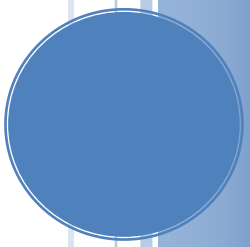
Getting Started With LINQ to RDF

Semantic web applications in .NET

This short document shows you the steps needed to write a simple semantic web application. It covers the tools and systems prerequisites, the techniques and the expected behavior. All examples are in C#, although LinqToRdf will work on all .NET language.



Andrew Matthews

6/19/2007

GETTING STARTED WITH LINQ TO RDF

Semantic web applications in .NET

WHAT ARE THE COMPONENTS OF A SEMANTIC WEB APPLICATION?

Within the context of this document, Semantic Web Application means "*application that uses RDF and related technologies".* That is – an application that represents information as a graph structure using RDF. It's beyond the scope of this document to explain the whole pyramid of standards and technologies needed to support the semantic web. Instead I'll give enough context for you to know what steps are required to get your Semantic Web Application working.

Figure 1 Major technologies employed with LinqToRdf

LinqToRdf uses the SemWeb class library by Joshua Tauberer, which provides a platform for working with OWL and SPARQL. It uses the .NET 3.5 namespace System.Query which will be released as part of Visual Studio .NET 2008.

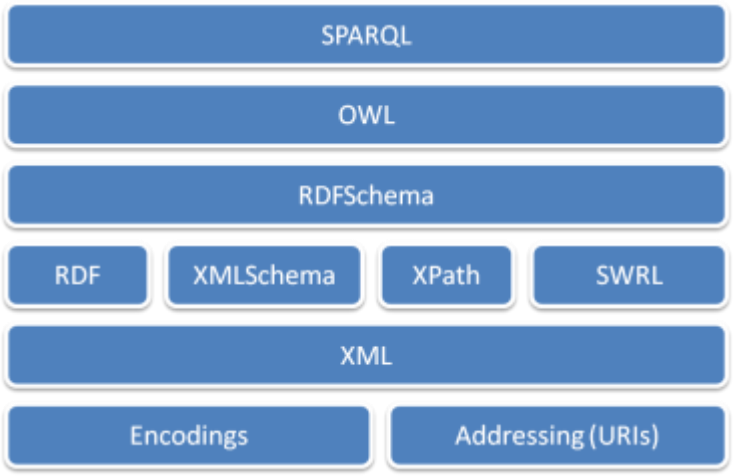


Figure 2 The semantic web technology stack

Figure 2 above shows the hierarchy of technologies that are involved in the semantic web.

What do you need to do semantic web programming with LinqToRdf?

LinqToRdf requires an IDE that supports the latest version of LINQ, which at the time of writing was Visual Studio .NET Orcas Beta 1. The LINQ framework is still subject to change, and as it stabilizes, the LinqToRdf library will be updated to reflect the new changes.

Get the latest release, if you haven't already, and install it.

The

Where to get LinqToRdf

You can download the latest release from Google code. Check The linqtordf-discuss discussion forum for announcements of newer releases.

INSTALLATION PROCEDURE

Installation is a simple matter of double clicking the MSI file, and deciding where to install the assemblies for LinqToRdf.

CREATING AN ONTOLOGY

The details of the OWL standard are beyond the scope of this document. The standards document is found at the W3C[1]. There are various tools available for creating RDF, and it is important to know that the SemWeb library can understand the Notation 3 syntax[2], which is a human readable (non-XML) variant of RDF. The examples we'll be using in the rest of this document use Notation 3 (or N3 for short). You might want to consider tools such as Protégé, or CMapTools for graphical environments, or notepad if you're hard-core.

Let's start with a simple ontology for recording MP3 files. The ontology file should have an extension of n3. Let's call ours music.n3. First you define the XML namespaces you will be working with:

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix daml: <http://www.daml.org/2001/03/daml+oil#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix xsdt: <http://www.w3.org/2001/XMLSchema#>.
@prefix : <http://aabs.purl.org/ontologies/2007/04/music#> .
```

This imports standard namespaces for OWL, RDF, XML Schema datatypes and others. It also defines a default namespace to be used for all classes and properties that are going to be defined in the rest of the document. Now let's define some classes:

```
:Album a owl:Class.
:Track a owl:Class.
:title rdfs:domain :Track;
       rdfs:range  xsdt:string.
```

[1] The best place to start is with http://www.w3.org/2004/OWL/ - the specification for OWL. You should also be aware of http://www.w3.org/TR/2000/CR-rdf-schema-20000327/ which is the specification for RDFS (RDF Schema).

[2] See Tim Berners-Lee's Guide for further information: http://www.w3.org/2000/10/swap/Primer

```
:artistName
        rdfs:domain :Track;
        rdfs:range  xsdt:string.
:albumName
        rdfs:domain :Track;
        rdfs:range  xsdt:string.
:year
        rdfs:domain :Album;
        rdfs:range  xsdt:integer.
:genreName
        rdfs:domain :Track;
        rdfs:range  xsdt:string.
:comment
        rdfs:domain :Track;
        rdfs:range  xsdt:string.
:isTrackOn
        rdfs:domain :Track;
        rdfs:range  :Album.
:fileLocation
        rdfs:domain :Track;
        rdfs:range  xsdt:string.
```

What I've done is create a class Track of type owl:Class. After the class declaration, I defined some properties on the Track Class (:title, :artistName &c). Because the prolog section previously defined a default namespace, these declarations are now in the `<http://aabs.purl.org/ontologies/2007/04/music#>` namespace. That takes care of class declarations, now we need to create some data for MP3 files.

Create another file called mp3s.n3 and add the following:

```
@prefix ns1: <http://aabs.purl.org/ontologies/2007/04/music#> .
ns1:Track_-861912094 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ns1:Track ;
        ns1:title "History 5 | Fall 2006 | UC Berkeley" ;
        ns1:artistName "Thomas Laqueur" ;
        ns1:albumName "History 5 | Fall 2006 | UC Berkeley" ;
        ns1:year "2006" ;
        ns1:genreName "History 5 | Fall 2006 | UC Berkeley" ;
        ns1:comment " (C) Copyright 2006, UC Regents" ;
        ns1:fileLocation "C:\\Users\\andrew.matthews\\Music\\hist5_20060829.mp3" .
ns1:Track_-1378138934 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ns1:Track ;
        ns1:title "History 5 | Fall 2006 | UC Berkeley" ;
        ns1:artistName "Thomas Laqueur" ;
        ns1:albumName "History 5 | Fall 2006 | UC Berkeley" ;
        ns1:year "2006" ;
        ns1:genreName "History 5 | Fall 2006 | UC Berkeley" ;
        ns1:comment " (C) Copyright 2006, UC Regents" ;
        ns1:fileLocation "C:\\Users\\andrew.matthews\\Music\\hist5_20060831.mp3" .
ns1:Track_583675819 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ns1:Track ;
        ns1:title "Rory Blyth: The Smartest Man in the World\u0000" ;
        ns1:artistName "Rory Blyth\u0000" ;
        ns1:albumName "Rory Blyth: The Smartest Man in the World\u0000" ;
        ns1:year "2007\u0000" ;
        ns1:genreName "Rory Blyth: The Smartest Man in the World\u0000" ;
        ns1:comment "Einstein couldn't do it again if he lived today. He'd be too
distracted by the allure of technology, and by all those buttheads at Mensa trying to
prove how smart they are." ;
        ns1:fileLocation "C:\\Users\\andrew.matthews\\Music\\iTunes\\iTunes
Music\\Podcasts\\Rory Blyth_ The Smartest Man in the Worl\\A Few Thoughts on the Subject
of Gen.mp3" .
```

These entries were taken randomly from a list of podcasts that I subscribe to. In addition, I wrote a program to create them, but you ***could*** do it by hand if you want to. ☺ In this file I have defined a namespace ns1 as referring to what was the default namespace in music.n3. It doesn't matter what you call it – I called it ns1, because that's what my program wanted to do. The point is that the type ns1:Track

in this file refers to the :Track class defined in music.n3. the triple store that we'll get to shortly will be able to make sense of that in order to know that a ns1:Track has a title, artist etc. It is also able to work out the types of the properties (which just happens to be string for the moment).

That's it. Well that's all there is to creating an ontology. Later on, we'll get onto the more complicated task of linking types together using ObjectProperties, but for now you have an ontology and some data that uses it.

HOSTING YOUR ONTOLOGY

Since the uptake of semantic web technologies has been pretty patchy in the .NET domain your best bet for industrial strength RDF triple stores will (for now) lie in the Java domain, and there are various triple store solutions that can be used. For this guide I shall confine us to .NET by using Joshua Tauberer's SPARQL enabled HttpHandler for ASP.NET, which is sufficient to demonstrate how LinqToRdf can connect to a SPARQL compatible triple store.

To use the HttpHandler as a triple store for music.n3, create an ASP.NET application in visual studio. Place the following into configuration section of the web.config of the project:

```
<configSections>
    <section name="sparqlSources" type="System.Configuration.NameValueSectionHandler,
System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
</configSections>
```

Next, add the following:

```
<sparqlSources>
    <add key="/[your vdir here]/SparqlQuery.aspx" value="n3:[your path here]\mp3s.n3"/>
</sparqlSources>
```

The name of the file doesn't matter – it doesn't exist. What this does is link the URL for the file SparqlQuery.aspx to the SPARQL HttpHandler that we next add to the system.web section of the web.config:

```
<httpHandlers>
   <!-- This line associates the SPARQL Protocol implementation with a path on your
        website. With this, you get a SPARQL server at http://yourdomain.com/sparql.  -->
   <add verb="*" path="SparqlQuery.aspx" type="SemWeb.Query.SparqlProtocolServerHandler,
SemWeb.Sparql" />
</httpHandlers>
```

This uses the HttpHandler defined in SemWeb to accept SPARQL queries and run them against the triples defined in mp3s.n3. That's all that's needed to turn your ASP.NET into a semantic web triple store! Yes, it's that easy.

LINKING TO THE ONTOLOGY FROM .NET

Now that we have an ontology defined, and somewhere to host it that understands SPARQL you can now get started with using LinqToRdf. First you need to create a .NET class to contain the data from music.n3. There is not currently a code generator to create this class for you, although there are plans…

First you should create a new class called Track in a file called Track.cs.

```
using LinqToRdf;
namespace RdfMusic
{
    [OntologyBaseUri("http://aabs.purl.org/ontologies/2007/04/music#")]
    [OwlClass("Track", true)]
    public class Track : OwlInstanceSupertype
    {
        [OwlProperty("title", true)]
        public string Title
        {
            get { return title; }
            set { title = value; }
        }
        [OwlProperty("artistName", true)]
        public string ArtistName
        {
            get { return artistName; }
            set { artistName = value; }
        }
        [OwlProperty("albumName", true)]
        public string AlbumName
        {
            get { return albumName; }
            set { albumName = value; }
        }
        [OwlProperty("year", true)]
        public string Year
        {
            get { return year; }
            set { year = value; }
        }
        [OwlProperty("genreName", true)]
        public string GenreName
        {
            get { return genreName; }
            set { genreName = value; }
        }
        [OwlProperty("comment", true)]
        public string Comment
        {
            get { return comment; }
            set { comment = value; }
        }
        [OwlProperty("fileLocation", true)]
        public string FileLocation
        {
            get { return fileLocation; }
            set { fileLocation = value; }
        }
        [OwlProperty("rating", true)]
        public int Rating
        {
            get { return rating; }
            set { rating = value; }
        }
        private string title;
        private string artistName;
        private string albumName;
        private string year;
        private string genreName;
        private string comment;
        private string fileLocation;
        private int rating;
    }
}
```

The class is just the same as any other entity class except that the class and its properties have been annotated with the OwlClass, OwlProperty and OntologyBaseUri attributes[3]. The critical bit to get right is to use the same URI in OntologyBaseUri as we used in music.n3 and mp3s.n3 for the namespace definitions. Using OntologyBaseUri allows you to define all other attributes as relative URIs which makes for a much more readable source file. The owlClassAttribute defines our .NET class RdfMusic.Track to correspond to the OWL class http://aabs.purl.org/ontologies/2007/04/music#Track. Likewise the FileLocation property defined on it corresponds to the RDF datatype property http://aabs.purl.org/ontologies/2007/04/music#fileLocation. The Boolean true on these attributes simply tells LinqToRdf that the URIs are relative. It then knows enough to be able to work out how to query for the details needed to fill each of the properties on the class Track.

This approach is deliberately as close as possible to LINQ to SQL. It is hoped that those who are already familiar with DLINQ (as LINQ to SQL used to be known) will be able to pick this up and start working with it quickly. In DLINQ, instead of URIs for resources defined in an ontology, you would find table and column names.

That's all you need to be able to model your ontology classes in .NET. Now we will move on to the techniques needed to query your RDF triple store.

QUERYING THE ONTOLOGY USING SPARQL

The steps to start making queries are very simple. First just create a simple LINQ enabled console application called MyRdfTest. Open up Program.cs up for editing, and add namespace import statement for System.Query, LinqToRdf and SemWeb:

```
using System;
using LinqToRdf;
using System.Query;
```

In Main, create a TripleStore object with the location of the SPARQL server:

```
private static void Main(string[] args)
{
    TripleStore ts = new TripleStore();
    ts.EndpointUri = @"http://localhost/linqtordf/SparqlQuery.aspx";
    ts.QueryType = QueryType.RemoteSparqlStore;
```

TripleStore is used to carry any information needed about the triple store for later use by the query. In this case I set up an IIS virtual directory on my local machine called linqtordf, and followed the steps outlined early. The QueryType just indicates to the query context that we will be using SPARQL over HTTP. That tells it what types of connections, commands, XML data types and the query language to use.

Now we're ready to perform the LINQ query. We'll get all of the tracks from 2007 that have a genre name of "*Rory Blyth: The Smartest Man in the World*". We'll create a new anonymous type to store the results in, and we're only interested in the Title and the FileLocation.

```
var q = from t in new RDF(ts).ForType<MyTrack>()
        where t.Year == "2007" &&
        t.GenreName == "Rory Blyth: The Smartest Man in the World"
        select new {t.Title, t.FileLocation};
```

[3] Navigate to http://linqtordf.googlecode.com/svn/trunk/src/linqtordf/Attributes.cs for more information

Then we'll just iterate over the results and wait for a keypress before quitting.

```
foreach(var track in q){
        Console.WriteLine(track.Title + ": " + track.FileLocation);
}
Console.ReadKey();
```

That's all there is to it. Of course there's a lot more going on behnd the scenes, but the beauty of LINQ is that you don't need to see all of that while you're only interested in getting some Tracks back! In the next section I'll give some links that you can go to if you want to know what's going on under the hood.

Reference Material