Exploring Strongly Typed DataSet

Source	http://www.dotnetspider.com/kb/Article1110.aspx

XML Schema

Before going in details, lets discuss briefly what is XML Schema.

In short, XML schema describes the structure of XML. It's an alternative to Document Type Definition.

An XML Schema:
- **defines elements that can appear in a document**
- **defines attributes that can appear in a document**
- **defines which elements are child elements**
- **defines the order of child elements**
- **defines the number of child elements**
- **defines whether an element is empty or can include text**
- **defines data types for elements and attributes**
- **defines default and fixed values for elements and attributes**

So, it is quite clear that besides writing raw data to XML, we can also maintain XML schema by defining elements, attributes, and types.
Just the way we define a table, thus validating it while entering data, we can validate a XML document by using XML Schema.

Here is the comparison between XSD type and DOT.NET Framework Type.
(most commonly used Types are listed here)

XML Schema Type >> DOT.NET Framework Type

base64Binary >> System.Byte[]
Boolean >> System.Boolean
Byte >> System.SByte
Date >> System.DateTime
dateTime >> System.DateTime
decimal >> System.Decimal
Double >> System.Double
Float >> System.Single
int >> System.Int32
integer >> System.Decimal
long >> System.Int64
short >> System.Int16
string >> System.String
time >> System.DateTime

Typed DataSet

It is similar to Dataset in DOT.NET only difference is that, the class is derived from Dataset

Class thus inheriting all properties, methods and events. The main advantage over using Typed Dataset over Dataset is that the types are defined within the class itself. So any mismatch of type will generate compile-time error. Also the great feature of Intellisense helps user to view the properties, methods and events of the Typed Dataset by pressing '.'.

Now, it can be confused to many guys, what's the use of XSD or what we are calling XML Schema in creating a typed dataset. You are right, there is no direct linking between the two. Although, you can create a typed dataset by deriving Dataset class and writing tons of code by yourself, here lies the difference. If we know, how to design right XML schema, we can save lot of time as DOT.NET framework provides a great tool in creating typed dataset class automatically using the XML schema provided.

I will discuss in details later, how to use the tool for creating typed dataset.

Designing XSD

Now think of a real-time situation, where a company wants to maintain their customer details having multiple addresses.

We will be creating 2 tables
1. Customer
2. Address

Customer Table Fields
- **ID XSD:int (Unique Constraint defined)**
- **Code XSD:string (Unique Constrain Defined)**
- **Name XSD:string**
- **DOB XSD:DateTime**

Address Table Fields
- **CustomerID XSD:int USE:required**
- **AddressType XSD:string**
- **Addtress1 XSD:string**
- **Address2 XSD:string**
- **City XSD:string**
- **State XSD:string**
- **Country XSD:string attribute => default to "India"**

Our aim is to create a Typed Dataset having two Tables – Customer and Address. Both the tables will contain fields with the structures provided above. A relation - Primary/Foreign referential integrity has been established between Customer and Address

Here is the XML Schema of the above structures,

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="CustomerDataSet" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-
microsoft-com:xml-msdata">
```

```
  <xs:element name="CustomerDataSet" msdata:IsDataSet="true">

     <xs:complexType>
      <xs:choice maxOccurs="unbounded">

        <xs:element name="Customer">
             <xs:complexType>
            <xs:sequence>
                        <xs:element name="ID" type="xs:int" minOccurs="0"/>
                       <xs:element name="Code" type="xs:string"
minOccurs="0"/>
                        <xs:element name="Name" type="xs:string"
minOccurs="0" />
                       <xs:element name="DOB" type="xs:date" minOccurs="0"
/>
                   </xs:sequence>
             </xs:complexType>
        </xs:element>

        <xs:element name="Address">

          <xs:complexType>
            <xs:sequence>
               <xs:element name="CustomerID" type="xs:int" minOccurs="0"/>
               <xs:element name="Address1" type="xs:string" minOccurs="0" />
               <xs:element name="City" type="xs:string" minOccurs="0" />
               <xs:element name="State" type="xs:string" minOccurs="0" />
               <xs:element name="Country" type="xs:string" default="India"
minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>

    <xs:unique name="constraint1">
            <xs:selector xpath="Customer" />
            <xs:field xpath="ID" />
    </xs:unique>

    <xs:unique name="constraint2">
            <xs:selector xpath="Customer" />
            <xs:field xpath="Code" />
    </xs:unique>

   <xs:keyref name="customeraddress" refer="constraint1">
            <xs:selector xpath="Address" />
            <xs:field xpath="CustomerID" />
   </xs:keyref>

  </xs:element>
</xs:schema>
```

If you go through the XML Schema, you will find 1 important addition, default=”India”
This is basically same as [DefaultValue] of [DataColumn]

How to generateTyped Dataset Class from XSD

xsd.exe tool is used to generate Dataset class from XSD.

Usage:
[xsd.exe /d SchemaFileName.xsd /n:Namespace]

{/d directive is used to generate a DataSet.}
{/n: directive is used to generate Namespace for the Dataset.}

To generate a .cs file named "customer.cs" having Namespace "System.Data.Customer", run the code below in command window,

[xsd.exe /d customer.cs /n: System.Data.Customer]

We can then use the customer.cs in our application or we can make a library (.dll), which can be used by any project that has a reference to customer.dll.

To make the library file,
[csc.exe /t:library customer.cs /r:System.dll /r:System.Data.dll]

{/t directive is used to generate a library}
{/n: directive is used to specify the dependent dll of the customer class.}

Well, we have completed most of the part of Typed DataSet. Now we will see some usage of the generated DataSet "Customer"

Sample Code

The code below illustrates Fetching CustomerDataSet from SQL Server Database and using operation Add/Update/Delete using SQLdataAdapter.

SQL script to create CustomerTracker database

```
CREATE DATABASE [CustomerTracker]
GO

use [CustomerTracker]
GO

CREATE TABLE [dbo].[Customer] (
        [ID] [int] NOT NULL ,
        Code [varchar] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Name] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [DOB] [datetime] NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Customer] ADD
        CONSTRAINT [PK_Customer] PRIMARY KEY  CLUSTERED
        (
```

```
                [ID]
        )  ON [PRIMARY]
GO

CREATE TABLE [dbo].[Address] (
        [CustomerID] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
,
        [AddressType] [varchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
,
        [Address1] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Address2] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [City] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [State] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Country] [varchar] (30) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
```

C# Code

```
using System.Data.Customer;

        CustomerDataSet ds;
        SqlConnection con;
        SqlDataAdapter AdptCust;
        SqlDataAdapter AdptAddr;

        private void FetchDataSet()
        {
                ds=new CustomerDataSet();
                con=new SqlConnection();
                con.ConnectionString="workstation id=<SERVER_NAME>;user
id=sa;data source='(local)';initial catalog=CustomerTracker";

                AdptCust=new SqlDataAdapter("Select * from Customer", con);
                AdptAddr=new SqlDataAdapter("Select * from Address", con);

                AdptCust.Fill(ds, "Customer");
                AdptAddr.Fill(ds, "Address");

                con.Close();
        }

        private void GetCustomerData()
        {
                foreach(CustomerDataSet.CustomerRow dr in ds.Customer.Rows)
                {
                        MessageBox.Show(dr["ID"].ToString());
                        MessageBox.Show(dr["Code"].ToString());
                        MessageBox.Show(dr["Name"].ToString());
                        MessageBox.Show(dr["DOB"].ToString());
                }
        }

        private void AddCustomer(int ID, string Code, string Name, DateTime
```

```
DOB)
        {
                if(AdptCust==null)
                {
                        FetchDataSet();
                }

                AdptCust.InsertCommand=con.CreateCommand();
                AdptCust.InsertCommand.CommandText="Insert Into Customer(ID,
Code, Name, DOB) Values (@ID, @Code, @Name, @DOB)";
                AdptCust.InsertCommand.Parameters.Add(new SqlParameter("@ID",
SqlDbType.Int, 4, "ID"));
                AdptCust.InsertCommand.Parameters.Add(new
SqlParameter("@Code", SqlDbType.VarChar, 10, "Code"));
                AdptCust.InsertCommand.Parameters.Add(new
SqlParameter("@Name", SqlDbType.VarChar, 50, "Name"));
                AdptCust.InsertCommand.Parameters.Add(new SqlParameter("@DOB",
SqlDbType.DateTime, 8, "DOB"));

                CustomerDataSet.CustomerRow drNew =
ds.Customer.NewCustomerRow();
                drNew["ID"] = ID;
                drNew["Code"] = Code;
                drNew["Name"] = Name;
                drNew["DOB"] = DOB;
                ds.Customer.Rows.Add(drNew);

                try
                {
                        AdptCust.Update(ds, "Customer");
                        ds.AcceptChanges();
                }
                catch(SqlException ex)
                {
                        Console.WriteLine(ex.Message);
                }
                finally
                {
                }
        }

        private void UpdateCustomer(int ID, string Code, string Name, DateTime
DOB)
        {
                if(AdptCust==null)
                {
                        FetchDataSet();
                }

                AdptCust.UpdateCommand=con.CreateCommand();
                AdptCust.UpdateCommand.CommandText="Update Customer Set
Code=@Code, Name=@Name, DOB=@DOB Where ID=@ID";
                AdptCust.UpdateCommand.Parameters.Add(new SqlParameter("@ID",
SqlDbType.Int, 4, "ID"));
                AdptCust.UpdateCommand.Parameters.Add(new
SqlParameter("@Code", SqlDbType.VarChar, 10, "Code"));
```

```
                AdptCust.UpdateCommand.Parameters.Add(new
SqlParameter("@Name", SqlDbType.VarChar, 50, "Name"));
                AdptCust.UpdateCommand.Parameters.Add(new SqlParameter("@DOB",
SqlDbType.DateTime, 8, "DOB"));
                DataRow[] dr= ds.Customer.Select("ID=" + ID.ToString());
                CustomerDataSet.CustomerRow drUpdate=null;
                if (dr.Length>0)
                {
                        drUpdate = (CustomerDataSet.CustomerRow) dr[0];
                }
                else
                {
                        MessageBox.Show("No such ID found.");
                        return;
                }

                drUpdate["Code"] = Code;
                drUpdate["Name"] = Name;
                drUpdate["DOB"] = DOB;
                try
                {
                        AdptCust.Update(ds, "Customer");
                        ds.AcceptChanges();
                }
                catch(SqlException ex)
                {
                        Console.WriteLine(ex.Message);
                }
                finally
                {
                }
        }

        private void DeleteCustomer(int ID)
        {
                if(ad==null)
                {
                        FetchDataSet();
                }

                ad.DeleteCommand=con.CreateCommand();
                ad.DeleteCommand.CommandText="Delete from Customer Where
ID=@ID";
                ad.DeleteCommand.Parameters.Add(new SqlParameter("@ID",
SqlDbType.Int, 4, "ID"));

                DataRow[] dr= ds.Customer.Select("ID=" + ID.ToString());
                if (dr.Length>0)
                {
                        dr[0].Delete();
                }
                else
                {
                        MessageBox.Show("No such ID found in the database.");
                        return;
                }
```

```
            try
            {
                    ad.Update(ds, "Customer");
                    ds.AcceptChanges();
            }
            catch(SqlException ex)
            {
                    Console.WriteLine(ex.Message);
            }
            finally
            {
            }
    }
```

~~~ End of Article ~~~