

# .NET Framework Notes for Professionals

## Chapter 6: ReadOnlyCollections Section 6.1: Creating a ReadOnlyCollection

### Using the Constructor

A `ReadOnlyCollection` is created by passing an existing `IList` object into the constructor:

```
var groceryList = new List<string> { "Apple", "Banana" };  
var readOnlyGroceryList = new ReadOnlyCollection<string>(groceryList);
```

### Using LINQ

Additionally, LINQ provides an `AsReadOnly()` extension method for `IList` objects:

```
var readOnlyVersion = groceryList.AsReadOnly();
```

### Note

Typically, you want to maintain the source collection privacy and allow public access. While you could create a `ReadOnlyCollection` from an in-line list, you should be unable after you created it.

## Section 6.2: Updating a ReadOnlyCollection

A `ReadOnlyCollection` cannot be edited directly. Instead, the source collection, `ReadOnlyCollection` will reflect these changes. This is the key feature of the `ReadOnlyCollection`.

```
var groceryList = new List<string> { "Apple", "Banana" };  
var readOnlyGroceryList = new ReadOnlyCollection<string>(groceryList);  
var itemCount = readOnlyGroceryList.Count; // There are currently 2 items.  
var listItem = readOnlyGroceryList[0]; // Compiler Error - 1  
//readOnlyGroceryList.Add("Candy"); // ...but they can be  
readOnlyCollection object  
groceryList.Add("Banana"); // How there are 3  
itemCount = readOnlyGroceryList.Count; // The last item is  
var listItem = readOnlyGroceryList.Last();
```

### View Demo

## Section 6.3: Warning: Elements in a ReadOnlyCollection are not inherently read-only

If the source collection is of a type that is not immutable, elements accessed through a `ReadOnlyCollection` can be

.NET Framework Notes for Professionals

## Chapter 21: SpeechRecognitionEngine class to recognize speech

LoadGrammar: Parameters	Details
grammar	The grammar to load. For example, a <code>DsctationGrammar</code> object to allow free text dictation.
RecognizeAsync: Parameters	Details
mode	The <code>RecognizeMode</code> for the current recognition: single for just one recognition, multiple to allow multiple.
GrammarBuilder.Append: Parameters	Details
choices	Provides some choices to the grammar builder. This means that, when the user inputs speech, the recognizer can follow different "branches" from a grammar.
Choices constructor: Parameters	Details
choices	An array of choices for the grammar builder. See <code>GrammarBuilder.Append</code> .
Grammar constructor: Parameter	Details
builder	The <code>GrammarBuilder</code> to construct a <code>Grammar</code> from.

## Section 21.1: Asynchronously recognizing speech based on a restricted set of phrases

```
SpeechRecognitionEngine recognitionEngine = new SpeechRecognitionEngine();  
GrammarBuilder builder = new GrammarBuilder();  
builder.Append(new Choices("I am", "You are", "He is", "She is", "We are", "They are"));  
builder.Append(new Choices("friendly", "unfriendly"));  
recognitionEngine.LoadGrammar(new Grammar(builder));  
recognitionEngine.SpeechRecognized += delegate(object sender, SpeechRecognizedEventArgs e)  
{  
    Console.WriteLine("You said: {0}", e.Result.Text);  
};  
recognitionEngine.SetInputToDefaultAudioDevice();  
recognitionEngine.RecognizeAsync(RecognizeMode.Multiple);
```

## Section 21.2: Asynchronously recognizing speech for free text dictation

```
using System.Speech.Recognition;  
// ...  
SpeechRecognitionEngine recognitionEngine = new SpeechRecognitionEngine();  
recognitionEngine.LoadGrammar(new DictationGrammar());  
recognitionEngine.SpeechRecognized += delegate(object sender, SpeechRecognizedEventArgs e)  
{  
    Console.WriteLine("You said: {0}", e.Result.Text);  
};  
recognitionEngine.SetInputToDefaultAudioDevice();  
recognitionEngine.RecognizeAsync(RecognizeMode.Multiple);
```

.NET Framework Notes for Professionals

## Chapter 19: Reading and writing Zip files

### Section 19.1: Listing ZIP contents

The snippet will list all the filenames of a zip archive. The filenames are relative to the zip root.

```
using (FileStream fs = new FileStream("archive.zip", FileMode.Open))  
using (ZipArchive archive = new ZipArchive(fs, ZipArchiveMode.Read))  
{  
    for (int i = 0; i < archive.Entries.Count; i++)  
    {  
        Console.WriteLine($"{i} {archive.Entries[i]}");  
    }  
}
```

### Section 19.2: Extracting files from ZIP files

Extracting all the files into a directory is very easy:

```
using (FileStream fs = new FileStream("archive.zip", FileMode.Open))  
using (ZipArchive archive = new ZipArchive(fs, ZipArchiveMode.Read))  
{  
    archive.ExtractToDirectory(AppDomain.CurrentDomain.BaseDirectory);  
}
```

When the file already exists, a `System.IO.IOException` will be thrown.

```
Extracting specific files:  
using (FileStream fs = new FileStream("archive.zip", FileMode.Open))  
using (ZipArchive archive = new ZipArchive(fs, ZipArchiveMode.Read))  
{  
    // Get a root entry file  
    archive.GetEntry("test.txt").ExtractToFile("test_extracted_getentries.txt", true);  
    // Enter a path if you want to extract files from a subdirectory  
    archive.GetEntry("subdir/test.txt").ExtractToFile("test_sub.txt", true);  
    // You can also use the Entries property to find files  
    archive.Entries.FirstOrDefault(f => f.Name == "test.txt").ExtractToFile("test_extracted_list.txt", true);  
    // This will throw a System.ArgumentOutOfRangeException because the file cannot be found  
    archive.GetEntry("nonexistingfile.txt").ExtractToFile("fail.txt", true);  
}
```

Any of these methods will produce the same result.

### Section 19.3: Updating a ZIP file

To update a ZIP file, the file has to be opened with `ZipArchiveMode.Update` instead.

```
using (FileStream fs = new FileStream("archive.zip", FileMode.Open))
```

.NET Framework Notes for Professionals

100+ pages  
of professional hints and tricks

# Contents

<b>About</b>	1
<b>Chapter 1: Getting started with .NET Framework</b>	2
<a href="#">Section 1.1: Hello World in C#</a>	2
<a href="#">Section 1.2: Hello World in F#</a>	3
<a href="#">Section 1.3: Hello World in Visual Basic .NET</a>	3
<a href="#">Section 1.4: Hello World in C++/CLI</a>	3
<a href="#">Section 1.5: Hello World in IL</a>	3
<a href="#">Section 1.6: Hello World in PowerShell</a>	4
<a href="#">Section 1.7: Hello World in Nemerle</a>	4
<a href="#">Section 1.8: Hello World in Python (IronPython)</a>	4
<a href="#">Section 1.9: Hello World in Oxygene</a>	4
<a href="#">Section 1.10: Hello World in Boo</a>	4
<b>Chapter 2: Strings</b>	5
<a href="#">Section 2.1: Count characters</a>	5
<a href="#">Section 2.2: Count distinct characters</a>	5
<a href="#">Section 2.3: Convert string to/from another encoding</a>	5
<a href="#">Section 2.4: Comparing strings</a>	6
<a href="#">Section 2.5: Count occurrences of a character</a>	6
<a href="#">Section 2.6: Split string into fixed length blocks</a>	6
<a href="#">Section 2.7: Object.ToString() virtual method</a>	7
<a href="#">Section 2.8: Immutability of strings</a>	8
<b>Chapter 3: DateTime parsing</b>	9
<a href="#">Section 3.1: ParseExact</a>	9
<a href="#">Section 3.2: TryParse</a>	10
<a href="#">Section 3.3: TryParseExact</a>	12
<b>Chapter 4: Dictionaries</b>	13
<a href="#">Section 4.1: Initializing a Dictionary with a Collection Initializer</a>	13
<a href="#">Section 4.2: Adding to a Dictionary</a>	13
<a href="#">Section 4.3: Getting a value from a dictionary</a>	13
<a href="#">Section 4.4: Make a Dictionary&lt;string, T&gt; with Case-Insensitivte keys</a>	14
<a href="#">Section 4.5: IEnumerable to Dictionary (≥ .NET 3.5)</a>	14
<a href="#">Section 4.6: Enumerating a Dictionary</a>	14
<a href="#">Section 4.7: ConcurrentDictionary&lt;TKey, TValue&gt; (from .NET 4.0)</a>	15
<a href="#">Section 4.8: Dictionary to List</a>	16
<a href="#">Section 4.9: Removing from a Dictionary</a>	16
<a href="#">Section 4.10: ContainsKey(TKey)</a>	17
<a href="#">Section 4.11: ConcurrentDictionary augmented with Lazy'1 reduces duplicated computation</a>	17
<b>Chapter 5: Collections</b>	19
<a href="#">Section 5.1: Using collection initializers</a>	19
<a href="#">Section 5.2: Stack</a>	19
<a href="#">Section 5.3: Creating an initialized List with Custom Types</a>	20
<a href="#">Section 5.4: Queue</a>	22
<b>Chapter 6: ReadOnlyCollections</b>	24
<a href="#">Section 6.1: Creating a ReadOnlyCollection</a>	24
<a href="#">Section 6.2: Updating a ReadOnlyCollection</a>	24
<a href="#">Section 6.3: Warning: Elements in a ReadOnlyCollection are not inherently read-only</a>	24

<b>Chapter 7: Stack and Heap</b>	26
Section 7.1: Value types in use	26
Section 7.2: Reference types in use	26
<b>Chapter 8: LINQ</b>	28
Section 8.1: SelectMany (flat map)	28
Section 8.2: Where (filter)	29
Section 8.3: Any	29
Section 8.4: GroupJoin	30
Section 8.5: Except	31
Section 8.6: Zip	31
Section 8.7: Aggregate (fold)	31
Section 8.8: ToLookup	32
Section 8.9: Intersect	32
Section 8.10: Concat	32
Section 8.11: All	32
Section 8.12: Sum	33
Section 8.13: SequenceEqual	33
Section 8.14: Min	33
Section 8.15: Distinct	34
Section 8.16: Count	34
Section 8.17: Cast	34
Section 8.18: Range	34
Section 8.19: ThenBy	35
Section 8.20: Repeat	35
Section 8.21: Empty	35
Section 8.22: Select (map)	35
Section 8.23: OrderBy	36
Section 8.24: OrderByDescending	36
Section 8.25: Contains	36
Section 8.26: First (find)	36
Section 8.27: Single	37
Section 8.28: Last	37
Section 8.29: LastOrDefault	37
Section 8.30: SingleOrDefault	38
Section 8.31: FirstOrDefault	38
Section 8.32: Skip	38
Section 8.33: Take	39
Section 8.34: Reverse	39
Section 8.35: OfType	39
Section 8.36: Max	39
Section 8.37: Average	39
Section 8.38: GroupBy	40
Section 8.39: ToDictionary	40
Section 8.40: Union	41
Section 8.41: ToArray	42
Section 8.42: ToList	42
Section 8.43: ElementAt	42
Section 8.44: ElementAtOrDefault	42
Section 8.45: SkipWhile	42
Section 8.46: TakeWhile	43

<a href="#">Section 8.47: DefaultIfEmpty</a> .....	43
<a href="#">Section 8.48: Join</a> .....	43
<a href="#">Section 8.49: Left Outer Join</a> .....	44
<b><a href="#">Chapter 9: ForEach</a></b> .....	46
<a href="#">Section 9.1: Extension method for IEnumerable</a> .....	46
<a href="#">Section 9.2: Calling a method on an object in a list</a> .....	46
<b><a href="#">Chapter 10: Reflection</a></b> .....	47
<a href="#">Section 10.1: What is an Assembly?</a> .....	47
<a href="#">Section 10.2: Compare two objects with reflection</a> .....	47
<a href="#">Section 10.3: Creating Object and setting properties using reflection</a> .....	48
<a href="#">Section 10.4: How to create an object of T using Reflection</a> .....	48
<a href="#">Section 10.5: Getting an attribute of an enum with reflection (and caching it)</a> .....	48
<b><a href="#">Chapter 11: Expression Trees</a></b> .....	50
<a href="#">Section 11.1: building a predicate of form field == value</a> .....	50
<a href="#">Section 11.2: Simple Expression Tree Generated by the C# Compiler</a> .....	50
<a href="#">Section 11.3: Expression for retrieving a static field</a> .....	51
<a href="#">Section 11.4: InvocationExpression Class</a> .....	51
<b><a href="#">Chapter 12: Custom Types</a></b> .....	54
<a href="#">Section 12.1: Struct Definition</a> .....	54
<a href="#">Section 12.2: Class Definition</a> .....	54
<b><a href="#">Chapter 13: Code Contracts</a></b> .....	56
<a href="#">Section 13.1: Contracts for Interfaces</a> .....	56
<a href="#">Section 13.2: Installing and Enabling Code Contracts</a> .....	56
<a href="#">Section 13.3: Preconditions</a> .....	58
<a href="#">Section 13.4: Postconditions</a> .....	59
<b><a href="#">Chapter 14: Settings</a></b> .....	60
<a href="#">Section 14.1: AppSettings from ConfigurationSettings in .NET 1.x</a> .....	60
<a href="#">Section 14.2: Reading AppSettings from ConfigurationManager in .NET 2.0 and later</a> .....	60
<a href="#">Section 14.3: Introduction to strongly-typed application and user settings support from Visual Studio</a> .....	61
<a href="#">Section 14.4: Reading strongly-typed settings from custom section of configuration file</a> .....	62
<b><a href="#">Chapter 15: Regular Expressions (System.Text.RegularExpressions)</a></b> .....	65
<a href="#">Section 15.1: Check if pattern matches input</a> .....	65
<a href="#">Section 15.2: Remove non alphanumeric characters from string</a> .....	65
<a href="#">Section 15.3: Passing Options</a> .....	65
<a href="#">Section 15.4: Match into groups</a> .....	65
<a href="#">Section 15.5: Find all matches</a> .....	65
<a href="#">Section 15.6: Simple match and replace</a> .....	66
<b><a href="#">Chapter 16: File Input/Output</a></b> .....	67
<a href="#">Section 16.1: C# File.Exists()</a> .....	67
<a href="#">Section 16.2: VB WriteAllText</a> .....	67
<a href="#">Section 16.3: VB StreamWriter</a> .....	67
<a href="#">Section 16.4: C# StreamWriter</a> .....	67
<a href="#">Section 16.5: C# WriteAllText()</a> .....	68
<b><a href="#">Chapter 17: System.IO</a></b> .....	69
<a href="#">Section 17.1: Reading a text file using StreamReader</a> .....	69
<a href="#">Section 17.2: Serial Ports using System.IO.SerialPorts</a> .....	69
<a href="#">Section 17.3: Reading/Writing Data Using System.IO.File</a> .....	70
<b><a href="#">Chapter 18: System.IO.File class</a></b> .....	72

Section 18.1: Delete a file .....	72
Section 18.2: Strip unwanted lines from a text file .....	73
Section 18.3: Convert text file encoding .....	73
Section 18.4: Enumerate files older than a specified amount .....	74
Section 18.5: Move a File from one location to another .....	74
<b>Chapter 19: Reading and writing Zip files .....</b>	<b>76</b>
Section 19.1: Listing ZIP contents .....	76
Section 19.2: Extracting files from ZIP files .....	76
Section 19.3: Updating a ZIP file .....	76
<b>Chapter 20: Managed Extensibility Framework .....</b>	<b>78</b>
Section 20.1: Connecting (Basic) .....	78
Section 20.2: Exporting a Type (Basic) .....	78
Section 20.3: Importing (Basic) .....	79
<b>Chapter 21: SpeechRecognitionEngine class to recognize speech .....</b>	<b>80</b>
Section 21.1: Asynchronously recognizing speech based on a restricted set of phrases .....	80
Section 21.2: Asynchronously recognizing speech for free text dictation .....	80
<b>Chapter 22: System.Runtime.Caching.MemoryCache (ObjectCache) .....</b>	<b>81</b>
Section 22.1: Adding Item to Cache (Set) .....	81
Section 22.2: System.Runtime.Caching.MemoryCache (ObjectCache) .....	81
<b>Chapter 23: System.Reflection.Emit namespace .....</b>	<b>83</b>
Section 23.1: Creating an assembly dynamically .....	83
<b>Chapter 24: .NET Core .....</b>	<b>86</b>
Section 24.1: Basic Console App .....	86
<b>Chapter 25: ADO.NET .....</b>	<b>87</b>
Section 25.1: Best Practices - Executing Sql Statements .....	87
Section 25.2: Executing SQL statements as a command .....	88
Section 25.3: Using common interfaces to abstract away vendor specific classes .....	89
<b>Chapter 26: Dependency Injection .....</b>	<b>90</b>
Section 26.1: How Dependency Injection Makes Unit Testing Easier .....	90
Section 26.2: Dependency Injection - Simple example .....	90
Section 26.3: Why We Use Dependency Injection Containers (IoC Containers) .....	91
<b>Chapter 27: Platform Invoke .....</b>	<b>94</b>
Section 27.1: Marshaling structs .....	94
Section 27.2: Marshaling unions .....	95
Section 27.3: Calling a Win32 dll function .....	96
Section 27.4: Using Windows API .....	97
Section 27.5: Marshalling arrays .....	97
<b>Chapter 28: NuGet packaging system .....</b>	<b>98</b>
Section 28.1: Uninstalling a package from one project in a solution .....	98
Section 28.2: Installing a specific version of a package .....	98
Section 28.3: Adding a package source feed (MyGet, Klondike, ect) .....	98
Section 28.4: Installing the NuGet Package Manager .....	98
Section 28.5: Managing Packages through the UI .....	99
Section 28.6: Managing Packages through the console .....	99
Section 28.7: Updating a package .....	99
Section 28.8: Uninstalling a package .....	100
Section 28.9: Uninstall a specific version of package .....	100
<b>Chapter 29: Globalization in ASP.NET MVC using Smart internationalization for ASP.NET .....</b>	<b>101</b>

[Click here to download full PDF material](#)