <OrderID>10000</OrderID>
    <CustomerID>VOID</CustomerID>
    <EmployeeID>0</EmployeeID>
  </value>
  <value>
    <OrderID>10249</OrderID>
    <CustomerID>TOMSP</CustomerID>
    <EmployeeID>6</EmployeeID>
  </value>
  <value>
    <OrderID>10248</OrderID>
    <CustomerID>VINET</CustomerID>
    <EmployeeID>5</EmployeeID>
  </value>
  …
  <value>
    <OrderID>11077</OrderID>
    <CustomerID>RATTC</CustomerID>
    <EmployeeID>1</EmployeeID>
  </value>
</root>
```

The final results of this test, **T2**, are similar to the results of test **T1**.

Appendix 3: Test T3 (Test folder)

A1.EXE

/URL-JSON "http://services.odata.org/Northwind/Northwind.svc/Orders()?$orderby=OrderID **desc**&$select=OrderID,CustomerID,EmployeeID&$format=json" **root**/**value** **OrderID** **OrderID**

/FILE-XML OrderDetails.**xml** OrderDetails/OrderDetail OrderID OrderDetailID

This test is similar to Tests **T1** and **T2**, except that the first sequence is sourced in **JSON** format from an **OData** service and also comes in several size-limited **chunks**, each chunk ends with a continuation note (odata.nextLink). Please also note that the result of the **OData** query gives a bit different information comparing to Orders.xml and Orders.json files provided. The differences are the descending order by OrderID and non-existence of the order with OrderID=1000. Please also note that the **_LeftSeq.xml** file generated by A1.exe is ordered by OrderID in ascending order as specified by the command-line arguments.


```
{
"odata.metadata":"http://services.odata.org/northwind/Northwind.svc/$metadata#Orders&$select=OrderID,CustomerID,EmployeeID",
"value":[
   {"OrderID":11077,"CustomerID":"RATTC","EmployeeID":1},
   {"OrderID":11076,"CustomerID":"BONAP","EmployeeID":4},
    …
   {"OrderID":10878,"CustomerID":"QUICK","EmployeeID":4}],
"odata.nextLink":"../../Northwind/Northwind.svc/Orders?$orderby=OrderID%20desc&$select=OrderID,CustomerID,EmployeeID&$skiptoken=10878,10878"
}
{
"odata.metadata":"http://services.odata.org/northwind/Northwind.svc/$metadata#Orders&$select=OrderID,CustomerID,EmployeeID",
"value":[
   {"OrderID":10877,"CustomerID":"RICAR","EmployeeID":1},
    ….
    ….
    ….
   {"OrderID":10678,"CustomerID":"SAVEA","EmployeeID":7}]
}
```

As in **T2**, each of these chunks is internally converted to XML (using JSON.NET) and wrapped under a single tag named **root**.

```
<root>
<odata.metadata>http://services.odata.org/northwind/Northwind.svc/$metadata#Orders&amp;$select=OrderID,CustomerID,EmployeeID</odata.metadata>
  <value>
    <OrderID>11077</OrderID>
    <CustomerID>RATTC</CustomerID>
    <EmployeeID>1</EmployeeID>
  </value>
  <value>
    <OrderID>11076</OrderID>
    <CustomerID>BONAP</CustomerID>
    <EmployeeID>4</EmployeeID>
  </value>
  ...
  <value>
    <OrderID>10878</OrderID>
    <CustomerID>QUICK</CustomerID>
    <EmployeeID>4</EmployeeID>
  </value>
<odata.nextLink>../../Northwind/Northwind.svc/Orders?$orderby=OrderID%20desc&$select=OrderID,CustomerID,EmployeeID&$skiptoken=10878,10878</odata.nextLink>
</root>

<root>
<odata.metadata>http://services.odata.org/northwind/Northwind.svc/$metadata#Orders&amp;$select=OrderID,CustomerID,EmployeeID</odata.metadata>
  <value>
    <OrderID>10877</OrderID>
    <CustomerID>RICAR</CustomerID>
    <EmployeeID>1</EmployeeID>
  </value>
  ....
  ....
  ....
  <value>
    <OrderID>10678</OrderID>
    <CustomerID>SAVEA</CustomerID>
    <EmployeeID>7</EmployeeID>
  </value>
</root>
```

Next, all **value** tags of these XML tags are concatenated and the **odata.metadata** and **odata.nextlink** tags, which are not anymore needed, are discarded.

The final results **_RightSeq.xml, _InnerJoin.xml** of T3 are similar to the ones in T1 and T2. The **_LeftSeq.xml, _GroupJoin.xml** and **_LeftOuterJoin.xml** are similar to T1 and T2 except for having extra node with OrderID=1000.

APPENDIX 4

Skeleton

The folder **A1Skel** contains a sample program which demonstrates some of the required tasks:

- it loads one XDocument from a given file path
- extracts a sequence of XElements, using a given XPATH expression
- sorts this sequence, according to a sorting key indicated by a second XPATH expression
  - this second XPATH expression is relative to each XElement in the extracted sequence
  - the sorting key is the text value of the first XElement or XAttribute selected by the second XPATH expression (null if the selection is empty)
- saves the sorted sequence to another given file path
- catches all exceptions and writes their text message to the standard error stream

Looping

As discussed, small penalties will be applied for each **explicit imperative loop**, i.e.

- **for** (…; …; …) { … }
- **foreach** (…) { … }
- **while** (…) { … }

You can use **recursion** to implement the loop for retrieving all ODATA chunks.

Fluent syntax vs query syntax

In this assignment, you can use any of these, or both of these, even mixed in combination, at your choice.

Just in case: the **query** (SQL-like, declarative, comprehension) syntax uses a declarative form of loops, indicated by the **from** keyword, which you can freely use, without any penalty (of course).

Errors

According to the specs, your solution must catch **all exceptions** and write their text message to the standard **error** stream (NOT to the standard output stream). The following is a comprehensive list of errors which may be tested:

- too few command line arguments

  Example: 9 arguments, instead of 10

- wrong "location" – should be one of **/FILE-XML**, **/FILE-JSON**, **/URL-JSON**

  Example: **/URL-XML**

- wrong input file path

  Example: NonExtantSample.xml

- wrong ODATA URL

  Example: http://services.odata.org/Northwind/Foo.svc

- wrong XML content (wrong format)

  Example: <tag> … </tag2>

- wrong selection XPATH expression (this XPATH expression must follow the correct format and return a sequence of XElements)

  Examples: /parts/+part, parts/part/@type

- wrong joining or sorting XPATH expression (must follow the correct format – more below)

  Example: +partno

  According to the specs, you are only required to consider the cases when the joining and sorting XPATH expressions return singleton sequences of either one XElement or one XAttribute. You are NOT required to consider legally formatted but more complex cases, such as: longer sequences, mixed sequences, other objects. Your program will NOT be tested on such more complex cases.