# Background Research and Investigations

The API input on movie locations in New York which was being used in the project was in XML. In (Kadluczka 2011) the different XML APIs 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). [EXPAND in new web page]

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.

# 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]