**C# DataTable**

The DataTable type is a convenient and powerful way to store data in memory. We get a DataTable with four columns of **type int, string and DateTime**.

**Program that uses DataTable: C#**

using System;

using System.Data;

class Program

{

static void Main()

{

// Get the DataTable.

DataTable table = GetTable();

// Use DataTable here with SQL.

}

/// <summary>

/// This example method generates a DataTable.

/// </summary>

static DataTable GetTable()

{

// Here we create a DataTable with four columns.

DataTable table = new DataTable();

table.Columns.Add("Dosage", typeof(int));

table.Columns.Add("Drug", typeof(string));

table.Columns.Add("Patient", typeof(string));

table.Columns.Add("Date", typeof(DateTime));

// Here we add five DataRows.

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;

}

}

**GetTable** instantiates a new DataTable reference, adds four column collections to it, and then adds five drug and patient records. The next step to using this code could be to assign the DataSource to a Windows Forms control.

**Example 2**

The next example here shows how to insert data from object collections such as List into a DataTable.

**Why use DataTable here?** It makes your DataGridView simpler and easier. You could manually add data to the DataGridView using the Add method. It is better to put the logic in a separate class.

**Caution:**DataGridView has performance problems with manually adding rows. Using DataTable instead alleviates these.

**How to use a DataGridView.** Make a new Windows Forms project and add a DataGridView to it. The control will be named dataGridView1 automatically. Its purpose will be rendering the DataTable you will make.

**And:**You need some actual data for the example. You will have something important—just use that.

**Getting started.** We want to make a new class and a method that returns DataTable. This method will return the full DataTable. In testing, the application handled one million numbers with only moderate slowdowns.

**Program that uses DataTable with DataGridView: C#**

using System.Collections.Generic;

using System.Data;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

/// Contains column names.

/// </summary>

List<string> \_names = new List<string>();

/// Contains column data arrays.

/// </summary>

List<double[]> \_dataArray = new List<double[]>();

public Form1()

{

InitializeComponent();

// Example column.

\_names.Add("Cat");

// Three numbers of cat data

\_dataArray.Add(new double[]

{

1.0,

2.2,

3.4

});

// Another example column

\_names.Add("Dog");

// Add three numbers of dog data

\_dataArray.Add(new double[]

{

3.3,

5.0,

7.0

});

// Render the DataGridView.

dataGridView1.DataSource = GetResultsTable();

}

/// This method builds a DataTable of the data.

/// </summary>

public DataTable GetResultsTable()

{

// Create the output table.

DataTable d = new DataTable();

// Loop through all process names.

for (int i = 0; i < this.\_dataArray.Count; i++)

{

// The current process name.

string name = this.\_names[i];

// Add the program name to our columns.

d.Columns.Add(name);

// Add all of the memory numbers to an object list.

List<object> objectNumbers = new List<object>();

// Put every column's numbers in this List.

foreach (double number in this.\_dataArray[i])

{

objectNumbers.Add((object)number);

}

// Keep adding rows until we have enough.

while (d.Rows.Count < objectNumbers.Count)

{

d.Rows.Add();

}

// Add each item to the cells in the column.

for (int a = 0; a < objectNumbers.Count; a++)

{

d.Rows[a][i] = objectNumbers[a];

}

}

return d;

}

}

}

**Description**

The two arrays are initialized in the class and in the constructor. They contain column information.

**In the above code,** we create a new DataTable. This is populated with data and put into the DataGrid. There are more efficient ways of modifying existing DataTables. By changing an existing one, you could avoid excess allocations.

**Loop through columns.** We have a collection that contains many arrays. Each of those arrays needs to be a new column. So the main for-loop in the above code goes through the data we want to put in each column.

**Tip:**This site has more information about looping through the rows and cells in the DataTable type.

[**DataTable Foreach Loop**](http://www.dotnetperls.com/datatable-foreach)

**Add columns by name.** We add the column names to the DataTable. This can be the data series type, such as "name", "number" or "id". These are the column headers, which must be added with Columns.Add.

**Tip:**DataTable requires object arrays to assign to the cells. Object arrays hold any derived type.

**We keep adding rows** until we have enough to contain all the data points in our array. If we don't add empty rows, the runtime will throw an exception. Right after this step, we assign the cells.

**DataSource**

Here we assign the DataSource of a DataGridView directly to the result value. We use the built-in display logic. For Windows Forms, DataSource provides better display performance than trying to individually add elements.

**Example code that sets DataSource: C#**

//

// Draw new cells on DataGridView.

dataGridView1.DataSource = null;

dataGridView1.DataSource = GetResultsTable();

**Convert List**

The List type is often useful. The DataTable type is better for when you are interacting with controls such as the DataGridView or with SQL databases. It is sometimes a good idea to convert your List into a DataTable.

[**Convert List, DataTable**](http://www.dotnetperls.com/convert-list-datatable)

**DataRow, DataColumn**



The two most important parts of DataTable are its Rows collection and its Columns collection. Use the instance Add method to add to either of these collections. We look also into the types' internal representations.

**Tip:**There is more information, including many examples, of using DataRow and DataColumn collections on this site.

[**DataRow**](http://www.dotnetperls.com/datarow)[**DataColumn**](http://www.dotnetperls.com/datacolumn)

**DataSet:**You can combine multiple DataTable instances into one collection by using the DataSet type. This provides useful functionality.

[**DataSet Examples**](http://www.dotnetperls.com/dataset)

**DataView:**One useful way to manipulate your DataTable's representation is to sort it using DataView. The DataView has a Sort property.

[**DataView Examples**](http://www.dotnetperls.com/dataview)

**Select**

You can use the Select method on the DataTable itself to filter elements based on some criteria. You receive an array of DataRow instances that match your filter. The filter is specified in string format.

[**DataTable Select**](http://www.dotnetperls.com/datatable-select)

**Using**

DataTable can be placed in a using statement. But is this worthwhile? Often the using block construct helps improve resource management in programs. We show the using statement wrapping a DataTable.

**Tip:**You can add Columns and Rows to the DataTable instance inside, but not outside, the using block.

using System;

using System.Data;

class Program

{

static void Main()

{

**using** (DataTable table = new DataTable())

{

table.Columns.Add("Name", typeof(string));

table.Columns.Add("Date", typeof(DateTime));

table.Rows.Add("cat", DateTime.Now);

table.Rows.Add("dog", DateTime.Today);

}

}

}

**In this example,** the Dispose method is invoked. After the using statement, the Dispose method on DataTable is called. This method is implemented on the base class for DataTable: MarshalValueByComponent.

[**Using**](http://www.dotnetperls.com/using)

**Worthwhile?** Is enclosing the DataTable inside a using statement worthwhile? When Dispose is called, native resources from MarshalValueByComponent (a base class of DataTable) are released. This may be helpful.

**Therefore:**It is possible that the using statement could alleviate some resource usage problems.

**Also,** it is probably not important, but the using statement is good coding hygiene for DataTables. It is recommended if your program has any possible memory or resource usage issues. Some resources may be abandoned without it.

**Summary**

We used DataTable in several C# programs. To it we added data, in the form of columns and rows. This code is effective when storing more than one million data points and rendering them every three seconds. It is efficient.

**Tip:**DataTable is faster in Windows Forms than manually adding rows. It reduces complexity.