Go

C# : DataTable

.NET **Array** **Dictionary** **List** **String** 2D Async DataTable Dates DateTime Enum File For Foreach Format IEnumerable If IndexOf Lambda LINQ Parse Path Process Property Random Regex Replace Sort Split Static Substring Switch Tuple While

DataSet. This is a collection of DataTables. We use the DataSet type to store many DataTables in a single collection. Conceptually, the DataSet acts as a set of DataTable instances.

Usage. DataSet simplifies programs that use many DataTables. To effectively use the DataSet, you will need to have some DataTables handy. We begin by creating some.

Create. In this program, we create two DataTables. One stores two rows of patient information. And the second stores two rows of medication information.

Next:

We create a DataSet with the DataSet constructor. Then we add the two DataTables to the DataSet instance. Finally we print the XML.

Based on: .NET 4.6

C# program that uses DataSet

```
using System;
using System.Data;

class Program
{
    static void Main()
    {
        // Create two DataTable instances.
        DataTable table1 = new DataTable("patients");
        table1.Columns.Add("name");
        table1.Columns.Add("id");
        table1.Rows.Add("sam", 1);
        table1.Rows.Add("mark", 2);
```

```
        DataTable table2 = new DataTable("medications");
        table2.Columns.Add("id");
        table2.Columns.Add("medication");
        table2.Rows.Add(1, "atenolol");
        table2.Rows.Add(2, "amoxicillin");

        // Create a DataSet and put both tables in it.
        DataSet set = new DataSet("office");
        set.Tables.Add(table1);
        set.Tables.Add(table2);

        // Visualize DataSet.
        Console.WriteLine(set.GetXml());
    }
}
```

Output

```
<office>
  <patients>
    <name>sam</name>
    <id>1</id>
  </patients>
  <patients>
    <name>mark</name>
    <id>2</id>
  </patients>
  <medications>
    <id>1</id>
    <medication>atenolol</medication>
  </medications>
  <medications>
    <id>2</id>
    <medication>amoxicillin</medication>
  </medications>
</office>
```

Dispose, using DataSet. You can put your DataSet in a using block. This ensures the Dispose method is called as soon as possible when the DataSet is no longer being used.

Tip:

If you are having resource usage problems, adding using blocks can help. This is a good first step to fixing such issues.

Using

C# program that creates DataSet in using block

```
using System.Data;

class Program
{
    static void Main()
    {
        // Create a DataSet in using statement.
        using (DataSet set = new DataSet("office"))
        {
            // Put code that adds stuff to DataSet here.
            // ... The DataSet will be cleaned up outside the block.
        }
    }
}
```

Namespace, Prefix. One important use of the DataSet is to encode data in XML format. Often, XML data needs to have an XML namespace of a tag element prefix.

And:
Fortunately, the DataSet provides the Namespace and Prefix properties to specify this.

Next:
This example specifies both the Namespace and the Prefix, and you can see them appear in the output.

C# program that uses Namespace and Prefix

```
using System;
using System.Data;

class Program
{
    static void Main()
    {
        DataTable table1 = new DataTable("patients");
        table1.Columns.Add("name");
        table1.Columns.Add("id");
        table1.Rows.Add("sam", 1);
```

```
        // Create a DataSet.
        DataSet set = new DataSet("office");
        set.Tables.Add(table1);
        set.Namespace = "y";
        set.Prefix = "x";

        // Visualize DataSet.
        Console.WriteLine(set.GetXml());
    }
}
```

Output

```
<x:office xmlns:x="y">
  <patients xmlns="y">
    <name>sam</name>
    <id>1</id>
  </patients>
</x:office>
```

DataSetName. Every DataSet can have a name specified. Usually, it is easiest to specify this inside the DataSet constructor. But you don't always know the name at this time.

However:

You can also change the name by assigning to the DataSetName property. You can also read the DataSetName property.

C# program that uses DataSetName

```
using System;
using System.Data;

class Program
{
    static void Main()
    {
        // Create a DataSet.
        DataSet set = new DataSet("office");

        // Show original name.
        Console.WriteLine(set.DataSetName);

        // Change its name.
```

```
        set.DataSetName = "unknown";
        Console.WriteLine(set.DataSetName);
    }
}
```

Output

```
office
unknown
```

Copy, Clear. DataSet has a Clear method that clears all the DataTables in the set. It also provides a Copy method that will make a deep copy of all the DataTables in the set.

Tip:

If you call Copy and the Clear the original, your copied data will still exist unchanged.

C# program that uses Copy and Clear

```
using System;
using System.Data;

class Program
{
    static void Main()
    {
        DataTable table1 = new DataTable("patients");
        table1.Columns.Add("name");
        table1.Columns.Add("id");
        table1.Rows.Add("sam", 1);

        DataTable table2 = new DataTable("medications");
        table2.Columns.Add("id");
        table2.Columns.Add("medication");
        table2.Rows.Add(1, "atenolol");

        // Create a DataSet.
        DataSet set = new DataSet("office");
        set.Tables.Add(table1);
        set.Tables.Add(table2);

        // Copy the DataSet.
        DataSet copy = set.Copy();

        // Clear the first DataSet.
        set.Clear();
```

```
        // Show contents.
        Console.WriteLine("set: {0}", set.GetXml());
        Console.WriteLine("copy: {0}", copy.GetXml());
    }
}
```

Output

```
set: <office />
copy: <office>
  <patients>
    <name>sam</name>
    <id>1</id>
  </patients>
  <medications>
    <id>1</id>
    <medication>atenolol</medication>
  </medications>
</office>
```

Tables. This property returns an instance of a DataTableCollection. You can use the Count property and indexer on this sub-collection to access all the individual tables.

Indexer

Also:

You can get a DataTable from the Tables collection with its name. We use set.Tables["medications"] to show this behavior.

C# program that uses Tables and DataTableCollection

```
using System;
using System.Data;

class Program
{
    static void Main()
    {
        DataTable table1 = new DataTable("patients");
        table1.Columns.Add("name");
        table1.Columns.Add("id");
        table1.Rows.Add("sam", 1);

        DataTable table2 = new DataTable("medications");
```

```
        table2.Columns.Add("id");
        table2.Columns.Add("medication");
        table2.Rows.Add(1, "atenolol");
        table2.Rows.Add(6, "trifluoperazine");

        // Create a DataSet.
        DataSet set = new DataSet("office");
        set.Tables.Add(table1);
        set.Tables.Add(table2);

        // Loop over DataTables in DataSet.
        DataTableCollection collection = set.Tables;
        for (int i = 0; i < collection.Count; i++)
        {
            DataTable table = collection[i];
            Console.WriteLine("{0}: {1}", i, table.TableName);
        }

        // Write name of first table.
        Console.WriteLine("x: {0}", set.Tables[0].TableName);

        // Write row count of medications table.
        Console.WriteLine("y: {0}", set.Tables["medications"].Rows.Count);
    }
}
```

Output

```
0: patients
1: medications
x: patients
y: 2
```

Relations. The Relations property is a way to specify many DataRelations on the DataTables. A DataRelation indicates which tables are dependent on other tables (sub-tables).

CaseSensitive. The DataSet provides the CaseSensitive property. The default value is False. There may be cases where you want string lookups on your DataSet to be case-sensitive.

Note:

One case involves two elements with the same name but different character casing ("Medications", "medications").

GetXml. It is possible to convert a DataSet to a string representation in XML syntax. The GetXml() method on the DataSet instance is ideal for this.

DataSet:
The example program constructs a new DataSet instance with the name "Hospital".

Then:
It adds a new DataTable to this set. This DataTable has four rows and five columns.

Finally:
The GetXml instance method is invoked on the DataSet, and the result is printed to the screen.

C# that uses GetXml method

```
using System;
using System.Data;

class Program
{
    static DataTable Table()
    {
        DataTable table = new DataTable("Prescription");
        table.Columns.Add("Dosage", typeof(int));
        table.Columns.Add("Drug", typeof(string));
        table.Columns.Add("Patient", typeof(string));
        table.Columns.Add("Date", typeof(DateTime));

        table.Rows.Add(25, "Indocin", "David", DateTime.Now);
        table.Rows.Add(50, "Enebrel", "Sam", DateTime.Now);
        table.Rows.Add(10, "Hydralazine", "Christoff", DateTime.Now);
        table.Rows.Add(21, "Combivent", "Janet", DateTime.Now);
        table.Rows.Add(100, "Dilantin", "Melanie", DateTime.Now);
        return table;
    }

    static void Main()
    {
        // Create DataSet instance.
        DataSet set = new DataSet("Hospital");
        // Add new table.
```

```
        set.Tables.Add(Table());
        // Write xml data.
        Console.WriteLine(set.GetXml());
    }
}
```

Output

```
<Hospital>
  <Prescription>
    <Dosage>25</Dosage>
    <Drug>Indocin</Drug>
    <Patient>David</Patient>
    <Date>2015-06-17T08:39:41.0879713-06:00</Date>
  </Prescription>
  <Prescription>
    <Dosage>50</Dosage>
    <Drug>Enebrel</Drug>
    <Patient>Sam</Patient>
    <Date>2015-06-17T08:39:41.0879713-06:00</Date>
  </Prescription>
  <Prescription>
    <Dosage>10</Dosage>
    <Drug>Hydralazine</Drug>
    <Patient>Christoff</Patient>
    <Date>2015-06-17T08:39:41.0879713-06:00</Date>
  </Prescription>
  <Prescription>
    <Dosage>21</Dosage>
    <Drug>Combivent</Drug>
    <Patient>Janet</Patient>
    <Date>2015-06-17T08:39:41.0879713-06:00</Date>
  </Prescription>
  <Prescription>
    <Dosage>100</Dosage>
    <Drug>Dilantin</Drug>
    <Patient>Melanie</Patient>
    <Date>2015-06-17T08:39:41.0879713-06:00</Date>
  </Prescription>
</Hospital>
```

XML notes. In the XML, the element names Hospital, Prescription, Dosage, Drug, Patient and Date correspond to the name of the DataSet, the name of the DataTable, and the four DataColumns.

DataColumn

GetXmlSchema. An XML schema indicates the structure of an XML document. GetXmlSchema() on the DataSet type generates an XML schema from the known structure encoded in your DataSet.

Then:

We create a DataSet and then add a DataTable instance to it. We call the GetXmlSchema instance method, which reveals the XML schema.

C# that demonstrates GetXmlSchema

```
using System;
using System.Data;

class Program
{
    static DataTable Table()
    {
        DataTable table = new DataTable("Prescription");
        table.Columns.Add("Dosage", typeof(int));
        table.Columns.Add("Drug", typeof(string));
        table.Columns.Add("Patient", typeof(string));
        table.Columns.Add("Date", typeof(DateTime));

        table.Rows.Add(25, "Indocin", "David", DateTime.Now);
        table.Rows.Add(50, "Enebrel", "Sam", DateTime.Now);
        table.Rows.Add(10, "Hydralazine", "Christoff", DateTime.Now);
        table.Rows.Add(21, "Combivent", "Janet", DateTime.Now);
        table.Rows.Add(100, "Dilantin", "Melanie", DateTime.Now);
        return table;
    }

    static void Main()
    {
        // Create DataSet instance.
        DataSet set = new DataSet("Hospital");
        // Add new table.
        set.Tables.Add(Table());
        // Write xml schema data.
        Console.WriteLine(set.GetXmlSchema());
    }
}
```

Output: edited

```
<?xml version="1.0" encoding="utf-16"?>
<xs:schema id="Hospital" xmlns="" xmlns:xs="" xmlns:msdata="">
```

```
  <xs:element name="Hospital" msdata:IsDataSet="true"
msdata:UseCurrentLocale="true">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element name="Prescription">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Dosage" type="xs:int" minOccurs="0" />
              <xs:element name="Drug" type="xs:string" minOccurs="0" />
              <xs:element name="Patient" type="xs:string" minOccurs="0" />
              <xs:element name="Date" type="xs:dateTime" minOccurs="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML schema notes. What can we do with an XML schema? The DataSet type provides two methods, InferXmlSchema and ReadXmlSchema, which can load a file we specify that contains the schema data.

Then:

We have a DataSet with all the appropriate constraints in it. After this, we could use ReadXml on an XML file of the data itself.

A summary. DataSet collects many DataTables inside a single collection. It provides useful helper methods for acting upon those DataTables. Thus DataSet introduces a key abstraction.