.NET

Using .NET to access and use Stardog Server

Taught by:

Joseph Hayes
Senior Software Engineer

Learning Objectives

- Install dotNetRDF library
- Connect to Stardog and create a database using dotNetRDF
- Query Stardog using dotNetRDF
- Install the TrinityRDF object mapper library
- Connect to Stardog using Trinity RDF and create ontology mappings
- Query Stardog using Trinity RDF

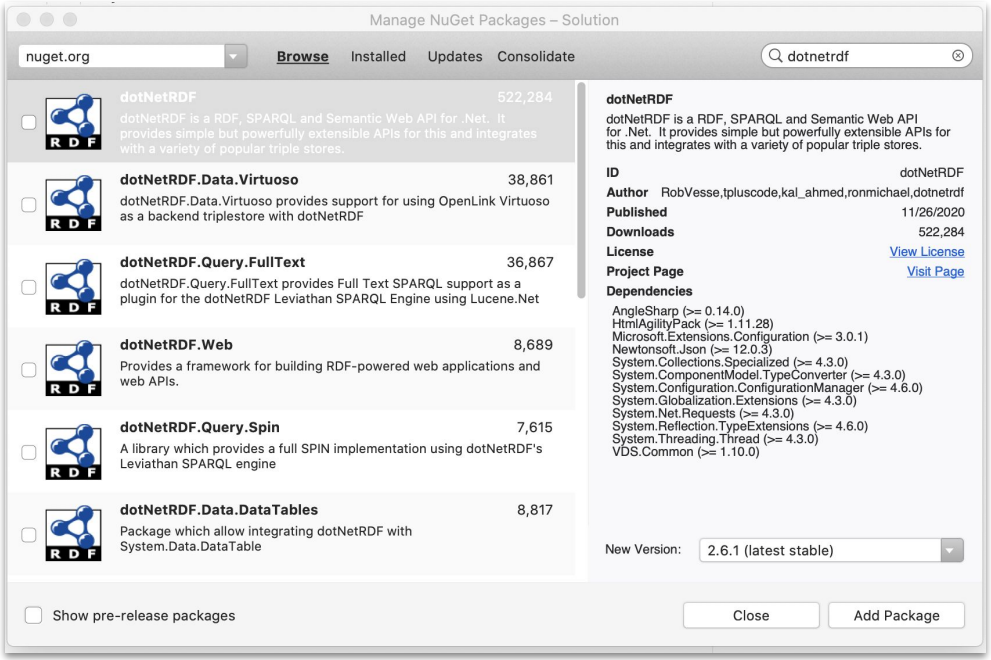
.NET

Introduction: dotNetRDF

- .NET developers can use dotNetRDF as a client for Stardog, providing access to functions such as executing SPARQL queries, administrating Stardog, and the Inference API
- dotNetRDF's implementation uses Stardog's HTTP API. For more information, go to the Stardog's HTTP API documentation

dotNetRDF Installation

- In typical .NET environments, use NuGet to install dotNetRDF



Development: Managing a Server

- To connect to a Stardog server so we can run administrative commands against it, we create an instance of dotNetRDF's `VDS.RDF.Storage.Management.StardogServer` class
  - The `StardogServer` class constructor takes a Stardog server URL and a user's username and password as parameters. It can also take proxy server information if a proxy server is required to connect to the Stardog server

Development: Managing a Server

- With a valid instance of the `StardogServer` class, we can perform the following functions:
  - ListStores - get a list of databases defined on the server
  - DeleteStore - delete a database
  - CreateStore - create a database using a template

Development: Working with a Server

- To work with graphs in Stardog database, we need to create an instance of the `VDS.RDF.Storage.StardogConnector` class
  - The `StardogConnector` class constructor takes a Stardog server URL, database name, and a user's username and password as parameters. It can also take proxy server information if a proxy server is required to connect to the Stardog server

Development: Working with a Server

- With a valid instance of the `StardogServer` class, we can perform the following functions:
  - UpdateGraph - add or remove triples to a graph
  - SaveGraph - save a graph to the database
  - Query - run a SPARQL query. We can run a reasoning query by simply setting the reasoning parameter to true
  - ListGraphs - list the graphs in the database

Demo

Using dotNetRDF to Connect & Create Databases

STARDOG ACADEMY

Creating Connections

- Below are two examples (Server and Connector) of how to establish a connection to a Stardog server

Server

```
const string SERVER_URL = "http://localhost:5820";
const string STARDOG_USERNAME = "admin";
const string STARDOG_PASSWORD = "admin";
StardogServer stardog = new StardogServer(SERVER_URL, STARDOG_USERNAME, STARDOG_PASSWORD));
```

Connector

```
const string SERVER_URL = "http://localhost:5820";
const string STARDOG_USERNAME = "admin";
const string STARDOG_PASSWORD = "admin";
const string DATABASE_NAME = "MyDB";
StardogConnector stardogConn = new StardogConnector(SERVER_URL, DATABASE_NAME, STARDOG_USERNAME, STARDOG_PASSWORD);
```

Creating/Dropping Database

- Create a new database

```
// stardog is an instance of StardogServer
IStoreTemplate template =
stardog.GetDefaultTemplate("MyDB");
stardog.CreateStore(template);
```

- Drop an existing database

```
stardog.DeleteStore("MyDB");
```

Creating/Dropping Database

- Check if a database called “myDB” exists. If it does, drop it. Then we create a new database with the default settings:

```
if (stardog.ListStores().Contains(“myDB”)) {
    stardog.DeleteStore(“myDB”);
}

IStoreTemplate template =
stardog.GetDefaultTemplate(“myDB”);
stardog.CreateStore(template);
```

Adding Data

- In this example, we create a triple and add it to the database using an instance of dotNetRDF's StardogConnector

```
// make a triple
Graph g = new Graph();
INode s = g.CreateBlankNode();
INode p = g.CreateUriNode(new Uri(RdfSpecsHelper.RdfType));
INode o = g.CreateUriNode(new Uri("http://example.org/Example"));
Triple t = new Triple(s, p, o);

// add it to a graph in the store
if (stardogConn.UpdateSupported) {
	// UpdateGraph takes lists of Triples to add or remove (or null if you do not want to perform that operation)
	stardogConn.UpdateGraph("http://example.org/graph", new Triple[] { t }, null);
}
```

Adding Data

- In this example, we load a file into the database using an instance of dotNetRDF's StardogConnector

```
Graph g = new Graph();
g.LoadFromFile("example.rdf");

// Set its BaseUri property to the graph URI
g.BaseUri = new Uri("http://example.org/graph");

// write it to the store
if (!stardogConn.IsReadOnly) {
      stardogConn.SaveGraph(g);
}
```

Demo

Querying Stardog Using dotNetRDF

Querying

- In this example, we query the database by running a SPARQL query. Reasoning is off by default.

```
object results = stardogConn.Query("SELECT * WHERE { GRAPH ?g { ?s ?p ?o } } LIMIT 100");
if (results is SparqlResultSet) {
      SparqlResultSet rset = (SparqlResultSet)results;
      foreach (SparqlResult r in rset)
      {
      Console.WriteLine("RESULT: {0}", r.ToString());
      }
}
```

Querying Continued

- In this example, we query the database by running a SPARQL query. Reasoning is turned on by passing “true” as the second argument.

```
object results = stardogConn.Query("SELECT * WHERE { GRAPH ?g { ?s ?p ?o } } LIMIT 100", true);
if (results is SparqlResultSet) {
      SparqlResultSet rset = (SparqlResultSet)results;
      foreach (SparqlResult r in rset)
      {
      Console.WriteLine("RESULT: {0}", r.ToString());
      }
}
```

Demo

TrinityRDF

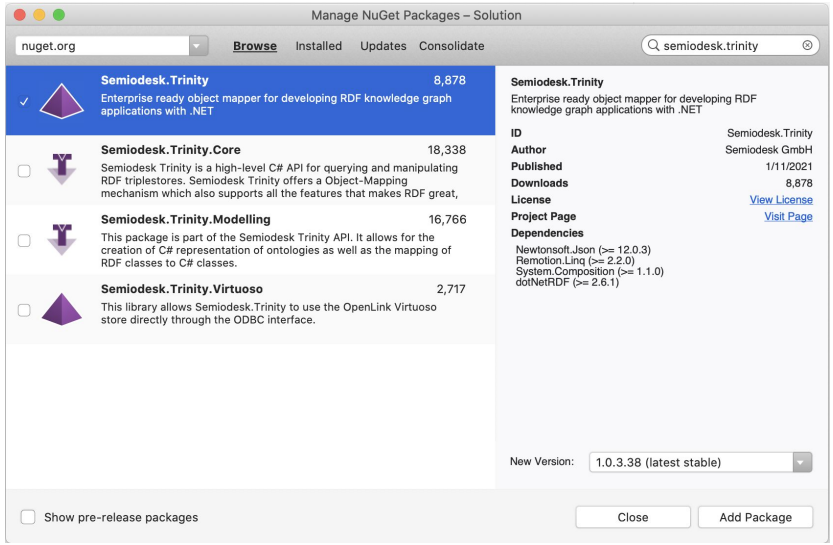
Introduction: TrinityRDF

- .NET developers who want an experience similar to Entity Framework can use TrinityRDF to interact with the Stardog server. A big advantage to using TrinityRDF is being able to use LINQ to query the database
- TrinityRDF is built on top of dotNetRDF

Installing TrinityRDF Object Mapper Library

TrinityRDF Installation

- In typical .NET environments, use NuGet to install Semiodesk.Trinity



Demo

Connecting TrinityRDF & Ontology Mapping

TrinityRDF: Connecting to Stardog

- We provide a connection string to the `StardogStoreProvider` class (provided by TrinityRDF) so we can connect Trinity to our Stardog server

```
// make Trinity aware of ontologies and mapped classes in our assembly
OntologyDiscovery.AddAssembly(Assembly.GetExecutingAssembly());
MappingDiscovery.RegisterCallingAssembly();

// TODO: Change the connection string for your Stardog instance's host, port and credentials
const string connectionString = "provider=stardog;host=http://localhost:5820;uid=admin;pw=admin;sid=music";

// Load the stardog store provider
StoreFactory.LoadProvider<StardogStoreProvider>();

// Connect to the stardog store
IStore store = StoreFactory.CreateStore(connectionString);
```

TrinityRDF: Loading ontologies

- We have to make Trinity aware of our ontologies, so we can use a special XML configuration file called ontologies.config. This file specifies the namespace, prefix, and file source for each ontology.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <ontologies namespace="example" >
        <!--http://www.w3.org/1999/02/22-rdf-syntax-ns#-->
        <ontology uri="http://www.w3.org/1999/02/22-rdf-syntax-ns#" prefix="rdf">
        <filesource location="ontologies/rdf.ttl"/>
        </ontology>
        <!--http://www.w3.org/2000/01/rdf-schema#-->
        <ontology uri="http://www.w3.org/2000/01/rdf-schema#" prefix="rdfs">
        <filesource location="ontologies/rdfs.ttl"/>
        </ontology>
        <ontology uri="http://stardog.com/tutorial/" prefix="music">
            <filesource location="ontologies/music_schema.ttl"/>
        </ontology>
  </ontologies>
</configuration>
```

TrinityRDF: Loading ontologies

- We can use the ontologies.config file to initialize our Stardog connection via the InitializeFromConfiguration method of our store's instance.

```
store.InitializeFromConfiguration(Path.Combine(Environment.CurrentDirectory,
"ontologies.config"));
```

TrinityRDF: Create mapping classes

- Trinity RDF reads the ontologies.config file and uses it to generate annotations which we can use to create object model classes that map to RDF classes.

```
using example;
using Semiodesk.Trinity;
using System;
using System.Collections.Generic;
namespace TrinityConsoleSample.ObjectModels {
    /// <summary>
    /// A music group; that is, is a group of people creating and performing music together.
    /// </summary>
    [RdfClass(MUSIC.Band)]
    class Band : Artist {
        /// <summary>
        /// A member of a band. Does not distinguish between past vs current members
        /// </summary>
        [RdfProperty(MUSIC.member)]
        public List<SoloArtist> Members { get; set; }
        public Band(Uri uri) : base(uri) { }
    }
}
```

Demo

Querying Stardog Using TrinityRDF

TrinityRDF: Querying data

- One the big advantages of using TrinityRDF is being able to use LINQ to query data - a developer no longer needs to learn SPARQL to use Stardog!

```
var matchingBands = Model.AsQueryable<Band>()
    .Where(band => band.Name.StartsWith("the beatles",
StringComparison.InvariantCultureIgnoreCase))
    .ToList();

var firstBand = matchingBands.First();

var albums = Model.AsQueryable<Album>().Where(album => album.Artist == firstBand).ToList();
Console.WriteLine($"Band {firstBand.Name} has {albums.Count} albums");
```

TrinityRDF: Querying data

- LINQ allows us to aggregate data

```
var rand = new Random();
var randomAlbum = albums.Skip(rand.Next(albums.Count)).First();

Console.WriteLine($"Track list for album '{randomAlbum.Name}' by {randomAlbum.Artist.Name}:");

foreach (var track in randomAlbum.Tracks) {
    Console.WriteLine($"\tTrack: {track.Name}, Length:
{TimeSpan.FromSeconds(track.Length).TotalMinutes.ToString("0.##")}");
    foreach (var writer in track.Writers) {
        Console.WriteLine($"\t\tWritten by: {writer.Name}");
    }
}
int totalLength = randomAlbum.Tracks.Sum(s => s.Length);

Console.WriteLine($"Total length is {TimeSpan.FromSeconds(totalLength).TotalMinutes.ToString("0.##")}");
```

TrinityRDF: Querying data

- We can even use reasoning queries by passing true to TrinityRDF's AsQueryable.
- In the following example, our ontology defines Artist as the base class of both SoloArtist and Band.

```
var artists = Model.AsQueryable<Artist>(true);

Console.WriteLine($"With reasoning set to 'true', the artists query returned
{artists.Count()} records");
```

Demo

Learning Objectives

Learning Objectives

- Install dotNetRDF library
- Connect to Stardog and create a database using dotNetRDF
- Query Stardog using dotNetRDF
- Install the TrinityRDF object mapper library
- Connect to Stardog using Trinity RDF and create ontology mappings
- Query Stardog using Trinity RDF

Thank you