**WCF - Exposing a DataTable in a Service Operation**

**Aug** 15

Written by:   
8/15/2012 12:34 AM  [RssIcon](http://www.terrymarshall.com.au/Blog/tabid/162/rssid/2/Default.aspx)

So you've decided you to want to provide a list of data in a service operation, but just what are the options available to us for exposing that data to our clients, how can we retrieve the list of data from the database, and what if any custom types do we need to enable our service operation to be consumed by a service client.   
  
Well lets start at the first step....... for our data, we'll consider a list of countries, so lets imagine we have a table in our database that stores countries, and contains the following three columns: -

1 - CountryID  
2 - ISOCode  
3 - CountryName  
  
The following list is an example of how the data in our table might look: -  
  
1   AD  Andorra  
2   AE  United Arab Emirates  
3   AF  Afghanistan  
4   AG  Antigua and Barbuda  
5   AI    Anguilla  
6   AL   Albania  
7   AM   Armenia  
8   AN   Netherlands Antilles  
9   AO   Angola  
10 AQ   Antartica

**Option 1: Return a DataSet (not recommended)**

Whilst it is possible to return a .Net DataTable / DataSet in a service operation (if we can guarantee that all clients are built on the .Net Framework, and therefore avoid any interoperability issues), it's also not a recommended approach. When building a data transfer service we should streamline the service as much as possible and concentrate on providing only the data that we need to send. Unfortunately a .Net DataTable contains a significant amount of information in addition to the actual data, making it an extremely inefficient method for the transfer of data in a service operation. It however, you decided to use this approach, it could be done using a service operation similar to the following: -

[ServiceContract]

**interface** ICountryService

{

    [OperationContract]

    DataSet GetCountries();

}

// Operation implementation

**public** DataSet GetCountries()

{

    DataTable dt = GetTestDataTable();

    DataSet ds = **new** DataSet();

    ds.Tables.Add(dt);

**return** ds;

}

**Option 2: Return a set of custom data objects (recommended)**

This is the most common, and the recommended approach for the transfer of data in a service operation, simply because we define the custom type to represent the data, and as such, we can concentrate on only those data members that we need to provide, ensuring that we can deliver an efficient service over the wire. To do this, we would create a custom type to represent a Country and then create a list of this custom type, with each entry in the list representing a row from our database table.

// Step 1 - Create a custom type to represent a Country

[DataContract]

**class** Country

{

    [DataMember]

**public** **int** CountryID;

    [DataMember]

**public** **string** ISOCode;

    [DataMember]

**public** **string** CountryName;

}

// Step 2 - Define our OperationContract

[ServiceContract]

**interface** ICountryService

{

    [OperationContract]

    List<Country> GetCountryCollection();

}

// Step 3 - Implement the OperationContract

**public** List<Country> GetCountryCollection()

{

    DataTable dt = GetCountriesTable();

    var countries = from row **in** dt.AsEnumerable()

    select **new** Country { CountryID = (**int**)row["CountryID"],

                          ISOCode = (**string**)row["ISOCode"],

        CountryName = (**string**)row["CountryName"]};

**return** countries.ToList<Country>();

}

**Option 3: Return an XML document**

Whilst not the recommended approach, there are times when you may want to transfer data using a naturally interoperable format like XML, which can easily be consumed by non .Net clients. Since a .Net DataTable is fully compatible with XML, and there is functionality built into the DataTable class that converts the data to an XML format, this becomes a very easy process, although not as efficient as option 2.

//returns a DataTable as raw XML content

[ServiceContract]

**interface** ICountryService

{

    [OperationContract]

    XElement GetCountriesAsXML();

}

// Implementation of the GetCountriesAsXML OperationContract

**public** System.Xml.Linq.XElement GetCountriesAsXML()

{

    DataTable dt = GetCountriesTable();

    StringWriter sw = **new** StringWriter();

    dt.WriteXml(sw);

**return** XElement.Parse(sw.ToString());

}