**Convert List, DataTable.** The DataGridView control can display a List. This can be done by converting the List to a DataTable first.

**Some notes.**With some custom List-to-DataTable conversion logic, we get the best performance from the DataGridView by using the DataSource property.

**An example.**This example shows a List of string arrays. This is a data structure that could be parsed from a text file easily, as with the Split method.

**Method:** We introduce the ConvertListToDataTable method, which converts a List<string[]> to a DataTable.

[**Arrays**](https://www.dotnetperls.com/array)

**Here:** We create a new empty DataTable. Next, we find the maximum number of columns that will be required by looping through the input List.

**Then:**We add those columns, and then add all the rows. We return the DataTable.

**Based on:** .NET (2019)

**C# program that converts List to DataTable for DataGridView**

using System;

using System.Collections.Generic;

using System.Data;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

// Example list.

List<string[]> list = new List<string[]>();

list.Add(new string[] { "Column 1", "Column 2", "Column 3" });

list.Add(new string[] { "Row 2", "Row 2" });

list.Add(new string[] { "Row 3" });

// Convert to DataTable.

**DataTable** table = ConvertListToDataTable(list);

dataGridView1.**DataSource** = table;

}

static DataTable ConvertListToDataTable(List<string[]> list)

{

// New table.

**DataTable** table = new DataTable();

// Get max columns.

int columns = 0;

foreach (var array in list)

{

if (array.Length > columns)

{

columns = array.Length;

}

}

// Add columns.

for (int i = 0; i < columns; i++)

{

table.Columns.Add();

}

// Add rows.

foreach (var array in list)

{

table.Rows.Add(array);

}

return table;

}

}

}

**Assign to DataSource.** Once we have a DataTable instance, it is appropriate to assign it to the DataSource property. Assigning the List<string[]> to the DataSource will not work correctly. [**DataSource**](https://www.dotnetperls.com/datasource)

**Note:** The DataSource provides the best performance for displaying large amounts of data instantly in a DataGridView.

**Warning:** If you were to add Columns and Rows individually, slowdowns are more likely to occur.

**Discussion.**DataTable can be used as an optimization. Using controls in Windows Forms or WPF directly often results in rendering.

**However:** This is often much slower, and more processor-intensive, than in-memory operations.

**So:** When using DataGridView or similar controls, processing the data first seems like it might be slower, but it ends up much faster.

[**DataGridView**](https://www.dotnetperls.com/datagridview)

**A summary.**We converted a List of string arrays into a DataTable. This yielded a collection that can be displayed in a DataGridView with the DataSource property. Our code was effective. [**List**](https://www.dotnetperls.com/list)[**DataTable**](https://www.dotnetperls.com/datatable)

**You could imperatively**add rows and columns to the DataGridView. But this can be slower. Building a collection in-memory with a DataTable is often more effective.

**Convert List, string.** Think of a sentence. It contains some words. We could represent a sentence as a single string—one with spaces. But a list of strings (of words) is sometimes better.

**We can convert**our string into a List of smaller strings. For the reverse, we can convert our List into a string. This is possible with the ToArray method on the List type. [**Conversions**](https://www.dotnetperls.com/cast)

**First example.**We use the string. Join method to combine a List of strings into one string. The output can be used as a CSV record. On new .NET Framework versions, ToArray is not required. [**Join**](https://www.dotnetperls.com/string-join)

**However:** In previous versions, we had to call ToArray on a List before using Join. In older programs this is still required.

**Based on:** .NET (2019)

**C# program that converts List**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

List<string> dogs = new List<string>();

dogs.Add("Aigi"); // Add string 1

dogs.Add("Spitz"); // 2

dogs.Add("Mastiff"); // 3

dogs.Add("Finnish Spitz"); // 4

dogs.Add("Briard"); // 5

string dogCsv = **string.Join**(",", dogs.ToArray());

Console.WriteLine(dogCsv);

}

}

**Output**

Aigi,Spitz,Mastiff,Finnish Spitz,Briard

**Example 2.**Here we use the StringBuilder class to convert a List to a single string. Note that you can convert a List of any object type into a string this way. **[StringBuilder](https://www.dotnetperls.com/stringbuilder)**

**Final delimiter:** The example has a final delimiter on the end. This is not present in code that uses string.Join. It can be inconvenient.

**TrimEnd:** Sometimes, it is good to remove the end delimiter with TrimEnd. Other times it is best left alone.

[**TrimEnd, TrimStart**](https://www.dotnetperls.com/trimend)

**C# program that uses List and StringBuilder**

using System;

using System.Collections.Generic;

using System.Text;

class Program

{

static void Main()

{

List<string> cats = new List<string>(); // Create new list of strings

cats.Add("Devon Rex"); // Add string 1

cats.Add("Manx"); // 2

cats.Add("Munchkin"); // 3

cats.Add("American Curl"); // 4

cats.Add("German Rex"); // 5

**StringBuilder** builder = new StringBuilder();

foreach (string cat in cats) // Loop through all strings

{

builder.Append(cat).Append("|"); // Append string to StringBuilder

}

string result = builder.**ToString**(); // Get string from StringBuilder

Console.WriteLine(result);

}

}

**Output**

Devon Rex|Manx|Munchkin|American Curl|German Rex|

**Example 3.**Here we convert a List of ints into a single string. The StringBuilder's Append method receives a variety of types. We can simply pass it the int.

**And:**Append() will handle the int on its own. It will convert it to a string and append it.

**Performance:** StringBuilder is fast for most programs. More speed could be acquired by using a char[] and then converting to a string.

[**Char Array**](https://www.dotnetperls.com/char-array)

**C# program that converts List types**

using System;

using System.Collections.Generic;

using System.Text;

class Program

{

static void Main()

{

List<int> safePrimes = new List<int>(); // Create list of ints

safePrimes.Add(5); // Element 1

safePrimes.Add(7); // Element 2

safePrimes.Add(11); // Element 3

safePrimes.Add(23); // Element 4

**StringBuilder** builder = new StringBuilder();

foreach (int safePrime in safePrimes)

{

// Append each int to the StringBuilder overload.

builder.Append(safePrime).Append(" ");

}

string result = builder.**ToString**();

Console.WriteLine(result);

}

}

**Output**

5 7 11 23

**Example 4.**Finally, we get a List of strings from a string in CSV format. This requires the Split method. If you require per-item conversion, loop over the string array returned by Split.

**C# program that converts string to List**

using System;

using System.Collections.Generic;

class Program

{

static void Main()

{

string csv = "one,two,three"; // The input string

string[] parts = csv.**Split**(','); // Call Split method

List<string> list = new List<string>(parts); // Use List constructor

foreach (string item in list)

{

Console.WriteLine(item);

}

}

}

**Output**

one

two

three

**A summary.**We converted Lists and strings using the string.Join methods and the StringBuilder approach. The List is easily concatenated and stored in a database or file with these methods.[**List**](https://www.dotnetperls.com/list)[**Strings**](https://www.dotnetperls.com/string)

[**Dot Net Perls**](https://www.dotnetperls.com/)

**DataTable.**A crystal has many sides. In a similar sense, computer data has many rows and columns. The DataTable class stores rows and columns of data.

**DataTable is part**of the System. Data namespace. We add, select and iterate over stored data. The foreach loop can be used on the Rows in a DataTable.

**First program.**Data can come from a database, from a method, from memory. Here, GetTable generates a table with 4 columns of different types.

**Then:** The table could be persisted to disk, displayed, or stored in memory. It is like any other object.

**GetTable:** This method creates a new DataTable reference. It adds 4 column collections, then 5 rows.

**Typeof:**We must specify a type (as with typeof) to create a Column. All fields from rows in this column must have this type.

**Based on:** .NET (2019)

**C# program that uses DataTable**

using System;

using System.Data;

class Program

{

static void Main()

{

// Get the DataTable.

**DataTable** table = GetTable();

// ... Use the 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;

}

}

**Foreach-loop.**Often we want to loop over our DataTable rows. This can be done with a foreach-loop. The exact syntax can be confusing, and objects are sometimes needed. [**Foreach**](https://www.dotnetperls.com/datatable-foreach)

**Compare rows.**Data may sometimes contain duplicate rows. This condition can be detected with a simple looping algorithm. The syntax can be used for many DataTable tasks. [**Compare Rows**](https://www.dotnetperls.com/datatable-compare-rows)

**Rows, field.** We build in complexity—here we loop over all Rows of our DataTable. We then access the first field, with the Field extension method, as an int.

**Generic method:** Field is a generic method. So we must specify its parametric type (here, int) to indicate its behavior.

[**Field**](https://www.dotnetperls.com/datarow-field)

**Rows:** Often we must access the rows of a DataTable. The Rows property, which can be looped over, is ideal here.

**C# program that uses Rows, Field**

using System;

using System.Data;

class Program

{

static void Main()

{

// This uses the GetTable method (please paste it in).

DataTable data = GetTable();

// ... Loop over all rows.

foreach (DataRow row in data.**Rows**)

{

// ... Write value of first field as integer.

Console.WriteLine(row.Field<int>(0));

}

}

}

**Output**

25

50

10

21

100

**Using.** DataTable can be placed in a using statement. This can help programs' performance. Often the using block construct helps improve resource management.

**Tip:** We can add Columns and Rows to the DataTable instance inside (but not outside) the using block.

**Tip 2:** In this example, the Dispose method is invoked. After the using statement, Dispose() on DataTable is called.

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

**Dispose:** When Dispose is called, native resources are released. This may help resource usage problems.

**C# program that uses using statement**

using System;

using System.Data;

class Program

{

static void Main()

{

// Safely create and dispose of a DataTable.

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

{

// Two columns.

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

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

// ... Add two rows.

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

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

// ... Display first field.

Console.WriteLine(table.Rows[0].Field<string>(0));

}

}

}

**Output**

cat

**DataGridView example.** We insert data from object collections (like List) into DataTable. We then render that table to the screen with Windows Forms. DataTable helps display data.

**Tip:** DataTable works well with DataGridView. It makes DataGridView simpler and easier.

[**DataGridView**](https://www.dotnetperls.com/datagridview)

**Caution:** DataGridView has performance problems with manually adding rows. Using DataTable, List and DataSource helps.

**C# program that uses DataGridView**

using System.Collections.Generic;

using System.Data;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

/// <summary>

/// Contains column names.

/// </summary>

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

/// <summary>

/// 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();

}

/// <summary>

/// 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;

}

}

}

**Notes on example.**The code creates a DataTable and populates it with data. It is an entire Form and can be dropped into a Windows Forms application with a DataGridView in the designer.

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

**Note:** Sorry for the confusing code. The important parts are how we interact with the DataTable, as with Columns.Add.

**Loop through columns.** We have a collection that contains many arrays. Each of those arrays needs to be a new column. We use DataTable's methods to expand its size.

**Add:**We add the column names to our DataTable with Columns.Add. These are the column headers.

**Tip:** DataTable requires objects to assign to the cells. Objects can hold any type of data.

[**Object**](https://www.dotnetperls.com/object)

**DataSource.**With this property in Windows Forms, we can assign a DataTable to render the data. DataSource improves display performance. Everything is added at once.

**Sometimes:**It is helpful to assign DataSource to null to clear it—this internally resets the data binding.

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

// ... Draw new cells on DataGridView.

dataGridView1.**DataSource** = null;

dataGridView1.DataSource = GetResultsTable();

**Convert.**The List type is often useful. DataTable is better when we interact with controls such as DataGridView or with SQL databases. We can convert between the types. [**Convert List, DataTable**](https://www.dotnetperls.com/convert-list-datatable)

**Note:** Many DataTable-List conversions may be needed. Often conversions require custom logic to be successful.

**Rows.**Two important parts of DataTable are its Rows and Columns collections. With Rows, we access fields from a DataTable. We can loop over Rows with foreach. [**DataRow**](https://www.dotnetperls.com/datarow)

**Columns.**Think of columns as templates for rows. The fields in rows must adhere to the types specified in the corresponding DataColumns. Only a few columns are usually needed. [**DataColumn**](https://www.dotnetperls.com/datacolumn)

**DataSet.**We can combine multiple DataTable instances into one collection with the DataSet type. So a DataSet is a set of DataTables (and other things from System.Data).**[DataSet](https://www.dotnetperls.com/dataset)**

**DataView.** With this type, we can sort a DataTable's representation. We use the Sort property. The underlying data is not mutated. [**DataView**](https://www.dotnetperls.com/dataview)

**Select.**We can use the Select method on DataTable. This filters elements based on some criteria. We receive an array of DataRow instances that match our filter.[**Select**](https://www.dotnetperls.com/datatable-select)

**RowChanged.**A DataTable can monitor its changes. We use the RowChanged event, and similar events like ColumnChanged to detect changes. The AcceptChanges method too is helpful. [**RowChanged**](https://www.dotnetperls.com/datatable-rowchanged)

**A summary.**DataTable is a powerful, convenient class. We added columns. We added rows. DataTable is faster, in Windows Forms, than manually adding rows. It reduces complexity.