**11. Use of web services**

The information for Filming Scenes and Locations from New York is available on the NYC Open Data site, on the following URL

https://nycopendata.socrata.com/Business-and-Economic/Filming-Locations-Scenes-from-the-City-/qb3k-n8mm

Or enter the NYC Open Data site[[1]](#footnote-1) and search for filming locations.

From the site press the XML file button. The XML file is Interactive\_Map\_Data.xml. The structure of the XML file is given in Appendix A.

The API input on movie locations in New York which was being used in the project was therefore in XML. In (Kadluczka 2011) the different XML APIs for Microsoft Visual Studio are outlined, along with when best to use each one. There are 4 APIs for working with XML documents in Visual Studio:

* XMLReader
* XPathDocument
* XmlDocument
* XDocument/XElement (LINQ to Xml)

Each approach has its own best case scenario, all of which are outlined below.

XmlReader, as MSDN says, “provides fast, non-cached, forward-only access to XML data”. XmlReader reads in XML data from the source stream, validates the XML and returns the XML. Typically XmlReader is called by functions that tell it what to read (ReadXXXX() methods) or what to skip (MoveToXXXX()/Skip methods). The code which calls XmlReader needs to record the return values from XmlReader, as it doesn’t record them itself. NB XmlReader is the lowest level XML API. Any code handling XML will call it, either implicitly or explicitly.

Typical times when to use XmlReader:

* Handling very large files – because XmlReader does not record data it can be used where otherwise there wouldn’t be enough memory.
* Validation of Xsd schema – if you just need to validate an Xml document against its schema XmlReader is all you need.
* Your own object model caches the data – e.g. reading an Atom feed and rather than handling Xml elements and attributes you need to handle Author, Contributor and Link instances.
* You don’t need to read the entire document – XmlReader is efficient and maintainable for small usage.
* You need Simple transformations – XmlReader with XmlWriter does simple transformations (like filtering), using XmlReader to read the XML file and XmlWriter to write the output XML file.

Times when XmlReader is not indicated:

* Caching – you need random access.
* Complex processing – XmlReader is not very intuitive to use and you are trading off efficiency for readability and maintainability. Therefore it is best to limit it to small projects.

XPathDocument stores the Xml data in a cache. All the document must be read (It uses XmlReader for this), before allowing it to be queried. In order to provide better performance and lower memory footprint there can be no changes to the cached document.

Typical times when to use XmlPathDocument:

* Using XPath to query an Xml document – of course for a simple XPath expression you could just use XmlReader.
* Using Xslt to transform an Xml document – XmlPathDocument is the best choice for this as you don’t need write access to the source but will likely query the source a lot.

Where use of XPathDocument is not indicated:

* You need to change the loaded Xml document.

XmlDocument is based on the DOM (Document Object Model). Like XPathDocument it is a cache and therefore reads in the whole Xml document. You can create Xml programmatically rather than loading a file. Compared to XPathDocument XmlDocument is slower and uses more memory, but on the other hand can change the Xml data.

Typical times when to use XmlDocument:

* You require building your Xml document programmatically with some extra processing. NB XMLWriter will be better if no extra processing is required.
* You require loading and changing an XML document with some more processing. NB again if no extra processing is required then XmlReader/XMLWriter will perform better.

Times not to use XMLDocument:

* You are just doing XPath queries or Xslt transformations – XPathDocument would be better.
* You are just creating and saving the document – XmlWriter would be better.
* You are just changing the loaded document and saving it – for small files XmlReader/XmlWriter would be better.

XDocument/XElement etc. (LINQ to XML) was added to the .NET framework version 3.5, with a new method of querying (Language Integrated Query - LINQ). With these APIs you can do most of what you can do with XmlDocument, but much more easily. For instance, you can build and change Xml documents, validate against the Xsd, do Xslt transformations, query with XPath etc. You gain a way of querying Xml documents (LINQ). Microsoft recommend usage of LINQ over XmlDocument in new projects.

Unlike XDocument most LINQ queries will not cache the whole Xml document, but rather IENumerable<T> where T is XElement, or XAttribute etc, because of how IENumerable works (deferred execution).

When LINQ to XML usage is indicated:

* You need to create your Xml document programmatically and do extra processing.
* You need to load and change your Xml document and do extra processing.
* You need to query an XML document using LINQ.
* You need to do simple transformations in a lighter manner than Xslt.

When not to use LINQ to XML:

* You just create the document and save it – use XmlWriter instead.
* You just load the document, make a simple change and save it – use XMLReader/XMLWriter.
* You need to do a lot of XPath queries – you could consider replacing them with LINQ queries, or if you don’t change the document try XPathDocument.

The most abstract API is LINQ to XML. Therefore using it would be better for readability, maintainability and ease of use. Although this project is relatively small it is a multi-programmer project so readability and maintainability are very important. Furthermore in a time limited project ease of use is very important. Also LINQ to XML is the most up to date API in the .NET framework and Microsoft recommends its use for future projects. Therefore it is safe to assume Microsoft has more support for this API. For this project we are simply loading the XML document and reading it into memory and the database, which means that LINQ queries are a perfect fit. Performance-wise it takes a few seconds to initially load the XML file but for a time-limited project this is acceptable.

**13. References**

Kadluczka, P. (2011) ‘Effective Xml Part 1: Choose the right API’ in *Microsoft XML Team Blog,* Available from: <http://blogs.msdn.com/b/xmlteam/archive/2011/09/14/effective-xml-part-1-choose-the-right-api.aspx> [Accessed 14th July 2012]

**A. New York Filming Locations XML Data**

The XML Data is plainly downloaded from an MS Office Excel spreadsheet file. This can be seen from its structure. It is divided into rows and cells unlike a straight forward XML file, which might contain the following excerpt.

<Film>Die Hard

<Location>

<Place>Manhattan

<Latitude> 40.7792578185718</Latitude>

<Longitude>-73.9815366268158</Longitude>

</Place>

</Film>

In the actual XML file the first row contains nothing relevant. The second row contains the titles for each column. Further rows contain the data such as film title and location name. The type of data is given by the cell position in the row. Each cell’s type is given by that cell’s title in the second row. For instance if cell 1 in row 2 is “Film” then all rows (after the second) contain the film title in cell 1. Similarly “Location Display Text” indicates which cell contains the location names. To indicate more than one shooting location for a film the film name is repeated in the next entry. For instance:

{“Die Hard”, “Manhattan”, “30.0”, “-75.0”},

{“Die Hard”, “Queens”, “35.0”, “-74.0”},

The user interface sees this as one film (“Die Hard”) with two locations (“Manhattan”, and “Queens”). The input XML data is structured as follows:

<Workbook>

<DocumentProperties>

...

</DocumentProperties>

<OfficeDocumentSettings>

...

</OfficeDocumentSettings>

<ExcelWorkbook>

...

</ExcelWorkbook>

<Styles>

...

</Styles>

<Worksheet ss:Name="FullMapList">

<Names>

...

</Names>

<Table>

<Column>

...

</Column>

<Row>

...

<Row ss:Index="2" ss:Height="18.75">

<Cell ss:StyleID="s32"><Data ss:Type="String">Scenes from the City: Locations</Data></Cell>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s97"/>

<Cell ss:StyleID="s57"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s90"/>

<Cell ss:StyleID="s47"/>

<Cell ss:StyleID="s48"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s32"/>

<Cell ss:StyleID="s33"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s31"/>

<Cell ss:StyleID="s56"/>

</Row>

<Row ss:Height="38.25">

<Cell ss:StyleID="s24"><Data ss:Type="String">Film</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Year</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s96"><Data ss:Type="String">URL Encoded name</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Image File Name</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Agency Credit</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Artist Credit</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Director/Filmmaker Name</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Director/Filmmaker IMDB Link</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s91"><Data ss:Type="String">Location Display Text</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s49"><Data ss:Type="String">LATITUDE</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s49"><Data ss:Type="String">LONGITUDE</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Borough</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Neighborhood</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Scene Type</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">Media</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s24"><Data ss:Type="String">IMDB LINK</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Client or book location indicator</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Notes</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Book Image</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Book Page</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">Display?</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s52"><Data ss:Type="String">IMAGE OF LOCATION</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

</Row>

...

<Row ss:Height="25.5" ss:StyleID="s37">

<Cell ss:StyleID="s26"><Data ss:Type="String">Hannah and Her Sisters</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><Data ss:Type="Number">1986</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s95"><Data ss:Type="String">Hannah%20and%20Her%20Sisters</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s46"><Data ss:Type="String">Directed by</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><Data ss:Type="String">Woody Allen</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">http://www.imdb.com/name/nm0000095/</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s93"><Data ss:Type="String">W. 95th St. and West End Ave.&lt;br&gt;Upper West Side&lt;br&gt;Manhattan</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s50"><Data ss:Type="Number">40.794600000000003</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s50"><Data ss:Type="Number">-73.973699999999994</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">Manhattan</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">Upper West Side</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">N/A</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">Film</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">http://www.imdb.com/title/tt0091167/</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><Data ss:Type="String">95th Street and West End Avenue</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s28"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s43"><Data ss:Type="String">N</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s27"><NamedCell ss:Name="\_FilterDatabase"/></Cell>

<Cell ss:StyleID="s43"><Data ss:Type="String">N</Data><NamedCell

ss:Name="\_FilterDatabase"/></Cell>

</Row>

...

1. https://nycopendata.socrata.com/ [↑](#footnote-ref-1)