

<http://iringtools.org>

SDK Guide

Version 1.02.01

| Notice |
| --- |
| This is a controlled document. Unauthorized access, copying, replication or usage for a purpose other than for which it is intended, are prohibited.  All trademarks that appear in the document have been used for identification purposes only and belong to their respective companies. |

Document Release Note

Document Details

|  |  |  |
| --- | --- | --- |
| Name | Version no. | Description |
| ***iRINGTools*** SDK Guide | 1.03.00 | ***iRINGTools*** SDK Guide |

Reviewers

|  |  |  |
| --- | --- | --- |
| L | Role | Organization Unit |
| Lee Colson | Documentation | ***iRINGTools*** |
| Hahn Le | Software Engineer | ***iRINGTools*** |

|  |
| --- |
| **Terms of License**  Copyright (c) 2010, iringug.org  All rights reserved.  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:  \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.  \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.  \* Neither the name of the iringug.org nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.  THIS DOCUMENT IS PROVIDED BY iringug.org ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL iringug.org BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. |

Contents

[List of Abbreviations vi](#_Toc260046465)

[1 Overview 7](#_Toc260046466)

[1.1 Data Exchange and Data Services 8](#_Toc260046467)

[1.2 Adapter Framework 9](#_Toc260046468)

[1.3 Dependency Injection 10](#_Toc260046469)

[1.4 Adapter Layers 10](#_Toc260046470)

[2 Custom Data Layer 11](#_Toc260046471)

[2.1 Components 12](#_Toc260046472)

[2.2 Naming Convention 12](#_Toc260046473)

[2.3 Data Dictionary XML 12](#_Toc260046474)

[2.3.1 Data Dictionary format 13](#_Toc260046475)

[2.3.2 Creating a Data Dictionary XML 13](#_Toc260046476)

[2.4 Custom Data Layer DLL 14](#_Toc260046477)

[2.4.1 Entity Classes 14](#_Toc260046478)

[2.4.2 Entity Data Object Classes 15](#_Toc260046479)

[2.4.3 The IDataLayer Interface 16](#_Toc260046480)

[2.4.4 IDataLayer implementation class 16](#_Toc260046481)

[2.5 Binding Configuration 16](#_Toc260046482)

[2.5.1 Dependency Injection 16](#_Toc260046483)

[2.5.2 Binding Configuration XML 17](#_Toc260046484)

[2.6 Example Implementation 17](#_Toc260046485)

The total number of pages in this document, including the cover page, is .

List of Abbreviations

|  |  |
| --- | --- |
| Acronym | Description |
| iRING | ISO 15926 Realtime Interoperability Network Grid |
| ISO | International Organization for Standardization |
| RDSWIP | Reference Data Service Work in Progress |
| SP | Service Pack |
| GUI | Graphical User Interface |
| IIS | Internet Information Services |
| MIME | Multipurpose Internet Mail Extensions |
| OLTP | Online Transaction Processing |
| API | Application Programming Interface |
| CRUD | Create, Read, Update and Delete |
| LAN | Local Area Network |
| FIPS | Federal Information Processing Standard |

1. Overview

***iRING*** is a set of information interoperability and integration protocols and reference data that are compliant with the ISO 15926, Parts 7, 8, and 9 standards, which builds and depends on ISO 15926 Parts 1 through 6.

***iRINGTools*** is a set of free, public domain, open source (BSD 3 license) software applications and utilities that implement ***iRING*** protocols. ***iRINGTools*** provide users with production ready deployable solutions. ***iRINGTools*** also provides technology solution providers with usage patterns for the implementation of ***iRING*** protocols in their respective solutions.

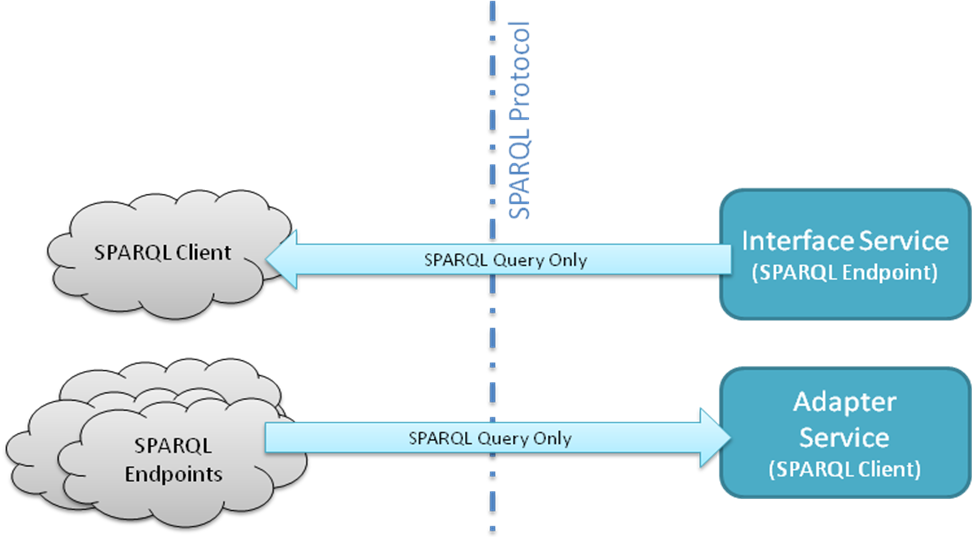
***iRINGTools*** is deployed in two packages, ***iRINGTools*** Adapter and ***iRINGTools*** Sandbox. These packages are separated for deployment purposes. The ***iRINGTools*** Adapter is used to map and transform legacy data to ISO 15926 representations. The ***iRINGTools*** Sandbox is used to host Reference Data of local interest or draft quality for use with the ***iRINGTools*** Adapter. ***iRINGTools*** Sandbox data should be used locally only or eventually, be submitted to the IDS-ADI Core Team for addition into the RDSWIP. The ***iRINGTools*** Adapter will use reference data from configured Sandboxes, as well as the RDSWIP.

This guide focuses on the ***iRINGTools*** Adapter. The adapter is an extenisble generic web service adapter. It uses NHibernate to connect to databases for the purpose of exposing their data as a web service. Besides the native NHibernate data layer, the ***iRINGTools*** Adapter can be extended with a custom Data Layer. Custom Data Layers can handle other sources of data, such as an API, Excel, CSV file, etc. This document provides step by step detailed instructions for how to create and deploy a custom Data Layer with ***iRINGTools*** Adapter.

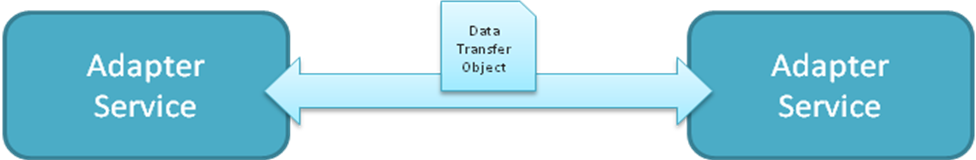
* 1. Data Exchange and Data Services

The ***iRINGTools*** Adapter can produce data representations at different levels of the ISO standard. It can produce Part 7 XML, Part 8 RDF/OWL, or a Part 9 Facade. The purpose of exposing data in these various formats is to enable data exchange between applications. The exposed data can also be consumed by ESBs or used for reporting purposes.

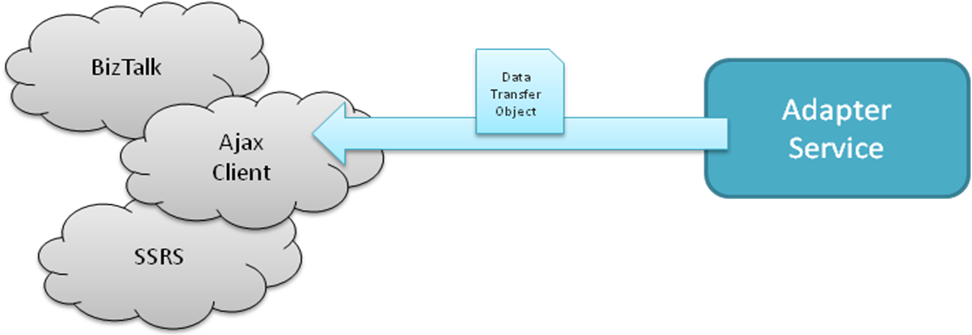
For the purposes of data exchange the adapter enables pull based data exchange. This means that when the adapter receives a pull request, it can pull data from other endpoints. For full ISO 15926 compliance, the adapter uses SPARQL as the protocol for data exchange. As shown below, the Interface Service is the SPARQL endpoint. The Adapter Service acts as a SPARQL client to pull data from other SPARQL endpoints. For more information about the [SPARQL protocol](http://www.w3.org/TR/rdf-sparql-protocol/) and [SPARQL query language](http://www.w3.org/TR/rdf-sparql-query/), please see the linked W3C documents.



The adapter can also exchange a Part 7 XML called a Data Transfer Object (DTO). The DTO is a generic object that describes the data using Part 7 Templates. The adapter uses the HTTP protocol to enable RESTful access to the DTO’s. DTO’s are only intended to be exchanged with other Adapter Service endpoints. This is not part of the ISO standard, and is not recommended for interoperability across the internet. However, RESTful access to the DTO’s is designed for testing and debugging purposes. It is also designed to be the easiest and fastest means of data exchange between two adapters using Part 7 templates.

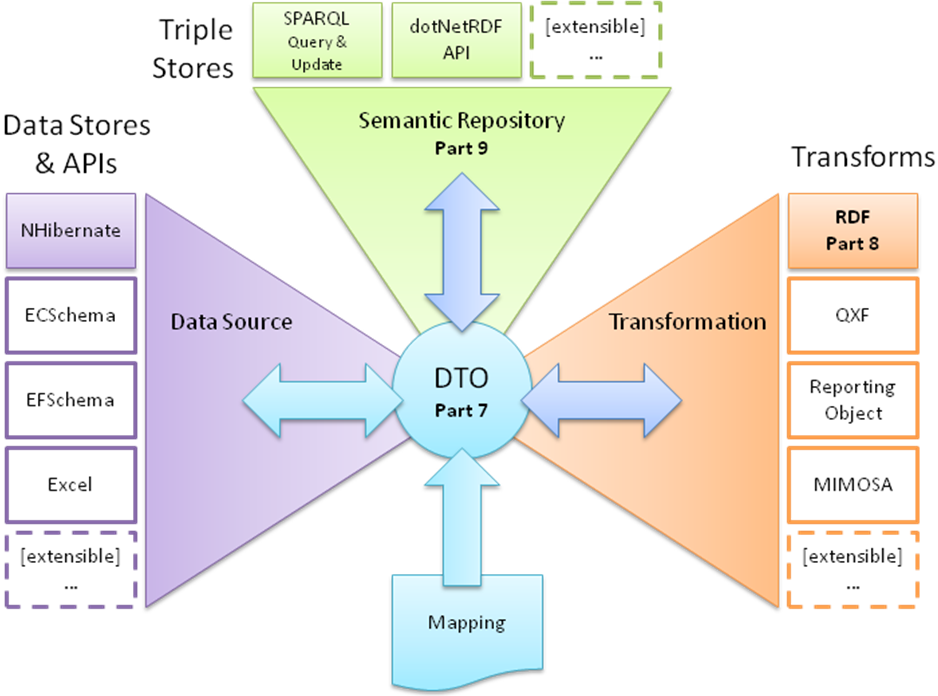


The RESTfull endpoint is very useful as a data service as well. In particular, Ajax clients can easily request and use the XML for their own purposes. Because the use of the adapter as a data service has proven very useful, it also includes a SOAP endpoint for interoperability with BizTalk and SSRS. Further, to enable reporting with SSRS, a less generic, hierarchical version of the DTO is available as well.



* 1. Adapter Framework

The ***iRINGTools*** Adapter is a generic and extensible service. The collection of interfaces and classes used by developers to extend the service is referred to as the Adapter Framework. The framework is centered around the DTO, which is based on the mapping to the ISO 15926 Reference Data. All of the functionality of the adapter deals with persisting, producing, consuming and transforming the DTO.



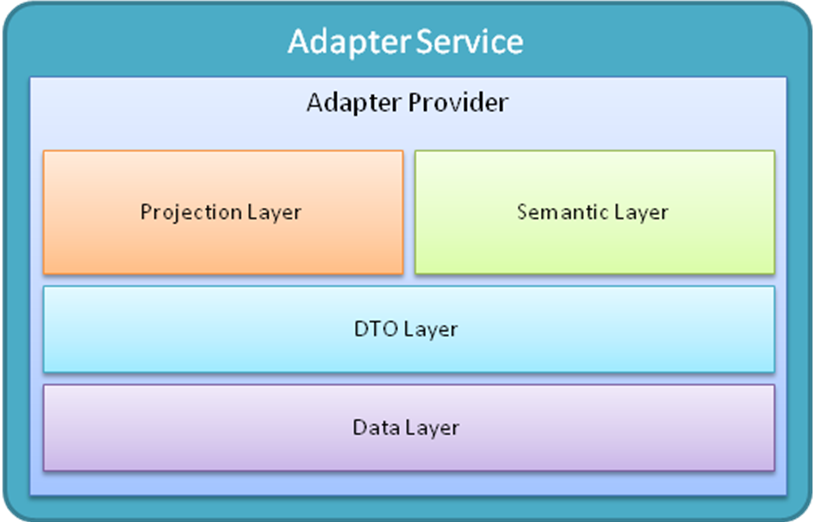
The Adapter Framework provides three primary services:

* Reading from and writing to a data source.
* Querying and writing to a semantic repository.
* Projecting to and consuming different XML representations.
  1. Dependency Injection

This framework is implemented using dependency injection. Dependency injection is a technique for supplying an external dependency (i.e. a reference) to a software component. It is a specific form of inversion of control where the concern being inverted is the process of obtaining the needed dependency. (Wikimedia Foundation Inc., 2010) The dependency injection implementation used in ***iRINGTools*** Adapter is Ninject. For more information about Ninject, go to <http://www.ninject.org>.

* 1. Adapter Layers

The Adapter Framework is broken down into different layers which are separated by interfaces. Each layer has a defined set of functionality that it provides to the adapter.



This version of the guide will focus on the IDataLayer interface. The other layers are extensible as well, and they will be documented in a later version of this guide. The IDataLayer interface consists of methods that must be implemented by each Data Layer. Through this interface, the DTO Layer will produce DTO’s based on your Data Objects and the iRING Mapping. The DTO Layer also uses this interface to create, post, and delete your Data Objects.

This requires you to wrap your custom types in new classes that implement the IDataObject interface. In this way, the DTO Layer is abstracted from knowing anything about your custom types. The DTO Layer will simply use the Data Dictionary you provide to interact with your objects through the interfaces.

1. Custom Data Layer

This section will outline the steps required to create a Custom Data Layer. Creating a Custom Data Layer involves C# development and XML configuration. The process will require the installation of the ***iRINGTools*** Adapter. ***iRINGTools*** Sandbox will not be required as the one installed on the ***iRINGSandbox*** (<http://www.iringsandbox.org>) can be used. A Custom Data Layer consists of a C# Library which implements IDataLayer.

This interface has 4 primary methods for Create, Get, Post and Delete. These methods work with Data Objects that implement IDataObject. The interface also has a GetDictionary method that enables the adapter to get metadata about your entities. Included in the installation of the ***iRINGTools*** Adapter, are libraries that will be should be reference by your Custom Data Layer. These libraries provide the interfaces, and all of the classes used in them.

The steps for creating an implementation of IDataLayer are as follows:

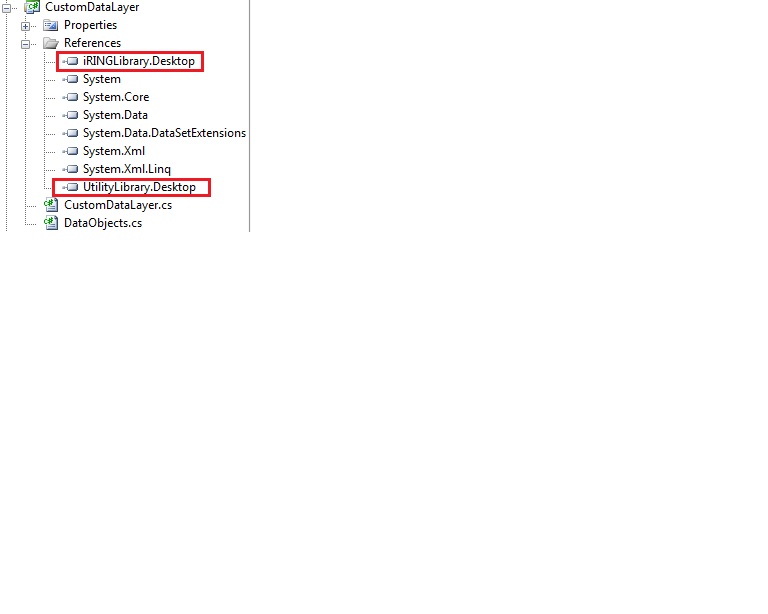
1. Create a C# ClassLibrary Project.
2. Wrap your Entities with Data Objects that implement IDataObject.
3. Create a class that implements IDataLayer.
4. Deploy your Custom Data Layer into the ***iRINGTools*** Adapter.

This section will guide you through each of these steps, using the sample included in the SDK.

* 1. Creating project

Create a new project of type C# ClassLibrary. Then, add a reference to the iRINGLibrary.dll and UtilityLibrary.dll from the Tools folder in the iRINGTools\_Adapter.zip.

The Utility Library is only required if any of the utility methods are to be used. It is probably best to add it now, and decide later whether or not to use the library.



* 1. IDataObject

The IDataObject interface is how the adapter’s DTO Layer interacts with your entities. Developers do not need to modify their application entities in order to implement the interface. It is recommended to wrap application entities with another class that implements the interface. Below is a listing of how this can be done.

public class Equipment

{

public string Tag { get; set; }

public string PumpType { get; set; }

public string PumpDriverType { get; set; }

public Double DesignTemp { get; set; }

public Double DesignPressure { get; set; }

public Double Capacity { get; set; }

public Double SpecificGravity { get; set; }

public Double DifferentialPressure { get; set; }

}

public class EquipmentDataObject : Equipment, IDataObject

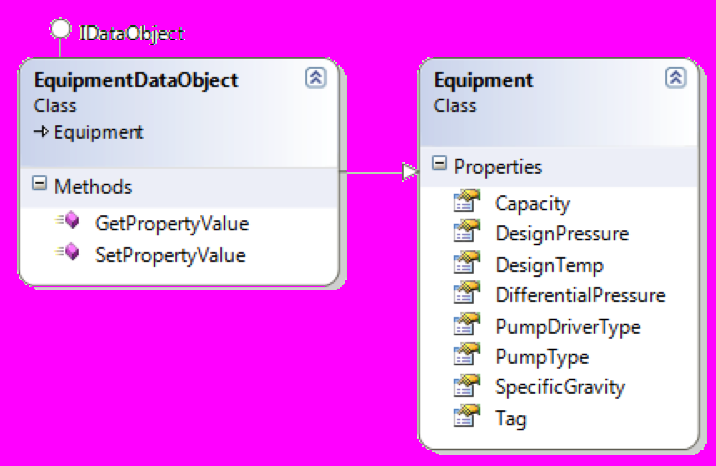
{

public object GetPropertyValue(string propertyName){...}

public void SetPropertyValue(string propertyName, object value){...}

}

Below is a class diagram that shows the listing above.



The interface allows the DTO Layer to delegate the interaction with Entity objects to the Data Layer. Specifically, the DTO Layer uses GetPropertyValue and SetPropertyValue to access the Entity properties. In this way, your implementations of IDataObject do not necessarily need to physically wrap your Entities, but just logically wrap them. This enables the flexibility to build a Data Layer around all sorts of data structures (e.g. concrete, generic, dynamic, persisted, file-based, etc).

* 1. IDataLayer

A Custom Data Layer implements the IDataLayer interface. This section will describe the steps involved in implementing this interface. Each method, and how it is used by the DTO Layer to interact with your Entities, will be explained.

To create a class that properly implements IDataLayer, the following overall steps are recommended.

1. Create Framework Hooks using Ninject.
2. Implement Create.
3. Implement GetDataDictionary.
4. Implement Get, Post and Delete Operations.

The listing below shows the definition of the IDataLayer interface.

public class CustomDataLayer : IDataLayer

{

public IList<IDataObject> Create(  
 string objectType,   
 IList<string> identifiers) {...}

public IList<string> GetIdentifiers(  
 string objectType,   
 DataFilter filter) {...}

public IList<IDataObject> Get(  
 string objectType,   
 DataFilter filter,   
 int pageSize,   
 int pageNumber) {...}

public IList<IDataObject> Get(  
 string objectType,   
 IList<string> identifiers) {...}

public Response Post(  
 IList<IDataObject> dataObjects) {...}

public Response Delete(

string objectType,   
 IList<string> identifiers) {...}

public Response Delete(  
 string objectType,   
 DataFilter filter) {...}

public DataDictionary GetDictionary(){...}

}

* + 1. Framework Hooks

For the Custom Data Layer to be used by the Adapter Framework, certain hooks need to be added to the constructor. Ninject is used to inject dependencies into the constructor of the Custom Data Layer. This is called constructor injection. It is done, but placing the Inject attribute on the constructor. In this way, Ninject will ensure that all of the arguments on the constructor are created before calling the constructor.

[Inject]

public CustomDataLayer(  
 AdapterSettings settings,   
 ApplicationSettings appSettings) {...}

There are other types of objects that may be available from the dependency injection container, but AdapterSettings and ApplicationSettings are the most useful. These dependencies are used in the provided sample. These parameters enable the Custom Dictionary to get the XML path and scope information that is specified by adapter requests.

* + 1. Create Method

The Create method is used by the DTO Layer to get empty objects for the purpose of posting them back to your DataLayer.

Post requests should be processed by your applications business logic, and a detailed response should be returned for each IDataObject. This detailed response should include success, warnings, errors, or a detailed rejection explanation.

The DataDictionary is used to generate the DTOLayer, which does most of the interaction with your DataLayer.

Create the DataLayer by implementing IDataLayer interface present in iringtools.library.

* 1. Deploying the Custom DataLayer

Build the Custom dataLayer and hook its dll in the iRING Adapter. In order for ***iRINGTools*** Adapter to use a custom DataLayer, the custom DataLayer must have the following components. Additional files/components are needed depending on which type of the custom datalayer being used. The details of each component will be described in detail further in this document.

The DataDictionary does not have to be persisted as an XML, developers may choose to create it on the fly or get it from an API.

* 1. Data Dictionary

A data dictionary is class that defines data objects to be exposed to iRINGAdapter and ultimately exchanged with other applications. A data dictionary describes the C# classes exposed by your custom data layer. This information is used by the adapter to generate code that uses your custom data layer and its classes.

A sample of the data dictionary is as follows:

<DataDictionary xmlns=<http://ns.iringtools.org/library>

xmlns:i="http://www.w3.org/2001/XMLSchema-instance">

<dataObjects>

<DataObject>

<dataProperties>

<DataProperty>

<dataLength i:nil="true"/>

<dataType>String</dataType>

<isPropertyKey>true</isPropertyKey>

<isRequired>true</isRequired>

<propertyName>Tag</propertyName>

</DataProperty>

<DataProperty>

<dataLength i:nil="true"/>

<dataType>String</dataType>

<isPropertyKey>false</isPropertyKey>

<isRequired>false</isRequired>

<propertyName>PumpType</propertyName>

</DataProperty>

...

</dataProperties>

<dataRelationships>

<DataRelationship>

<relatedObject/>

<graphProperty/>

<Cardinality/>

</DataRelationship>

</dataRelationships>

<objectName/>

<objectNamespace/>

</DataObject>

...

</dataObjects>

</DataDictionary>

* + 1. Creating a Data Dictionary XML

The data dictionary XML can be created manually or programmatically via DataDictionary.cs class in the iRINGLibrary.dll and serialization method in the UtilityLibrary.dll. To create a data dictionary programmatically, you need add the iRINGLibrary.dll and UtilityLibrary.dll to your project references, create an instance of DataDictionary with data objects and optionally data relationships, and serialize it to disk. For example,

// Create a DataDictionary instance

DataDictionary dataDictionary = new DataDictionary()

{

dataObjects = new List<DataObject>()

{

new DataObject()

{

dataProperties = new List<DataProperty>()

{

new DataProperty()

{

dataLength = "50",

dataType = "String",

isPropertyKey = true,

isRequired = true,

propertyName = "Tag",

},

new DataProperty()

{

dataLength = "255",

dataType = "String",

isPropertyKey = false,

isRequired = false,

propertyName = "Description",

},

},

objectName = "Equipment",

objectNamespace = "Bechtel.CSVDataLayer.API",

}

}

};

It is important to note that each data object must have a key data property by setting the isPropertyKey element to true for iRINGAdapter to identify each individual data object. The key data property can/cannot be similar to primary key column in database terms.

* 1. Custom Data Layer DLL

A custom data layer DLL is a class library that is built from entity classes and an implementation class of the iRINGLibrary interface IDataLayer.cs

* + 1. Entity Classes

Entity classes are programming language classes representing data objects in the data dictionary. Below are examples of Equipment and Instrument entity class:

* + 1. Entity Data Object Classes

Entity Data Object Classes are inherited by Entity classes and implement IDataObjectClass. Below is the example of EquipmentDO Entity Data Object class:

* + 1. The IDataLayer Interface

The IDataLayer interface is packed in iRINGLibrary.dll. It consists of methods that must be implemented by each data layer. Those methods are:

* + 1. IDataLayer implementation class
  1. Binding Configuration
     1. Dependency Injection

Dependency injection is a technique for supplying an external dependency (i.e. a reference) to a software component - that is, indicating to a part of a program which other parts it can use. It is a specific form of inversion of control where the concern being inverted is the process of obtaining the needed dependency 1. The dependency injection implementation used in iRINGAdapter is Ninject. For more information about Ninject, go to [**http://ninject.org/**](http://ninject.org/).

* + 1. Binding Configuration XML

Binding configuration XML contains two bindings DataLayer and DTOService to specify which implementation of the DataLayer and DTOService being used in an application. It is loaded to iRINGAdapter per WCF service request and stored in Ninject session so that any reference to the DataLayer and DTOService will get resolved dynamically at runtime.

In CSVDataLayer, the binding configuration XML is as follows. Note that the second part after the comma in the Interface and Implementation definition are assembly names.

<BindingConfiguration

<Bindings>

<Binding

Name="DataLayer"

Interface="org.iringtools.library.IDataLayer, iRINGLibrary"

Implementation="Bechtel.CSVDataLayer.DataLayer,

Bechtel.CSVDataLayer"/>

<Binding

Name="DTOService"

Interface="org.iringtools.adapter.IDTOService, AdapterLibrary"

Implementation=

"org.iringtools.adapter.proj\_12345\_000.API.DTOService,

App\_Code"/>

</Bindings>

</BindingConfiguration>

* 1. Example Implementation

This document provides an example of a custom data layer used in iRINGTools demonstrations, called CSVDataLayer, which reads comma-separated values (CSV) data from disk, turns it into standard format ( ISO15926 triples) to be exchangeable with other applications, and writes it back to disk as its original CSV format. Below is the data flow between CSVDataLayer and the iRINGAdapter.

The complete CSVDataLayer source code and binary files mentioned in the document are in the Appendix section.

1. Works Cited

Wikimedia Foundation Inc. (2010, April 28). *Dependency injection*. Retrieved April 30, 2010, from Wikipedia: The free encyclopedia: http://en.wikipedia.org/wiki/Dependency\_injection