This PDF contains 2 Appendices: Appendix A and Appendix B.

Appendix A

Answers to the Test Your Knowledge Questions

This appendix has the answers to the questions in the *Test Your Knowledge* section at the end of each chapter.

Chapter 1 – Hello, C#! Welcome, .NET!

1. Why can a programmer use different languages, for example, C# and F#, to write applications that run on .NET?

   Answer: Multiple languages are supported on .NET because each one has a compiler that translates the source code into **intermediate language** (**IL**) code. This IL code is then compiled to native CPU instructions at runtime by the CLR.

2. What do you type at the prompt to create a console app?

   Answer: You enter `dotnet new console`.

3. What do you type at the prompt to build and execute C# source code?

   Answer: In a folder with a `ProjectName.csproj` file, you enter `dotnet run`.

4. What is the Visual Studio Code keyboard shortcut to view Terminal?

   Answer: *Ctrl* + ` (back tick).

5. Is Visual Studio 2019 better than Visual Studio Code?

   Answer: No. Each is optimized for different tasks. Visual Studio 2019 is large, heavy-weight, and can create applications with graphical user interfaces, for example, Windows Forms, WPF, UWP, and Xamarin.Forms mobile apps, but it is only available on Windows. Visual Studio Code is smaller, lighter-weight, code-focused, supports many more languages, and is available cross-platform. In 2021, with the release of .NET 6 and .NET **Multi-platform App User Interface** (**MAUI**), Visual Studio Code will get an extension that enables building user interfaces for desktop and mobile apps.

6. Is .NET Core better than .NET Framework?

   Answer: For modern development, yes, but it depends on what you need.

   .NET Core is a modern, cross-platform version of the legacy, mature .NET Framework. .NET Core is more frequently improved. .NET 5 is the latest version of .NET Core with a name change. .NET Framework has better support for legacy applications; however, the current version 4.8 was the last major release. It will never support some language features of C# 8 and 9.

7. What is .NET Standard and why is it still important?

   Answer: .NET Standard defines an API that a .NET platform can implement.

   The latest versions of .NET Framework, .NET Core, .NET 5, and Xamarin implement .NET Standard 2.0 to provide a single, standard API that developers can target for maximum reuse. .NET Core 3.0 or later, including .NET 5 and Xamarin, implement .NET Standard 2.1, which has some new features not supported by .NET Framework. If you want to create a new class library that supports all .NET platforms you will need it to be .NET Standard 2.0-compatible.

8. What is the name of the entry point method of a .NET console application and how should it be declared?

   Answer: The entry point of a .NET console application is the `Main` method. An optional `string` array for command-line arguments and a return type of `int` are recommended, but they are not required. It can be declared as shown in the following code:

   ```
   public static void Main() // minimum
   public static int Main(string[] args) // recommended
   ```

9. Where would you look for help about a C# keyword?

   Answer: The Microsoft Docs website. Specifically, C# keywords are documented at the following link: `https://docs.microsoft.com/en-us/dotnet/articles/csharp/language-reference/keywords/`.

10. Where would you look for solutions to common programming problems?

    Answer: `https://stackoverflow.com/`.

Chapter 2 – Speaking C#

Exercise 2.1 – Test your knowledge

What type would you choose for the following "numbers?"

1. A person's telephone number.

   Answer: `string`.

2. A person's height.

   Answer: `float` or `double`.

3. A person's age.

   Answer: `int` for best performance on most CPUs or `byte` (0 to 255) for smallest size.

4. A person's salary.

   Answer: `decimal`.

5. A book's ISBN.

   Answer: `string`.

6. A book's price.

   Answer: `decimal`.

7. A book's shipping weight.

   Answer: `float` or `double`.

8. A country's population.

   Answer: `uint` (0 to about 4 billion).

9. The number of stars in the universe.

   Answer: `ulong` (0 to about 18 quadrillion) or `System.Numerics.BigInteger` (allows an arbitrarily large integer).

10. The number of employees in each of the small or medium businesses in the United Kingdom (up to about 50,000 employees per business).

    Answer: Since there are hundreds of thousands of small or medium businesses, we need to take memory size as the determining factor, so choose `ushort`, because it only takes 2 bytes compared to an `int`, which takes 4 bytes.

Chapter 3 – Controlling the Flow and Converting Types

Exercise 3.1 – Test your knowledge

Answer the following questions:

1. What happens when you divide an `int` variable by 0?

   Answer: `DivideByZeroException` is thrown when dividing an integer or decimal.

2. What happens when you divide a `double` variable by 0?

   Answer: The `double` type contains a special value of `Infinity`. Instances of floating-point numbers can have the special values of: `NaN` (not a number), or in the case of dividing by 0, either `PositiveInfinity` or `NegativeInfinity`.

3. What happens when you overflow an `int` variable, that is, set it to a value beyond its range?

   Answer: It will loop unless you wrap the statement in a `checked` block, in which case, `OverflowException` will be thrown.

4. What is the difference between the statements `x = y++;` and `x = ++y;`?

   Answer: In the statement `x = y++;`, the current value of `y` will be assigned to `x` and then `y` will be incremented, and in the statement `x = ++y;`, the value of `y` will be incremented and then the result will be assigned to `x`.

5. What is the difference between `break`, `continue`, and `return` when used inside a loop statement?

   Answer: The `break` statement will end the whole loop and continue executing after the loop, the `continue` statement will end the current iteration of the loop and continue executing at the start of the loop block for the next iteration, and the `return` statement will end the current method call and continue executing after the method call.

6. What are the three parts of a `for` statement and which of them are required?

   Answer: The three parts of a `for` statement are the initializer, condition, and incrementer expressions. The condition expression is required to evaluate to `true` or `false`, but the other two are optional.

7. What is the difference between the `=` and `==` operators?

   Answer: The `=` operator is the assignment operator for assigning values to variables, while the `==` operator is the equality check operator that returns `true` or `false`.

8. Does the following statement compile? `for ( ; true; ) ;`

   Answer: Yes. The `for` statement only requires the condition expression that evaluates to `true` or `false`. The initializer and incrementer expressions are optional. This `for` statement will execute the `empty;` statement after the close brace, forever. It is an example of an infinite loop.

9. What does the underscore _ represent in a switch expression?

   Answer: The underscore _ represents the default return value.

10. What interface must an object implement to be enumerated over by using the `foreach` statement?

    Answer: An object must implement the `IEnumerable` interface.

Exercise 3.2 – Explore loops and overflow

What will happen if this code executes?

```
int max = 500;
for (byte i = 0; i < max; i++)
{
  WriteLine(i);
}
```

Answer:

The code will loop forever because the value of `i` can only be between `0` and `255`. Once `i` gets incremented beyond `255`, it loops back to `0` and therefore will always be less than the max (`500`).

To prevent the infinite loop, you can add a `checked` statement around the code. This would cause an exception to be thrown after `255` due to the overflow, as shown in the following output:

```
254
255
System.OverflowException says Arithmetic operation resulted in an
overflow.
```

Exercise 3.5 – Test your knowledge of operators

1. What are the values of `x` and `y` after the following statements execute?

   ```
   x = 3;
   y = 2 + ++x;
   ```

   Answer: `x` is `4` and `y` is `6`

2. What are the values of `x` and `y` after the following statements execute?

   ```
   x = 3 << 2;
   y = 10 >> 1;
   ```

   Answer: `x` is `12` and `y` is `5`

3. What are the values of `x` and `y` after the following statements execute?

```
x = 10 & 8;
y = 10 | 7;
```

Answer: `x` is `8` and `y` is `15`

Chapter 4 – Writing, Debugging, and Testing Functions

1. What does the C# keyword `void` mean?

   Answer: It indicates that a method has no return value.

2. What are some differences between imperative and functional programming styles?

   Answer: Imperative programming style means writing a sequence of statements that the runtime executes step by step like a recipe. Your code tells the runtime exactly how to perform the task. Do this. Now do that. It has variables meaning that the state can change at any time, including outside the current function. Imperative programming causes side effects, changing the value of some state somewhere in your program. Side effects are tricky to debug. Functional programming style describes what you want to achieve instead of how. It can also be described as declarative. But the most important point is that functional programming languages make all states immutable by default to avoid side effects.

3. In Visual Studio Code, what is the difference between pressing *F5*; *Ctrl* or *Cmd* + *F5*; *Shift* + *F5*; and *Ctrl* or *Cmd* + *Shift* + *F5*?

   Answer: *F5* saves, compiles, runs, and attaches the debugger; *Ctrl* or *Cmd* + *F5* saves, compiles, and runs the application with the debugger attached; *Shift* + *F5* stops the debugger and running application; and *Ctrl* or *Cmd* + *Shift* + *F5* restarts the application with the debugger attached.

4. Where does the `Trace.WriteLine` method write its output to?

   Answer: `Trace.WriteLine` writes its output to any configured trace listeners. By default, this includes Visual Studio Code's Terminal or the command line but can be configured to be a text file or any custom listener.

5. What are the five trace levels?

   Answer: 0 = Off, 1 = Error, 2 = Warning, 3 = Info, and 4 = Verbose.

6. What is the difference between `Debug` and `Trace`?

   Answer: `Debug` is active only during development. `Trace` is active during development and after release into production.

7. When writing a unit test, what are the three As?

   Answer: Arrange, Act, Assert.

8. When writing a unit test using `xUnit`, what attribute must you decorate the test methods with?

   Answer: `[Fact]`.

9. What `dotnet` command executes `xUnit` tests?

   Answer: `dotnet test`.

10. What is TDD?

    Answer: Test-Driven Development.

More Information: You can read about TDD in an article about unit testing at the following link: `https://docs.microsoft.com/en-us/dotnet/core/testing/`

Chapter 5 – Building Your Own Types with Object-Oriented Programming

1. What are the six combinations of access modifier keywords and what do they do?

   Answer: The six combinations of access modifier keywords and their effects are described in the following list:

   - `private`: This modifier makes a member only visible inside the class.
   - `internal`: This modifier makes a member only visible inside the class or within the same assembly.
   - `protected`: This modifier makes a member only visible inside the class or derived classes.
   - `internal protected`: This modifier makes a member only visible inside the class, derived classes, or within the same assembly.
   - `private protected`: This modifier makes a member only visible inside the class or derived classes and within the same assembly.
   - `public`: This modifier makes a member visible everywhere.

2. What is the difference between the `static`, `const`, and `readonly` keywords when applied to a type member?

   Answer: The difference between the `static`, `const`, and `readonly` keywords when applied to a type member are described in the following list:

   - `static`: This keyword makes the member shared by all instances and it must be accessed through the type, not an instance of the type.
   - `const`: This keyword makes the field a fixed literal value that must never change because during the compilation, assemblies that use the field copy the literal value at the time of compilation.

- `readonly`: This keyword restricts the field so that it can only be assigned to using a constructor or field initializer at runtime.

3. What does a constructor do?

   Answer: A constructor allocates memory and initializes field values.

4. Why do you need to apply the `[Flags]` attribute to an `enum` type when you want to store combined values?

   Answer: If you don't apply the `[Flags]` attribute to an `enum` type when you want to store combined values, then a stored `enum` value that is a combination will return as the stored integer value instead of a comma-separated list of text values.

5. Why is the `partial` keyword useful?

   Answer: You can use the `partial` keyword to split the definition of a type over multiple files.

6. What is a tuple?

   Answer: A data structure consisting of multiple parts. They are used when you want to store multiple values as a unit without defining a type for them.

7. What does the C# `record` keyword do?

   Answer: The C# `record` keyword defines a data structure that is immutable to enable a more functional programming style. Like a class, a record can have properties and methods, but the values of properties can only be set during initialization.

8. What does overloading mean?

   Answer: Overloading is when you define more than one method with the same method name and different input parameters.

9. What is the difference between a field and a property?

   Answer: A field is a data storage location that can be referenced. A property is one or a pair of methods that get and/or set a value. The value for a property is often stored in a `private` field.

10. How do you make a method parameter optional?

    Answer: You make a method parameter optional by assigning a default value to it in the method signature.

Chapter 6 – Implementing Interfaces and Inheriting Classes

1. What is a delegate?

   Answer: A delegate is a type-safe method reference. It can be used to execute any method with a matching signature.

2. What is an event?

   Answer: An event is a field that is a delegate having the `event` keyword applied. The keyword ensures that only `+=` and `-=` are used; this safely combines multiple delegates without replacing any existing event handlers.

3. How are a base class and a derived class related and how can the derived class access the base class?

   Answer: A derived class (or subclass) is a class that inherits from a base class (or superclass). Inside a derived class you use the `base` keyword to access the class that the subclass inherits from.

4. What is the difference between the `is` and `as` operators?

   Answer: The `is` operator returns `true` if an object can be cast to the type; otherwise it returns `false`. The `as` operator returns a reference to the object if an object can be cast to the type; otherwise, it returns `null`.

5. Which keyword is used to prevent a class from being derived from, or a method from being overridden?

   Answer: `sealed`.

More Information: You can read about the `sealed` keyword at the following link: `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/sealed`

6. Which keyword is used to prevent a class from being instantiated with the `new` keyword?

   Answer: `abstract`.

More Information: You can read about the `abstract` keyword at the following link: `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/abstract`

7. Which keyword is used to allow a member to be overridden?

   Answer: `virtual`.

More Information: You can read about the `virtual` keyword at the following link: `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual`

8. What's the difference between a **destructor** and a **deconstruct** method?

   Answer: A **destructor**, also known as a **finalizer**, must be used to release resources owned by the object. A **deconstruct** method is a feature of C# 7 or later that allows a complex object to be broken down into smaller parts. It is especially useful when working with tuples.

More Information: You can read about deconstructing at the following link: `https://docs.microsoft.com/en-us/dotnet/csharp/deconstruct`

9. What are the signatures of the constructors that all exceptions should have?

   Answer: The signatures of the three constructors that all exceptions should have are shown in the following list:

   - A constructor with no parameters.
   - A constructor with a `string` parameter, usually named `message`.
   - A constructor with a `string` parameter, usually named `message`, and an `Exception` parameter, usually named `innerException`.

10. What is an extension method and how do you define one?

    Answer: An extension method is a compiler trick that makes a `static` method of a `static` class appear to be one of the members of another type. You define which type you want to extend by prefixing the type parameter in the method with the `this` keyword.

Chapter 7 – Understanding and Packaging .NET Types

1. What is the difference between a namespace and an assembly?

   Answer: A **namespace** is the logical container of a type. An **assembly** is the physical container of a type. To use a type, the developer must reference its assembly. Optionally, the developer can import its namespace, or specify the namespace when specifying the type.

2. How do you reference another project in a `.csproj` file?

   Answer: You reference another project in a `.csproj` file by adding a `<ProjectReference>` element that sets its `Include` attribute to a path to the reference project file inside an `<ItemGroup>` element, as shown in the following markup:

```
<ItemGroup>
  <ProjectReference Include="..\Calculator\Calculator.csproj" />
</ItemGroup>
```

3. What is the benefit of a tool like ILSpy?

   Answer: A benefit of a tool like ILSpy is learning how to write code in C# for the .NET platform by seeing how other packages are written. You should never steal their intellectual property of course. But it is especially useful to see how the Microsoft developers have implemented key components of the Base Class Libraries.

Decompiling can also be useful when calling a third-party library that you need to better understand how it works to call it appropriately.

4. Which .NET type does the C# `float` alias represent?

   Answer: `System.Single`.

5. What tool should you use before porting an application from .NET Framework to .NET 5?

   Answer: You should use the .NET Portability Analyzer before porting an application from .NET Framework to .NET 5.

6. What is the difference between framework-dependent and self-contained deployments of .NET Core applications?

   Answer: Framework-dependent .NET Core applications require .NET Core to exist for an operating system to execute. Self-contained .NET Core applications include everything necessary to execute on their own.

7. What is a RID?

   Answer: RID is the acronym for Runtime Identifier. RID values are used to identify target platforms where a .NET Core application runs.

8. What is the difference between the `dotnet pack` and `dotnet publish` commands?

   Answer: The `dotnet pack` command creates a NuGet package. The `dotnet publish` command puts the application and its dependencies into a folder for deployment to a hosting system.

9. What types of applications written for .NET Framework can be ported to .NET Core 3.0 or later?

   Answer: Console, ASP.NET MVC, ASP.NET Web API, Windows Forms, and Windows Presentation Foundation (WPF).

10. Can you use packages written for .NET Framework with .NET 5?

    Answer: Yes, as long as they only call APIs in .NET Standard 2.0.

Chapter 8 – Working with Common .NET Types

1. What is the maximum number of characters that can be stored in a `string` variable?

   Answer: The maximum size of a `string` variable is 2 GB or about one billion characters, because each `char` uses 2 bytes due to the internal use of Unicode (UTF-16) encoding for characters in a `string`.

2. When and why should you use a `SecureString` type?

   Answer: The `string` type leaves text data in the memory for too long and it's too visible. The `SecureString` type encrypts its text and ensures that the memory is released immediately. For example, the `PasswordBox` control stores its password as a `SecureString` variable, and when starting a new process, the `Password` parameter must be a `SecureString` variable.

More Information: You can read about `SecureString` at the following link: `https://stackoverflow.com/questions/141203/when-would-i-need-a-securestring-in-net`

3. When is it appropriate to use a `StringBuilder` class?

   Answer: When concatenating more than about three `string` variables, you will use less memory and get improved performance using `StringBuilder` than using the `string.Concat` method or the `+` operator.

4. When should you use the `LinkedList<T>` class?

   Answer: Each item in a linked list has a reference to its previous and next siblings as well as the list itself. A linked list should be used when items need to be inserted and removed from positions in the list without actually moving the items in memory.

5. When should you use the `SortedDictionary<T>` class rather than the `SortedList<T>` class?

   Answer: The `SortedList<T>` class uses less memory than `SortedDictionary<T>`; `SortedDictionary<T>` has faster insertion and removal operations for unsorted data. If the list is populated all at once from sorted data, `SortedList<T>` is faster than `SortedDictionary<T>`.

More Information: You can read about `SortedDictionary<T>` and `SortedList<T>` at the following link: `https://stackoverflow.com/questions/935621/whats-the-difference-between-sortedlist-and-sorteddictionary`

6. What is the ISO culture code for Welsh?

   Answer: `cy-GB`.

More Information: You can view a complete list of culture codes at the following link: `https://lonewolfonline.net/list-net-culture-country-codes/`

7. What is the difference between **localization**, **globalization**, and **internationalization**?

   Answer:

   - **Localization** affects the user interface of your application. Localization is controlled by a neutral (language only) or specific (language and region) culture. You provide multiple language versions of text and other values. For example, the label of a text box might be **First name** in English, and **Prénom** in French.

- **Globalization** affects the data of your application. Globalization is controlled by a specific (language and region) culture, for example, `en-GB` for British English, or `fr-CA` for Canadian French. The culture must be specific because a decimal value formatted as currency must know to use Canadian dollars instead of French euros.
- **Internationalization** is the combination of localization and globalization.

8. In a regular expression, what does `$` mean?

   Answer: In a regular expression, `$` represents the end of the input.

9. In a regular expression, how could you represent digits?

   Answer: In a regular expression, you could represent digits using `\d` or `[0-9]`.

10. Why should you not use the official standard for email addresses to create a regular expression to validate a user's email address?

    Answer: The effort is not worth the pain for you or your users. Validating an email address using the official specification doesn't check whether that address exists or whether the person entering the address is its owner.

More Information: You can read about why you should simplify the way you validate email addresses at the following links: `https://davidcel.is/posts/stop-validating-email-addresses-with-regex/` and `https://stackoverflow.com/questions/201323/how-to-validate-an-email-address-using-a-regular-expression`

Chapter 9 – Working with Files, Streams, and Serialization

1. What is the difference between using the `File` class and the `FileInfo` class?

   Answer: The `File` class has `static` methods and it cannot be instantiated. It is best used for one-off tasks such as copying a file. The `FileInfo` class requires the instantiation of an object that represents a file. It is best used when you need to perform multiple operations on the same file.

2. What is the difference between the `ReadByte` method and the `Read` method of a stream?

   Answer: The `ReadByte` method returns a single byte each time it is called and the `Read` method fills a temporary array with bytes up to a specified length. It is generally best to use `Read` to process multiple bytes at once.

3. When would you use the `StringReader`, `TextReader`, and `StreamReader` classes?

   Answer:

   - `StringReader` is used for efficiently reading from a string stored in memory.
   - `TextReader` is an abstract class that `StringReader` and `StreamReader` both inherit from for their shared functionality.

- `StreamReader` is used for reading strings from a stream that can be any type of text file, including XML and JSON.

4. What does the `DeflateStream` type do?

   Answer: `DeflateStream` implements the same compression algorithm as GZIP, but without a cyclical redundancy check; so, although it produces smaller compressed files, it cannot perform integrity checks when decompressing.

5. How many bytes per character does the UTF-8 encoding use?

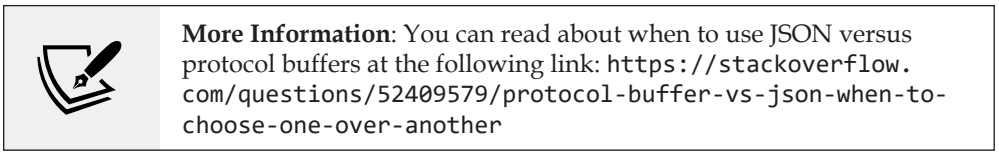
   Answer: The number of bytes per character used by the UTF-8 encoding depends on the character. Most Western alphabet characters are stored using one byte. Other characters may need two or more bytes.

6. What is an object graph?

   Answer: An object graph is any set of connected instances of classes that reference each other. For example, a `Customer` object may have a property named `Orders` that references a collection of `Order` instances.

7. What is the best serialization format to choose for minimizing space requirements?

   Answer: **JavaScript Object Notation** (**JSON**) has a good balance between space requirements and practical factors like human-readability, but protocol buffers is best for minimizing space requirements.



8. What is the best serialization format to choose for cross-platform compatibility?

   Answer: There is still an argument for **eXtensible Markup Language** (**XML**) if you need maximum compatibility, especially with legacy systems, although JSON is better if you need to integrate with web systems, or protocol buffers for best performance and minimum bandwidth use.

9. Why is it bad to use a `string` value like `"\Code\Chapter01"` to represent a path and what should you do instead?

   Answer: It is bad to use a `string` value like `"\Code\Chapter01"` to represent a path because it assumes that backslashes are used as a folder separator on all operating systems. Instead, you should use the `Path.Combine` method and pass separate `string` values for each folder, or a `string` array, as shown in the following code:

```
string path = Path.Combine(new[] { "Code", "Chapter01" });
```

10. Where can you find information about NuGet packages and their dependencies?

    Answer: You can find information about NuGet packages and their dependencies at the following link: `https://www.nuget.org/`.

Chapter 10 – Protecting Your Data and Applications

1. Of the encryption algorithms provided by .NET, which is the best choice for symmetric encryption?

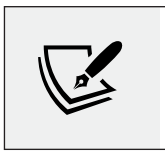
   Answer: The AES algorithm is the best choice for symmetric encryption.

2. Of the encryption algorithms provided by .NET, which is the best choice for asymmetric encryption?

   Answer: The RSA algorithm is the best choice for asymmetric encryption.

3. What is a rainbow attack?

   Answer: A rainbow attack uses a table of precalculated hashes of passwords. When a database of password hashes is stolen, the attacker can compare against the rainbow table hashes quickly and determine the original passwords.



More Information: You can read about rainbow tables at the following link: `https://learncryptography.com/hash-functions/rainbow-tables`

4. For encryption algorithms, is it better to have a larger or smaller block size?

   Answer: For encryption algorithms, it is better to have a smaller block size.

5. What is a cryptographic hash?

   Answer: A cryptographic hash is a fixed-size output that results from an input of arbitrary size being processed by a hash function. Hash functions are one-way, which means that the only way to recreate the original input is to brute-force all possible inputs and compare the results.

6. What is a cryptographic signature?

   Answer: A cryptographic signature is a value appended to a digital document to prove its authenticity. A valid signature tells the recipient that the document was created by a known sender and it has not been modified.

7. What is the difference between symmetric and asymmetric encryption?

   Answer: Symmetric encryption uses a secret shared key to both encrypt and decrypt. Asymmetric encryption uses a public key to encrypt and a private key to decrypt.

8. What does RSA stand for?

   Answer: Rivest-Shamir-Adleman, the surnames of the three men who publicly described the algorithm in 1978.

9. Why should passwords be salted before being stored?

   Answer: To slow down rainbow dictionary attacks.

10. SHA-1 is a hashing algorithm designed by the United States National Security Agency. Why should you never use it?

    Answer: SHA-1 is no longer secure. All modern browsers have stopped accepting SHA-1 SSL certificates.

Chapter 11 – Working with Databases Using Entity Framework Core

1. What type would you use for the property that represents a table, for example, the `Products` property of a database context?

   Answer: `DbSet<T>`, where `T` is the entity type, for example, `Product`.

2. What type would you use for the property that represents a one-to-many relationship, for example, the `Products` property of a `Category` entity?

   Answer: `ICollection<T>`, where `T` is the entity type, for example, `Product`.

3. What is the EF convention for primary keys?

   Answer: The property named `ID` or `ClassNameID` is assumed to be the primary key. If the type of that property is any of the following, then the property is also marked as being an `IDENTITY` column: `tinyint`, `smallint`, `int`, `bigint`, `guid`.

4. When might you use an annotation attribute in an entity class?

   Answer: You might use an annotation attribute in an entity class when the conventions cannot work out the correct mapping between the classes and tables; for example, if a class name does not match a table name or a property name does not match a column name. You might also define constraints, like a maximum length of characters in a text value or a range of numeric values, by decorating with validation attributes. These can be read by technologies like ASP.NET Core MVC and Blazor to provide automatic validation warnings to users.

5. Why might you choose Fluent API in preference to annotation attributes?

   Answer: You might choose Fluent API in preference to annotation attributes when you want to keep your entity classes free from extraneous code that is not needed in all scenarios. For example, when creating a .NET Standard 2.0 class library for entity classes, you might want to only use validation attributes so that that metadata can be read by Entity Framework Core and by technologies like ASP.NET Core model binding validation and Windows desktop and mobile apps. However, you might want to use Fluent API to define Entity Framework Core-specific functionality like mapping to a different table or column name.

6. What does a transaction isolation level of `Serializable` mean?

   Answer: Maximum locks are applied to ensure complete isolation from any other processes working with the affected data.

7. What does the `DbContext.SaveChanges` method return?

   Answer: An `int` value for the number of entities affected.

8. What is the difference between eager loading and explicit loading?

   Answer: Eager loading means related entities are included in the original query to the database so that they do not have to be loaded later. Explicit loading means related entities are not included in the original query to the database and they must be explicitly loaded just before they are needed.

9. How should you define an EF Core entity class with annotation attributes to match the following table?

```
CREATE TABLE Employees(
  EmpID INT IDENTITY,
  FirstName NVARCHAR(40) NOT NULL,
  Salary MONEY
)
```

   Answer: Use the following class:

```
public class Employee
{
  [Column("EmpID")]
  public int EmployeeID { get; set; }
  [Required] [StringLength(40)]
  public string FirstName { get; set; }
  [Column(TypeName = "money")]
  public decimal? Salary { get; set; }
}
```

10. What benefit do you get from declaring entity navigation properties as `virtual`?

    Answer: You can enable lazy loading if you declare entity navigation properties as `virtual`.

Chapter 12 – Querying and Manipulating Data Using LINQ

1. What are the two required parts of LINQ?

   Answer: A LINQ provider and the LINQ extension methods. You must import the `System.Linq` namespace to make the LINQ extension methods available and reference a LINQ provider assembly for the type of data that you want to work with.

2. Which LINQ extension method would you use to return a subset of properties from a type?

   Answer: The `Select` method allows projection (selection) of properties.

3. Which LINQ extension method would you use to filter a sequence?

   Answer: The `Where` method allows filtering by supplying a delegate (or lambda expression) that returns a Boolean to indicate whether the value should be included in the results.

4. List five LINQ extension methods that perform aggregation.

   Answer: Any five of the following: `Max`, `Min`, `Count`, `LongCount`, `Average`, `Sum`, and `Aggregate`.

5. What is the difference between the `Select` and `SelectMany` extension methods?

   Answer: `Select` returns exactly what you specify to return. `SelectMany` checks that the items you have selected are themselves `IEnumerable<T>` and then breaks them down into smaller parts. For example, if the type you select is a `string` value (which is `IEnumerable<char>`), `SelectMany` will break each `string` value returned into their individual `char` values.

6. What is the difference between `IEnumerable<T>` and `IQueryable<T>` and how do you switch between them?

   Answer: The `IEnumerable<T>` interface indicates a LINQ provider that will execute the query locally like LINQ to Objects. These providers have no limitations but can be less efficient. The `IQueryable<T>` interface indicates a LINQ provider that first builds an expression tree to represent the query and then converts it into another query syntax before executing it, like Entity Framework Core converts LINQ to SQL. These providers sometimes have limitations and throw exceptions. You can convert from an `IQueryable<T>` provider to an `IEnumerable<T>` provider by calling the `AsEnumerable` method.

7. What does the last type parameter in the generic `Func` delegates represent?

   Answer: The last type parameter in the generic `Func` delegates represents the type of the return value. For example, for `Func<string, int, bool>`, the delegate or lambda function used must return a Boolean value.

8. What is the benefit of a LINQ extension method that ends with `OrDefault`?

   Answer: The benefit of a LINQ extension method that ends with `OrDefault` is that it returns the default value instead of throwing an exception if it cannot return a value. For example, calling the `First` method on a sequence of `int` values would throw an exception if the collection is empty but the `FirstOrDefault` method would return `0`.

9. Why is query comprehension syntax optional?

   Answer: Query comprehension syntax is optional because it is just syntactic sugar. It makes code easier for humans to read but it does not add any additional functionality except the `let` keyword.

More Information: You can read about the `let` keyword at the following link: `https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/let-clause`

10. How can you create your own LINQ extension methods?

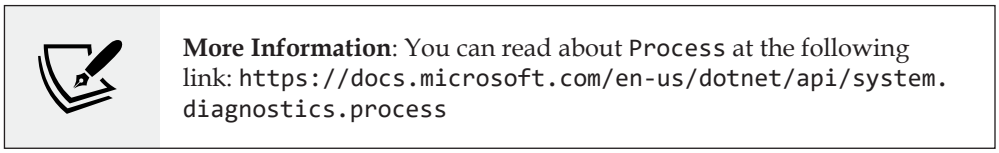
    Answer: Create a `static` class with a `static` method with an `IEnumerable<T>` parameter prefixed with `this`, as shown in the following code:

```
namespace System.LINQ
{
  public static class MyLINQExtensionMethods
  {
    public static IEnumerable<T> MyChainableExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
    // return something IEnumerable<T>
    }
    public static int? MyAggregateExtensionMethod<T>(
      this IEnumerable<T> sequence)
    {
    // return some int value
    }
  }
}
```

Chapter 13 – Improving Performance and Scalability Using Multitasking

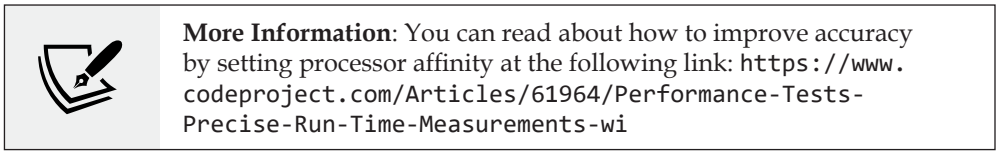
1. What information can you find out about a process?

   Answer: The `Process` class has many properties including: `ExitCode`, `ExitTime`, `Id`, `MachineName`, `PagedMemorySize64`, `ProcessorAffinity`, `StandardInput`, `StandardOutput`, `StartTime`, `Threads`, and `TotalProcessorTime`.



2. How accurate is the `Stopwatch` class?

   Answer: The `Stopwatch` class can be accurate to within a nanosecond (a billionth of a second), but you shouldn't rely on that.



3. By convention, what suffix should be applied to a method that returns `Task` or `Task<T>`?

   Answer: Add the suffix `Async` to the method name, for example, for a synchronous method named `Open`, use `OpenAsync` for the equivalent that returns a `Task`.

4. To use the `await` keyword inside a method, which keyword must be applied to the method declaration?

   Answer: The `async` keyword must be applied to the method declaration.

5. How do you create a child task?

   Answer: Call the `Task.Factory.StartNew` method with the `TaskCreationOptions.AttachToParent` option to create a child task.

6. Why should you avoid the `lock` keyword?

   Answer: The `lock` keyword does not allow you to specify a timeout; this can cause deadlocks. Use the `Monitor.Enter` method and pass a `TimeSpan` argument as a timeout and then call the `Monitor.Exit` method explicitly to release the lock at the end of your work instead.

7. When should you use the `Interlocked` class?

   Answer: You should use the `Interlocked` class to modify integers and floating-point numbers that are shared between multiple threads.

8. When should you use the `Mutex` class instead of the `Monitor` class?

   Answer: Use `Mutex` when you need to share a resource across process boundaries. `Monitor` only works on resources inside the current process.

9. What is the benefit of using `async` and `await` in a website or web service?

   Answer: In a website or web service, using `async` and `await` improves scalability, but not the performance of a specific request because extra work of handling over work between threads is required.

10. Can you cancel a task? How?

    Answer: Yes, you can cancel a task, as described at the following link: `https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/cancel-an-async-task-or-a-list-of-tasks`.

Chapter 15 – Building Websites Using ASP.NET Core Razor Pages

1. List six method names that can be specified in an HTTP request.

   Answer: `GET`, `HEAD`, `POST`, `PUT`, `PATCH`, and `DELETE`. Others include `TRACE`, `OPTIONS`, and `CONNECT`.

More Information: You can read about HTTP method definitions at the following link: `https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html`

2. List six status codes and their descriptions that can be returned in an HTTP response.

   Answer: `200 OK`, `201 Created`, `301 Moved Permanently`, `400 Bad Request`, `404 Not Found` (missing resource), `500 Internal Server Error`. Others include `101 Switching Protocols` (e.g. from HTTP to WebSocket), `202 Accepted`, `204 No Content`, `304 Not Modified`, `401 Unauthorized`, `403 Forbidden`, `406 Not Acceptable` (for example, requesting a response format that is not supported by a website), and `503 Service Unavailable`.

More Information: You can read about status code definitions at the following link: `https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html`

3. In ASP.NET Core, what is the `Startup` class used for?

   Answer: In ASP.NET Core, the `Startup` class is used to add and configure services in the request and response pipeline like error handling, security options, static files, default files, endpoint routing, Razor Pages and MVC, and Entity Framework Core data contexts.

4. What does the acronym HSTS stand for and what does it do?

   Answer: HTTP Strict Transport Security (HSTS) is an opt-in security enhancement. If a website specifies it and a browser supports it, then it forces all communication over HTTPS and prevents the visitor from using untrusted or invalid certificates.

5. How do you enable static HTML pages for a website?

   Answer: To enable static HTML pages for a website, you must add statements to the `Configure` method in the `Startup` class to use default files and then to use static files (this order is important!), as shown in the following code:

```
app.UseDefaultFiles(); // index.html, default.html, and so on
app.UseStaticFiles();
```

6. How do you mix C# code into the middle of HTML to create a dynamic page?

   Answer: To mix C# code into the middle of HTML to create a dynamic page, you can create a Razor file with the `.cshtml` file extension, and then prefix any C# expressions with the `@` symbol, and for C# statements wrap them in braces or create a `@functions` section, as shown in the following markup:

```
@page
@functions
{
  public string[] DaysOfTheWeek
  {
    get => System.Threading.Thread.CurrentThread
      .CurrentCulture.DateTimeFormat.DayNames;
  }

  public string WhatDayIsIt
```

```
    {
      get => System.DateTime.Now.ToString("dddd");
    }
  }
  <html>
  <head>
    <title>Today is @WhatDayIsIt</title>
  </head>
  <body>
    <h1>Days of the week in your culture</h1>
    <ul>
  @{
  // to add a block of statements: use braces
  foreach (string dayName in DaysOfTheWeek)
    {
    <li>@dayName</li>
    }
  }
  </ul>
  </body>
  </html>
```

7. How can you define shared layouts for Razor Pages?

   Answer: To define shared layouts for Razor Pages, create at least two files: `_Layout.cshtml` will define the markup for the shared layout, and `_ViewStart.cshtml` will set the default layout, as shown in the following markup:

   ```
   @{
     Layout = "_Layout";
   }
   ```

8. How can you separate the markup from the code behind in a Razor Page?

   Answer: To separate the markup from the code behind in a Razor Page, create two files: `MyPage.cshtml` contains the markup and `MyPage.cshtml.cs` contains a class that inherits from `PageModel`. In `MyPage.cshtml`, set the model to use the class, as shown in the following markup:

   ```
   @page
   @model MyProject.Pages.MyPageModel
   ```

9. How do you configure an Entity Framework Core data context for use with an ASP.NET Core website?

   Answer: To configure an Entity Framework Core data context for use with an ASP.NET Core website:

   - In the project file, reference the assembly that defines the data context class.
   - In the `Startup` class, import the namespaces for `Microsoft.EntityFrameworkCore` and the data context class.

- In the `ConfigureServices` method, add a statement that configures the data context with a database connection string for use with a specified database provider like SQLite, as shown in the following code:

```
services.AddDbContext<MyDataContext>(options => options.UseSqlite("my
database connection string"));
```

- In the Razor Page model class or `@functions` section, declare a private field to store the data context and then set it in the constructor, as shown in the following code:

```
private MyDataContext db;

public SuppliersModel(MyDataContext injectedContext)
{
  db = injectedContext;
}
```

10. How can you reuse Razor Pages with ASP.NET 2.2 or later?

    Answer: To reuse Razor Pages with ASP.NET 2.2 or later, everything related to a Razor page can be compiled into a class library. To create one, enter the following command: `dotnet new razorclasslib -s`.

Chapter 16 – Building Websites Using the Model-View-Controller Pattern

1. What do the files with the special names `_ViewStart` and `_ViewImports` do when created in the `Views` folder?

   Answer:

   - A `ViewStart` file contains a block of statements that are executed when the `View` method is executed when a controller action method passes a model to a view, for example, to set a default layout.
   - A `ViewImports` file contains `@using` statements to import namespaces for all views to avoid having to add the same import statements at the top of all views.

2. What are the names of the three segments defined in the default ASP.NET Core MVC route, what do they represent, and which are optional?

   Answer:

   - `{controller}`: For example, `/home` represents a controller class to instantiate, for example, `HomeController`. It is optional because it can use the default value: `Home`.
   - `{action}`: For example, `/index` represents an action method to execute, for example, `Index`. It is optional because it can use the default value: `Index`.

- `{id}`: For example, `/5` represents a parameter in the action method, for example, `int id`. It is optional because it is suffixed with `?`.

3. What does the default model binder do and what data types can it handle?

   Answer: The default model binder sets parameters in the action method. It can handle the following data types:

   - Simple types like `int`, `string`, and `DateTime`, including nullable types.
   - Complex types like `Person` and `Product`.
   - Collections types like `IEnumerable<T>`.

4. In a shared layout file like `_Layout.cshtml`, how do you output the content of the current view?

   Answer: To output the content of the current view in a shared layout, call the `RenderBody` method, as shown in the following markup:

   ```
   @RenderBody()
   ```

5. In a shared layout file like `_Layout.cshtml`, how do you output a section that the current view can supply content for, and how does the view supply the contents for that section?

   Answer: To output the content of a section in a shared layout, call the `RenderSection` method specifying a name for the section and if it is required, as shown in the following markup:

   ```
   @RenderSection("Scripts", required: false)
   ```

   To define the contents of the section in the view, create a named section, as shown in the following markup:

   ```
   @section Scripts
   {
     <script> alert('hello');
     </script>
   }
   ```

6. When calling the `View` method inside a controller's action method, what paths are searched for the view by convention?

   Answer: When calling the `View` method inside a controller's action method, three paths are searched for the view by default, based on combinations of the names of the controller and action method and a special `Shared` folder, as shown in the following example output:

   ```
   InvalidOperationException: The view 'Index' was not found. The following
   locations were searched:
   /Views/Home/Index.cshtml
   /Views/Shared/Index.cshtml
   /Pages/Shared/Index.cshtml
   ```

This can be generalized as follows:

- `/Views/[controller]/[action].cshtml`
- `/Views/Shared/[action].cshtml`
- `/Pages/Shared/[action].cshtml`

7. How can you instruct the visitor's browser to cache the response for 24 hours?

   Answer: To instruct the visitor's browser to cache the response for 24 hours, decorate the controller class or action method with the `[ResponseCache]` attribute, and set the `Duration` parameter to `86400` seconds and the `Location` parameter to `ResponseCacheLocation.Client`.

8. Why might you enable Razor Pages even if you are not creating any yourself?

   Answer: If you have used features like ASP.NET Core Identity UI, then it requires Razor Pages.

9. How does ASP.NET Core MVC identify classes that can act as controllers?

   Answer: ASP.NET Core MVC identifies classes that can act as controllers by looking to see if the class (or a class that it derives from) is decorated with the `[Controller]` attribute.

10. In what ways does ASP.NET Core MVC make it easier to test a website?

    Answer: The **Model-View-Controller** (**MVC**) design pattern separates the technical concerns of the shape of the data (model) and executable statements to process the incoming request and outgoing response, and then generates the response in a format requested by the user agent like HTML or JSON. This makes it easier to write unit tests. ASP.NET Core also makes it easy to implement the **Inversion-of-Control** (**IoC**) and **Dependency Injection** (**DI**) design patterns to remove dependencies when testing a component like a controller.

Chapter 17 – Building Websites Using a Content Management System

1. What are some of the benefits of using a Content Management System to build a website compared to using ASP.NET Core alone?

   Answer: A CMS separates the content (data values) from templates (layout, format, and style). A CMS provides a content management user interface so that non-technical users can manage the content quickly and easily while still providing a professional-looking website.

   An enterprise-level CMS would also provide features like SEO URLs, fine-level control over authentication and authorization, media asset management, various ways to reuse content, a visual forms designer, various ways to personalize content, and support for multiple versions and languages for content.

2. What is the special relative URL path to access the Piranha CMS management user interface and what is the username and password configured by default?

   Answer: The special relative URL path to access the Piranha CMS management user interface is `/manager` and the username and password configured by default is `admin` and `password`.

3. What is a slug?

   Answer: A slug is the relative URL path for a content item like a page or post.

4. What is the difference between saving content and publishing content?

   Answer: The difference between saving content and publishing content is that saving just saves the changes to the CMS database but publishing also makes those changes visible to visitors.

5. What are the three Piranha CMS content types and what are they used for?

   Answer: The three Piranha CMS content types are Sites, Pages, and Posts. Sites are for property values that need to be shared across all pages. Pages are for normal web pages. Posts are for special pages that can only be shown in an archive page with sorting and filtering capabilities.

6. What are the three Piranha CMS component types and what are they used for?

   Answer: The three Piranha CMS component types are Fields, Regions, and Blocks. Fields are for simple data values like strings and numbers. Regions are for complex data values that appear at a fixed location with a content type like a page. Blocks are for complex data values that appear in any order and combination defined by the content owner.

7. List three properties that a Page type inherits from its base classes and explain what they are used for.

   Answer: Three properties that a Page type inherits from its base classes include:

   - `ParentId`: A `Guid` value that references its parent with the content tree.
   - `Blocks`: An ordered collection of references to block instances.
   - `Published`: A `DateTime` value that shows when the page was published for visitors to view.

8. How do you define a custom region for a content type?

   Answer: You define a custom region for a page type by creating a class with properties decorated with the `[Field]` attribute, and then add a property to the content type with the region class as its type and decorate it with the `[Region]` and `[RegionDescription]` attributes, as shown in the following code:

```
namespace NorthwindCms.Models
{
  [PageType(Title = "Category Page", UseBlocks = false)]
  [PageTypeRoute(Title = "Default", Route = "/category")]
  public class CategoryPage : Page<CategoryPage>
  {
    [Region(Title = "Category detail")]
    [RegionDescription("The details for this category.")]
    public CategoryRegion CategoryDetail { get; set; }
```

9. How do you define routes for Piranha CMS?

   Answer: To define routes for Piranha CMS, in the `CmsController` class, add an action method and decorate it with the `[Route]` attribute, as shown in the following code:

```
[Route("category")]
public IActionResult Category(Guid id)
```

10. How do you retrieve a page from the Piranha CMS database?

    Answer: You retrieve a page from the Piranha CMS database by using the `IApi` dependency service, as shown in the following code:

```
var model = await _api.Pages.GetByIdAsync<Models.CategoryPage>(id);
```

Chapter 18 – Building and Consuming Web Services

1. Which class should you inherit from to create a controller class for an ASP.NET Core Web API service?

   Answer: To create a controller class for an ASP.NET Core Web API service, you should inherit from `ControllerBase`. Do not inherit from `Controller` as you would in MVC because this class includes methods like `View` that use Razor files to render HTML that are not needed for a web service.

2. If you decorate your controller class with the `[ApiController]` attribute to get default behavior like automatic 400 responses for invalid models, what else must you do?

   Answer: If you decorate your controller class with the `[ApiController]` attribute, then you must also call the `SetCompatibilityVersion` method in the `Startup` class.

3. What must you do to specify which controller action method will be executed in response to an HTTP request?

   Answer: To specify which controller action method will be executed in response to a request, you must decorate the action method with an attribute. For example, to respond to an HTTP POST request, decorate the action method with `[HttpPost]`.

4. What must you do to specify what responses should be expected when calling an action method?

   Answer: To specify what responses should be expected when calling an action method, decorate the action method with the `[ProducesResponseType]` attribute, as shown in the following code:

```
// GET: api/customers/[id]
[HttpGet("{id}", Name = nameof(Get))] // named route
[ProducesResponseType(200, Type = typeof(Customer))]
[ProducesResponseType(404)]
public IActionResult Get(string id)
{
```

5. List three methods that can be called to return responses with different status codes.

   Answer: Three methods that can be called to return responses with different status codes include:

   - `Ok`: This returns the 200 status code and the object passed to `Ok` in the body.
   - `CreatedAtRoute`: This returns the 201 status code and the object passed to this method in the body.
   - `NoContentResult`: This returns the 204 status code and an empty body.
   - `BadRequest`: This returns the 400 status code and an optional error message.
   - `NotFound`: This returns the 404 status code and an optional error message.

6. List four ways that you can test a web service.

   Answer: Four ways that you can test a web service include:

   - Using a browser to test simple HTTP GET requests.
   - Installing the REST Client extension for Visual Studio Code.
   - Installing the Swagger NuGet package in your web service project, enabling Swagger, and then using the Swagger testing user interface.
   - Installing the Postman tool from the following link: `https://www.getpostman.com`.

7. Why should you not wrap your use of `HttpClient` in a `using` statement to dispose of it when you are finished even though it implements the `IDisposable` interface, and what should you use instead?

   Answer: `HttpClient` is shared, reentrant, and partially thread-safe so it is tricky to use correctly in many scenarios. You should use `HttpClientFactory`, which was introduced in .NET Core 2.1.

8. What does the acronym CORS stand for and why is it important to enable it in a web service?

   Answer: The acronym CORS stands for Cross-Origin Resource Sharing. It is important to enable it for a web service because default browser same-origin policy prevents code downloaded from one origin from accessing resources downloaded from a different origin to improve security.

9. How can you enable clients to detect if your web service is healthy with ASP.NET Core 2.2 and later?

   Answer: To enable clients to detect if your web service is healthy, you can install health check APIs including database health checks for Entity Framework Core data contexts. Health checks can be extended to report detailed information back to the client.

10. What benefits does endpoint routing provide?

    Answer: Endpoint routing provides improved performance of routing and action method selection, and a link generation service.

Chapter 19 – Building Intelligent Apps Using Machine Learning

1. What are the four main steps of the machine learning lifecycle?

   Answer: The four main steps of the machine learning lifecycle are: Problem analysis, Data gathering and processing, Modeling, and Deploying the model.

2. What are the three sub-steps of the modeling step?

   Answer: The three sub-steps of the modeling step are: Identifying features, Training the model, and Evaluating the model.

3. Why do models need to be retrained after deployment?

   Answer: Models need to be retrained after deployment because the predictions they make could drift and become poorer because data can change over time.

4. Why must you split your dataset into a training dataset and a testing dataset?

   Answer: You must split your dataset into a training dataset and a testing dataset because if you use the whole dataset for training, then you have no data left over to test your model and you cannot use the same dataset for both because it would work perfectly!

5. What are some of the differences between clustering and classification machine learning tasks?

   Answer: Some of the differences between clustering and classification

machine learning tasks include the following:

- Clustering is for grouping data where you do not yet know if there are any commonalities or what the labels should be. It is an unsupervised machine learning technique.
- Classification is for allocating data to predefined labeled groups. It is a supervised machine learning technique.

6. What class must you instantiate to perform any machine learning task?

   Answer: You must instantiate an `MLContext` to perform any machine learning task.

7. What is the difference between a label and a feature?

   Answer: A feature is an input, for example, the tokenized text of an Amazon review. A label is a value used to train the model and can be an output, for example, positive or negative.

8. What does `IDataView` represent?

   Answer: `IDataView` represents the input data for a machine learning task. It is immutable, cursorable, lazily evaluated, heterogenous, and schematized.

9. What does the `count` parameter in the `[KeyType(count: 10)]` attribute represent?

   Answer: The `count` parameter represents the maximum possible value that can be stored in that column.

10. What does the score represent with matrix factorization?

    Answer: The score with matrix factorization represents the likelihood of being a positive case but it is not a probability in the traditional sense. The larger the score value, the higher the likelihood.

Chapter 20 – Building Web User Interfaces Using Blazor

1. What are the two primary hosting models for Blazor and how are they different?

   Answer: The two primary hosting models for Blazor are Server and WebAssembly.

   Blazor Server executes code on the server-side, which means the code has full and easy access to server-side resources like databases. This can simplify implementing functionality. UI updates are made using SignalR, which means a permanent connection is needed between the browser and server, which limits scalability.

   Blazor WebAssembly executes code on the client-side, which means the code only has access to resources within the browser. This can complicate the implementation because a call back to the server must be made whenever new data is required.

2. In a Blazor Server website project, compared to an ASP.NET Core MVC website project, what extra configuration is required in the Startup class?

   Answer: In the `Startup` class, in the `ConfigureServices` method you must call `AddServerSideBlazor`, and in the `Configure` method you must call `MapBlazorHub` and `MapFallbackToPage` when setting up endpoints.

3. One of the benefits of Blazor is being able to implement client-side components using C# and .NET instead of JavaScript. Does a Blazor component need any JavaScript?

   Answer: Yes, Blazor components need some minimal JavaScript. For Blazor Server this is provided by the file `_framework/blazor.server.js`. For Blazor WebAssembly this is provided by the file `_framework/blazor.webassembly.js`. Blazor WebAssembly with PWA also uses a JavaScript service worker file `service-worker.js`.

4. In a Blazor Server website project, what does the `App.razor` file do?

   Answer: The `App.razor` file configures a Router used by all Blazor components in the current assembly. For example, it sets a default shared layout for components that match a route and a view to use when no match is found.

5. What is a benefit of using the `<NavLink>` component?

   Answer: A benefit of using the `<NavLink>` component is that it integrates with the Blazor routing system so that the `NavigationManager` can then be used to programmatically navigate between components.

6. How can you pass a value into a component?

   Answer: You can pass a value into a component by decorating a public property in the component with the `[Parameter]` attribute and then setting the attribute in the component when using it, as shown in the following code:

```
// defining the component
@code {
  [Parameter]
  public string ButtonText { get; set; };

// using the component
<CustomerDetail ButtonText="Create Customer" />
```

7. What is a benefit of using the `<EditForm>` component?

   Answer: A benefit of using the `<EditForm>` component is automatic validation messages.

8. How can you execute some statements when parameters are set?

   Answer: You can execute some statements when parameters are set by defining an `OnParametersSetAsync` method to handle that event.

9. How can you execute some statements when a component appears?

   Answer: You can execute some statements when a component appears by defining an `OnInitializedAsync` method to handle that event.

10. What are two key differences in the `Program` class between a Blazor Server and Blazor WebAssembly project?

    Answer: Two key differences in the `Program` class between a Blazor Server and Blazor WebAssembly project are: (1) the use of `WebAssemblyHostBuilder` instead of `Host.CreateDefaultBuilder`, and (2) the registering of an `HttpClient` with a base address of the host environment.

Chapter 21 – Building Cross-Platform Mobile Apps Using Xamarin.Forms

1. What is the difference between Xamarin and Xamarin.Forms?

   Answer: Xamarin enables developers to build desktop and mobile apps using C# for Apple iOS (iPhone) and iPadOS, or for macOS, or for Google Android. Some business logic can be shared but you are basically creating separate projects for each platform. Xamarin.Forms extends Xamarin to make cross-platform mobile development even easier by sharing most of the user experience layer.

2. What are the four categories of Xamarin.Forms user interface components and what do they represent?

   Answer: The four categories of Xamarin.Forms user interface components are:

   - Pages: This represents mobile application screens.
   - Layouts: This represents the structure of a combination of the user interface components.
   - Views: This represents a single user interface component.
   - Cells: This represents a single item in a list or table view.

3. List four types of cell.

   Answer: Four types of cell are `TextCell`, `SwitchCell`, `EntryCell`, and `ImageCell`.

4. How can you enable a user to perform an action on a cell in a list view?

   Answer: To enable a user to perform an action on a cell in a list view, you can set some context actions that are menu items that raise an event, as shown in the following markup:

```
<TextCell Text="{Binding CompanyName}" Detail="{Binding Location}">
  <TextCell.ContextActions>
    <MenuItem Clicked="Customer_Phoned" Text="Phone" />
    <MenuItem Clicked="Customer_Deleted" Text="Delete"
      IsDestructive="True" />
  </TextCell.ContextActions>
</TextCell>
```

5. How do you define a dependency service to implement platform-specific functionality?

   Answer: To define a dependency service to implement platform-specific functionality:

   - Define an interface for the dependency service.
   - Implement the dependency service for each platform, for example, iOS and Android.
   - Decorate the dependency service with the `[assembly: Dependency]` attribute and pass the type of the dependency service as a parameter.
   - In the shared main project, get the dependency service, as shown in the following code:

```
var service = DependencyService.Get<IMyService>();
```

6. When would you use an `Entry` instead of an `Editor`?

   Answer: Use an `Entry` for a single line of text and use an `Editor` for multiple lines of text.

7. What is the effect of setting `IsDestructive` to `true` for a menu item in a cell's context actions?

   Answer: The menu item is colored red as a warning to the user.

8. When would you call the methods `PushAsync` and `PopAsync` in a Xamarin.Forms mobile app?

   Answer: To provide navigation between screens with built-in support to go back to the previous screen, wrap the first screen in a `NavigationPage` when the app first starts, as shown in the following code:

```
MainPage = new NavigationPage(new CustomersList());
```

   To go to the next screen, push the next page onto the `Navigation` object, as shown in the following code:

```
await Navigation.PushAsync(new CustomerDetails(c));
```

   To return to the previous screen, pop the page from the `Navigation` object, as shown in the following code:

```
await Navigation.PopAsync();
```

9. How do you show a pop-up modal message with simple button choices like Yes or No?

   Answer: To show a pop-up modal message with simple button choices like Yes or No, you can call the `DisplayAlert` method, as shown in the following code:

```
bool response = await DisplayAlert("Apple Survey",
  "Do you like your iPhone 11 Pro?", "Yes", "No");
```

10. What is Apple's ATS and why is it important?

    Answer: Apple's ATS is App Transport Security and it is important because it is on by default and it forces developers to use good practice including secure connections between an app and a web service.

Appendix B – Building Windows Desktop Apps

1. .NET 5 is cross-platform. Windows Forms and WPF apps can run on .NET 5. Can those apps therefore run on macOS and Linux?

   Answer: No. Although Windows Forms and WPF apps can run on .NET 5, they also need to make calls to Win32 APIs and so are limited to running on Windows.

2. How does a Windows Forms app define its user interface and why is this a potential problem?

   Answer: A Windows Forms app defines its user interface using C# code. This is why it is important to use a Windows Forms visual designer with a toolbox and drag-and-drop support. It is messy to edit the generated code directly.

3. How can a WPF or UWP app define its user interface and why is this good for developers?

   Answer: A WPF or UWP app can define its user interface using XAML, which is easier than C# code for developers to understand and write, even without a visual designer.

4. List five layout containers and how their child elements are positioned within them.

   Answer: Five layout containers are as follows:

   - `StackPanel`: Child elements are positioned by stacking them horizontally or vertically.
   - `Grid`: Child elements are positioned by `Row` and `Column` within the defined grid.
   - `DockPanel`: The single child element is positioned within its parent container by aligning it to the `Top`, `Left`, `Bottom`, `Right`, or by filling the remaining space.
   - `WrapPanel`: Child elements are positioned by stacking them horizontally or vertically and then wrapping to another column or row if there are more to show.
   - `Canvas`: Child elements are positioned using absolute `Top` and `Left` or `Bottom` and `Right` values.

5. What is the difference between `Margin` and `Padding` for an element like a `Button`?

   Answer: The difference between `Margin` and `Padding` for an element like a `Button` is that `Margin` is outside the `Border` and `Padding` is inside the `Border`.

6. How are event handlers attached to an object using XAML?

   Answer: Event handlers are attached to an object using XAML by setting an attribute for the event name to the name of a method in the code-behind class, as shown in the following markup:

   ```
   <Button Clicked="SaveButton_Clicked">
   ```

7. What do XAML styles do?

   Answer: XAML styles enable the setting of one or more properties.

8. Where can you define resources?

   Answer: You can define resources in any element depending on where you want to share those resources. To share resources throughout an app, define resources in the `<Application.Resources>` element. To share resources only within a page, define resources in its `<Page.Resources>` element.

   To use resources in a single element like a button, define resources in its `<Button.Resources>` element.

9. How can you bind the property of one element to a property on another element?

   Answer: To bind the property of one element to a property on another element, use a binding expression for the property that references the other element and its property, as shown in the following markup:

   ```
   <Slider Value="18" Minimum="18" Maximum="65" Name="slider" />
   <TextBlock Text="{Binding ElementName=slider, Path=Value}" />
   ```

10. Why might you implement the `IValueConverter` interface?

    Answer: You might implement the `IValueConverter` interface when you want to perform data binding between two incompatible types, for example, to convert an `int` value into a `string` value that defines a path to an image, or between a floating-point value representing a **body-mass index** (**BMI**) and a color that indicates good or bad health.

Appendix B

Building Windows Desktop Apps

This appendix is about building applications for Windows desktop using three technologies: **Windows Forms**, **Windows Presentation Foundation** (**WPF**), and **Universal Windows Platform** (**UWP**).

Windows Forms and WPF support using .NET Core 3.0 or later as their runtime, but current design-time support is limited so I only recommend this if you have existing Windows Forms or WPF apps that must be migrated to modern .NET.

Most of this appendix will cover UWP apps that use the modern Windows Runtime and can execute apps built using a custom version of .NET that compiles to native CPU instructions.

Although most of this appendix does not use .NET 5, in November 2021 Microsoft will release .NET 6, which will be the single unified platform for .NET used by all app models, including ASP.NET Core for web development, Windows Forms, WPF, and also cross-platform desktop and mobile apps using .NET **Multi-platform App User Interface** (**MAUI**).

More Information: You can read more about your choices of platform for building Windows desktop apps at the following link: `https://docs.microsoft.com/en-us/windows/apps/desktop/choose-your-platform`.

You will explore some of the new user interface features of Fluent Design, introduced in the Windows 10 Fall Creators Update in October 2017.

In a single appendix, we will only be able to scratch the surface of what can be achieved using .NET for the Windows desktop. However, I hope to excite you into wanting to learn more about the new ability to migrate legacy Windows applications to the modern .NET platform, and the cool new UWP technology with Fluent Design, including template-able controls, data binding, and animation!

Some important points about this appendix

UWP apps are not cross-platform, but they are cross-device, meaning they can run on desktop, laptop, tablet, and mixed reality devices like the HP Windows Mixed Reality Headset. Those devices must run a modern flavor of Windows. You will need Windows 10 May 2019 Update and Visual Studio 2019 version 16.3 or later to use the latest features, like XAML Islands, although older versions of Windows 10, like the April 2018 Update, should work for creating the example app in this appendix.

In this appendix, we will cover the following topics:

- Understanding legacy Windows application platforms
- Understanding the modern Windows platform
- Creating a modern Windows app
- Using resources and templates
- Using data binding

Understanding legacy Windows application platforms

With the Microsoft Windows 1.0 release in 1985, the only way to create Windows applications was to use the C language and call functions in three core **DLLs** named kernel, user, and GDI. Once Windows became 32-bit with Windows 95, the DLLs were suffixed with 32 and became known as the Win32 API.

In 1991, Microsoft introduced Visual Basic, which provided developers with a visual, drag-and-drop-from-a-toolbox-of-controls way to build the user interface for Windows applications. It was immensely popular, and the Visual Basic runtime is still part of Windows 10 today.

In 2002, Microsoft introduced .NET Framework, which included Windows Forms for building Windows applications. The code could be written in either Visual Basic or C# languages. Windows Forms had a similar drag and drop visual designer, although it generated C# or Visual Basic code to define the user interface, which can be difficult for humans to understand and edit directly.

In 2006, Microsoft introduced .NET Framework 3.0, which included WPF for building user interfaces using XAML, which is easy for developers to understand and edit directly.

Understanding .NET 5 support for legacy Windows platforms

The on-disk size of the .NET 5 SDKs for Linux and macOS are 332 MB and 337 MB, respectively. The on-disk size of the .NET 5 SDK for Windows is 441 MB. This is because it includes the Windows Desktop Runtime, which allows the legacy Windows application platforms Windows Forms and WPF to be run on .NET 5.

Installing Microsoft Visual Studio 2019 for Windows

Since October 2014, Microsoft has made a professional-quality edition of Visual Studio available to everyone for free. It is called Community Edition.

If you have not already installed it, let's do so now:

1. Download Microsoft Visual Studio 2019 version 16.8 or later for Windows from the following link: `https://visualstudio.microsoft.com/downloads/`
2. Start the installer.
3. On the **Workloads** tab, select the following:
    - **.NET desktop development**
    - **Universal Windows Platform development**
    - **.NET Core cross-platform development**
4. Click **Install** and wait for the installer to acquire the selected software and install it.
5. When the installation is complete, click **Launch**.
6. The first time that you run Visual Studio, you will be prompted to sign in. If you have a Microsoft account, you can use that account. If you don't, then register for a new one at the following link: `https://signup.live.com/`
7. The first time that you run Visual Studio, you will be prompted to configure your environment. For **Development Settings**, choose **Visual C#**. For the color theme, I chose **Blue**, but you can choose whatever tickles your fancy.

Working with Windows Forms

We will start by using the `dotnet` command tool to create a new Windows Forms app, then we will use Visual Studio to create a Windows Forms app for .NET Framework to simulate a legacy application, and then we will migrate that application to .NET 5.

Building a new Windows Forms application

In this task, you will see that it is not a good idea to build new Windows Forms applications for .NET 5. As a general rule, only migrate existing Windows Forms applications to .NET.

The following instructions start with creating some new folders to store a solution file and a project. You can choose to use another tool to do this instead of Command Prompt, but you should at least create the new solution and Windows Forms project by using the `dotnet` command-line tool in the correct folder:

1. From the Windows Start menu, open **Command Prompt**.
2. In Command Prompt, enter commands to perform the following tasks:
    - Change to the `Code` folder.
    - Create a folder named `WindowsDesktopApps` with a new solution file.

- Create a subfolder named `BasicWinForms` with a new Windows Forms project:

```
cd C:\Code\
mkdir WindowsDesktopApps
cd WindowsDesktopApps
dotnet new sln
mkdir BasicWinForms
cd BasicWinForms
dotnet new winforms
```

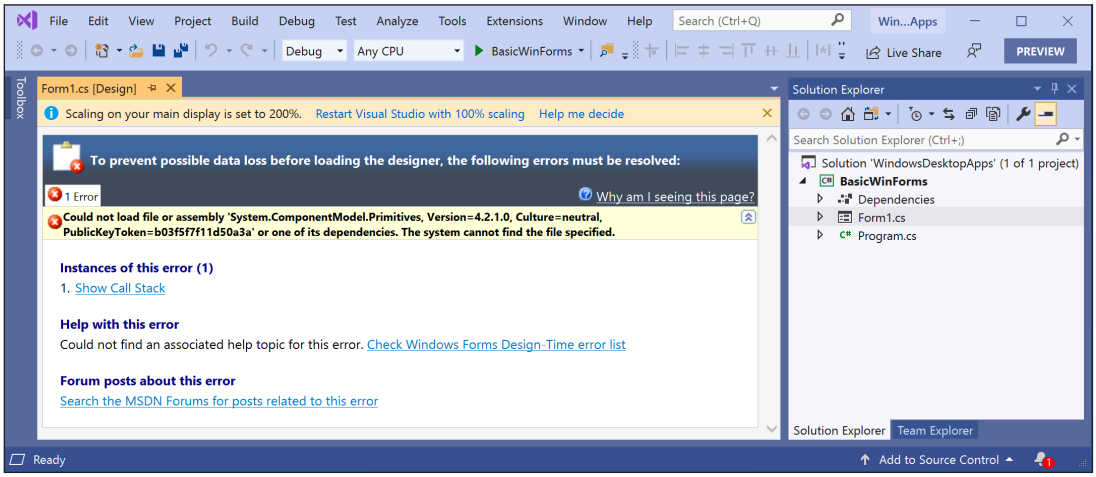
3. Start Visual Studio 2019 and click **Open a project or solution**.
4. Navigate to the `C:\Code\WindowsDesktopApps` folder, select the `WindowsDesktopApps.sln` solution file, and click **Open**.
5. Navigate to **File | Add | Existing Project...**, select the project named `BasicWinForms.csproj`, and then click **Open**.
6. Navigate to **Project | Edit Project File** or right-click the project in **Solution Explorer** and choose **Edit Project File**, and then note that the target framework is `net5.0-windows`, it uses `Windows Forms`, and the output type is a Windows executable instead of a console executable, as shown in the following markup:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net5.0-windows</TargetFramework>
    <UseWindowsForms>true</UseWindowsForms>
  </PropertyGroup>

</Project>
```

7. Close the tab.
8. In **Solution Explorer**, double-click or right-click `Form1.cs` and select **Open**, and note that at the time of writing in September 2020, the Windows Forms designer does not, by default, support .NET 5 (although the 16.8 version expected in November 2020 should work), as shown in the following screenshot:



More Information: You can track the progress of the Windows Forms designer at the following link: `https://github.com/dotnet/winforms/tree/master/docs/designer-releases`

9. Navigate to **Debug | Start Without Debugging** or press *Ctrl + F5*, note the resizable window with the title `Form1`, and then click the **Close** button in its top-right corner to exit the application.

Reviewing a new Windows Forms application

Let's review the code in an empty Windows Forms application:

1. In **Solution Explorer**, open `Program.cs` and note that it is similar to a console app with a `Main` entry point method, and that it instantiates a `Form1` class and runs it, as shown in the following code with comments removed to save space:

```
static class Program
{
  [STAThread]
  static void Main()
  {
    Application.SetHighDpiMode(HighDpiMode.SystemAware);
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Application.Run(new Form1());
  }
}
```

2. In **Solution Explorer**, expand `Form1.cs`, open the `Form1` class, and note it is a partial class and the call to `InitializeComponent` in its constructor, as shown in the following code:

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
  }
}
```

3. In **Solution Explorer**, expand `Form1.cs`, open `Form1.Designer.cs`, expand the **Windows Form Designer generated code** section, and note that the code would be messy to understand and modify manually, as shown in the following code, which defines an empty form:

```
private void InitializeComponent()
{
  this.components = new System.ComponentModel.Container();
  this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
  this.ClientSize = new System.Drawing.Size(800, 450);
  this.Text = "Form1";
}
```

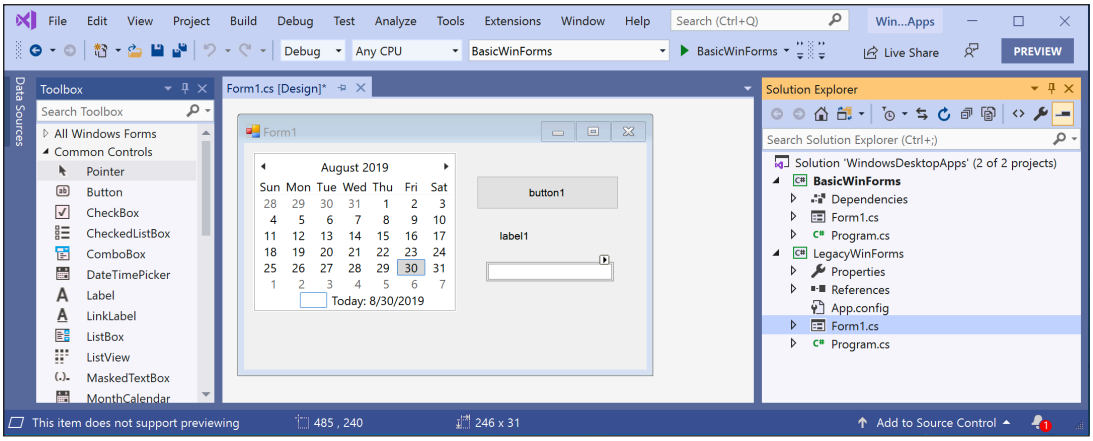
4. Close any edit windows.

Migrating a legacy Windows Forms application

In this task, you will add a Windows Forms application for .NET Framework to the solution so that you can use the Windows Forms visual designer, and then migrate it to .NET 5:

1. In Visual Studio 2019, navigate to **File | Add | New Project...**.
2. In the search box, enter `windows forms` and select **Windows Forms App (.NET Framework)**, and then click **Next**.
3. For **Project name**, enter `LegacyWinForms` and click **Create**.
4. Navigate to **View | Toolbox** or press *Ctrl* + *W*, *X* and then pin the Toolbox. Note that you might have different key bindings. If so, then you can navigate to **Tools | Import and Export Settings** and reset your settings to **Visual C#** to use the same key bindings as the ones I have used in this appendix.
5. In **Toolbox**, expand **Common Controls**.

6. Drag and drop some controls, like **Button**, **MonthCalendar**, **Label**, and **TextBox**, onto **Form1**, as shown in the following screenshot:



7. Navigate to **View | Properties Window** or press *Ctrl + W, P* or *F4*.
8. Select the button, change its (**Name**) property to `btnGoToChristmas`, and change its **Text** property to `Go to Christmas`.
9. Double-click the button, note that an event handler method is written for you, and enter statements to set the calendar to select Christmas Day 2020, as shown in the following code:

```
private void btnGoToChristmas_Click(object sender, EventArgs e)
{
  DateTime christmas = new DateTime(2020, 12, 25);
  monthCalendar1.SelectionStart = christmas;
  monthCalendar1.SelectionEnd = christmas;
}
```

10. In **Solution Explorer**, right-click the solution and select **Set StartUp Projects….**
11. In the **Solution 'WindowsDesktopApps' Properties Pages** dialog box, for **Startup Project**, select **Current selection**, and then click **OK**.
12. In **Solution Explorer**, select the **LegacyWinForms** project and note its name becomes bold to indicate that it is the current selection.
13. Navigate to **Debug | Start Without Debugging** or press *Ctrl + F5*, click the button, and note the month calendar animates to select **Christmas Day**, and then click the **close** button in the top-right corner to exit the app.

Migrating a Windows Forms app

Now, we can migrate this to the .NET 5 project. Remember that we only need to do this as a temporary measure. Once the Windows Forms designer currently in preview is ported to .NET 5 and is available in a future production release for Visual Studio 2019, migrating won't be necessary:

1. Drag and drop `Form1` from the **LegacyWinForms** folder into the **BasicWinForms** folder, which will prompt you to overwrite the following files:
    - `Form1.cs`
    - `Form1.Designer.cs`
    - `Form1.resx`
2. In the `BasicWinForms` project, modify `Program.cs` to specify the legacy namespace when instantiating and running `Form1`, as shown in the following code:

```
Application.Run(new LegacyWinForms.Form1());
```

3. In **Solution Explorer**, select the **BasicWinForms** project.
4. Navigate to **Debug | Start Without Debugging** or press *Ctrl + F5*, click the button, note the month calendar animates to select **Christmas Day**, and then close the app.
5. Close the project and solution.

Migrating WPF apps to .NET 5

If you have existing WPF apps that need to move from .NET Framework to .NET 5, then like migrating Windows Forms projects, it is possible today, but tricky. This will improve over the next year or so, especially with future releases of Visual Studio 2019.

More Information: You can read more about migrating WPF apps at the following link: `https://devblogs.microsoft.com/dotnet/migrating-a-sample-wpf-app-to-net-core-3-part-1/`

Migrating legacy apps using the Windows Compatibility Pack

The Windows Compatibility Pack provides access to APIs that were previously available only for .NET Framework.

More Information: You can read more about the Windows Compatibility Pack at the following link: `https://devblogs.microsoft.com/dotnet/announcing-the-windows-compatibility-pack-for-net-core/`

Understanding the modern Windows platform

Microsoft continues to improve the Windows platform, which includes technologies like **Universal Windows Platform** (**UWP**) and Fluent Design System for building modern apps.

Understanding Universal Windows Platform

UWP is Microsoft's latest technology solution to build applications for its Windows suite of operating systems. It provides a guaranteed API layer across multiple device types. You can create a single app package that can be uploaded to a single store to be distributed to reach all the device types your app can run on. These devices include Windows 10 desktops, laptops, and tablets, the Xbox One and later video game systems, and Mixed Reality Headsets like Microsoft HoloLens.

UWP provides standard mechanisms to detect the capabilities of the current device and then activate additional features of your app to fully take advantage of them.

UWP with XAML provides layout panels that adapt how they display their child controls to make the most of the device they are currently running on. It is the Windows app equivalent of web page responsive design. They also provide visual state triggers to alter the layout based on dynamic changes, such as the horizontal or vertical orientation of a tablet.

Understanding Fluent Design System

Microsoft's Fluent Design System will be delivered in multiple waves, rather than as a "Big Bang" all in one go, to help developers slowly migrate from traditional styles of user interface to more modern ones.

Wave 1, available in Windows 10 Fall Creators Update, released in October 2017 and refined in the subsequent waves as part of biannual Windows 10 updates, included the following features:

- Acrylic material
- Connected animations
- Parallax views
- Reveal lighting

Filling user interface elements with acrylic brushes

Acrylic material is a semi-transparent blur-effect brush that can be used to fill user interface elements to add depth and perspective to your apps. Acrylic can show through what is in the background behind the app, or elements within the app that are behind a pane. Acrylic material can be customized with varying colors and transparencies.

More Information: You can read more about how and when to use Acrylic material at the following link: `https://docs.microsoft.com/en-us/windows/uwp/design/style/acrylic`

Connecting user interface elements with animations

When navigating around a user interface, animating elements to draw connections between screens helps users to understand where they are and how to interact with your app.

More Information: You can read more about how and when to use connected animations at the following link: `https://docs.microsoft.com/en-us/windows/uwp/design/motion/connected-animation`

Parallax views and Reveal lighting

Parallax views are backgrounds, often images, that move at a slower rate than the foreground during scrolling to provide a feeling of depth and give your apps a modern feel.

More Information: You can read more about Parallax at the following link: `https://docs.microsoft.com/en-us/windows/uwp/design/motion/parallax`

Reveal lighting helps the user understand which of the visual elements in the user interface is an interactive element by *lighting up* as the user moves their mouse cursor over that element to draw their focus.

More Information: You can read more about how and when to use Reveal to bring focus to user interface elements at the following link: `https://docs.microsoft.com/en-us/windows/uwp/design/style/reveal`

Creating a modern Windows app

We will start by creating a simple UWP app, with some common controls and modern features of Fluent Design like acrylic material.

Enabling developer mode

To create apps for UWP, you must enable developer mode in Windows 10:

1. Navigate to **Start** | **Settings** | **Update & Security** | **For developers**, and then click on **Developer mode**.
2. Accept the warning about how it "could expose your device and personal data to security risk or harm your device," and then close the **Settings** app.

Creating a UWP project

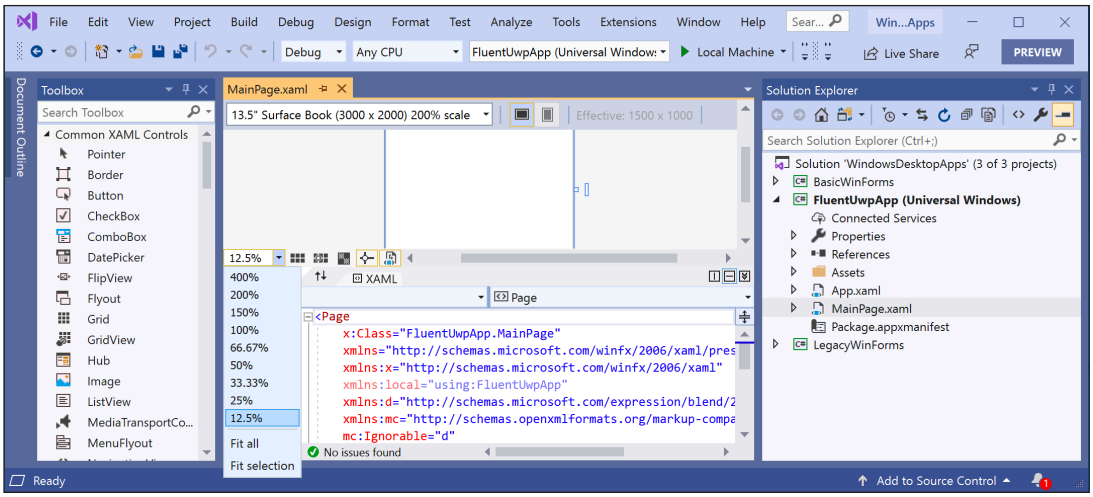
Now you will add a new UWP app project to your solution:

1. In Visual Studio 2019, open the **WindowsDesktopApps** solution.
2. Navigate to **File** | **Add** | **New Project....**
3. In the **Add a new project** dialog, enter `uwp` in the search box, select the **Blank App (Universal Windows)** template for C#, and then click **Next**. Make sure to select the one for C# and not Visual Basic!
4. For the **Project name**, enter `FluentUwpApp` and then click on **Create**.
5. In the **New Universal Windows Platform Project** dialog, choose the latest version of Windows 10 for **Target Version** and **Minimum Version** and click on **OK**.

Good Practice: Since the Fluent Design System was first released with Window 10 Fall Creators Update (10.0; Build 16299), you should be able to select that to support the features that we will cover in this appendix, but while you're learning, it's best to use the latest version to avoid unexpected incompatibilities. Developers writing UWP apps for a general audience should choose one of the latest builds of Windows 10 for Minimum Version. Developers writing enterprise apps should choose an older Minimum Version. Build 10240 was released in July 2015 and is a good choice for maximum compatibility, but you will not have access to modern features such as Fluent Design System.

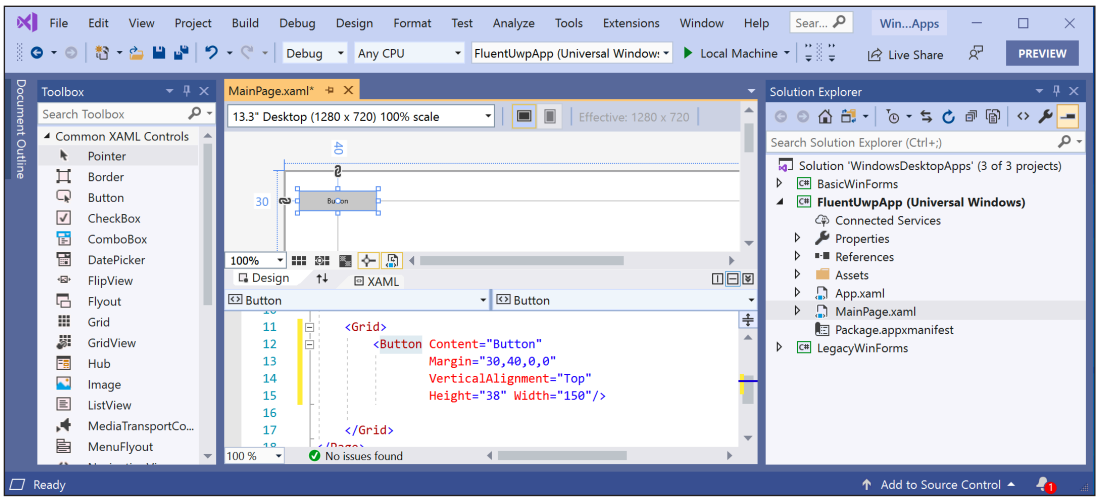
6. In **Solution Explorer**, double-click on the `MainPage.xaml` file to open it for editing. You will see the XAML design window showing a graphical view and a XAML view. You will be able to make the following observations:
    - The XAML designer is split horizontally, but you can toggle to a vertical split and collapse one side by clicking on the buttons on the right edge of the divider.
    - You can swap views by clicking on the double-arrow button in the divider.

- You can scroll and zoom in both views:



7. For **Design** view, change the device to **13.3" Desktop (1280 x 720)** **100**% scale and zoom to **100**%.
8. Navigate to **View** | **Toolbox** or press *Ctrl* + *W*, *X*. Note that the toolbox has sections for **Common XAML Controls, All XAML Controls**, and **General**.
9. At the top of the toolbox is a search box. Enter the letters `bu`, and then note that the list of controls is filtered.
10. Drag and drop the **Button** control from the toolbox onto the **Design** view.
11. Resize it by clicking, holding, and dragging any of the eight square-shape resize handles on each edge and in each corner.

    Note that the button is given a fixed width and height, and fixed left (30 units) and top (40 units) margins, to position and size it absolutely inside the grid, as shown in the following screenshot:



Although you can drag and drop controls, it is better to use the XAML view for layout so that you can position items relatively and implement more of a responsive design.

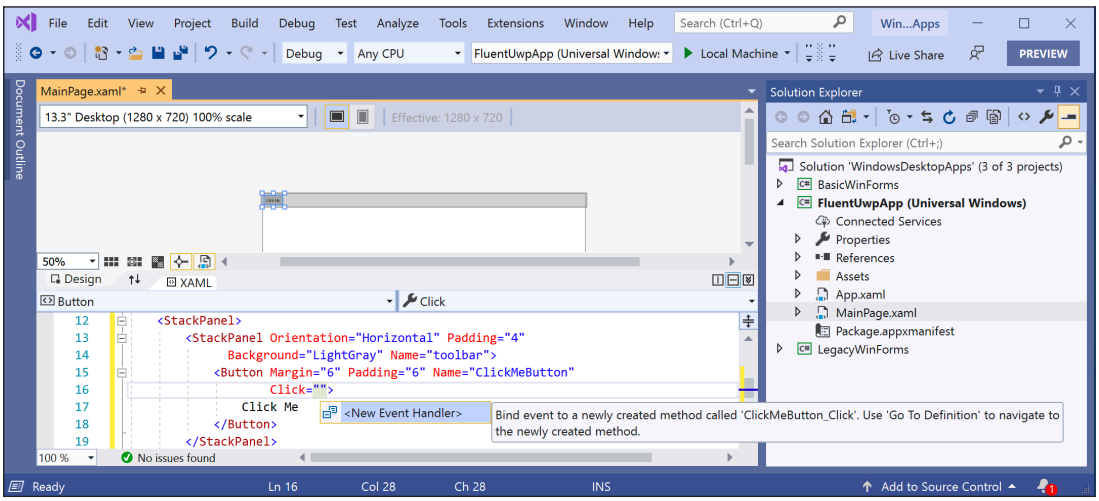
12. In the XAML view, find the `Button` element and delete it.
13. In the XAML view, inside the `Grid` element, enter the following markup:

```
<Button Margin="6" Padding="6" Name="ClickMeButton">
  Click Me
</Button>
```

14. Change the zoom to 50% and note that the button is automatically sized to its content, **Click Me**, aligned vertically in the center and aligned horizontally to the left, even if you toggle between vertical and horizontal phone layouts.
15. In the XAML view, delete the `Grid` element, and modify the XAML to wrap the `Button` element inside a horizontally orientated `StackPanel` with a light gray background that is inside a vertically orientated (by default) `StackPanel`, and note the change in its layout to be in the top left of the available space, as shown in the following code:

```
<StackPanel>
  <StackPanel Orientation="Horizontal" Padding="4"
              Background="LightGray" Name="toolbar">
    <Button Margin="6" Padding="6" Name="ClickMeButton">
      Click Me
    </Button>
  </StackPanel>
</StackPanel>
```

16. Modify the `Button` element to give it a new event handler for its `Click` event, as shown in the following screenshot:

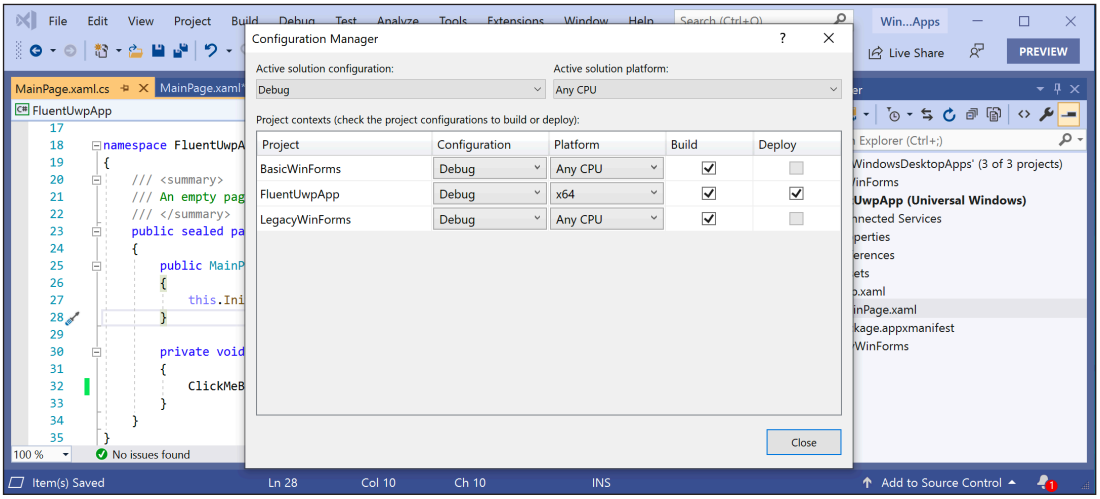


17. Right-click the event handler name and select **Go To Definition** or press *F12*.

18. Add a statement to the event handler method that sets the content of the button to the current time, as shown in the following code:

```
private void ClickMeButton_Click(object sender, RoutedEventArgs e)
{
  ClickMeButton.Content = DateTime.Now.ToString("hh:mm:ss");
}
```

19. Navigate to **Build | Configuration Manager...** for the **FluentUwpApp** project, select the **Build** and **Deploy** checkboxes, select a **Platform** of **x64**, and then select **Close**, as shown in the following screenshot:



20. Run the application by navigating to **Debug | Start Without Debugging** or pressing *Ctrl + F5*.

21. Click on the **Click Me** button and note that every time you click on the button, the button's content changes to show the current time.

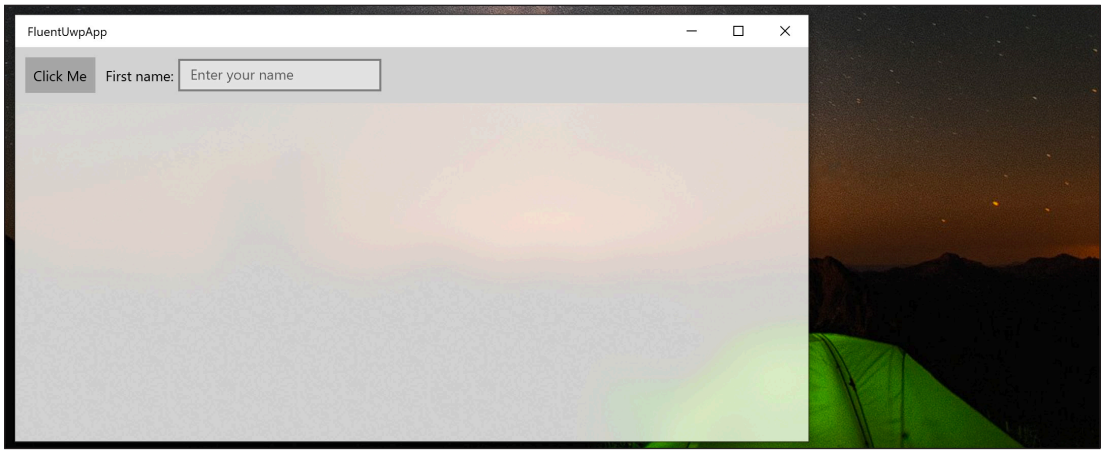
Exploring common controls and acrylic brushes

Now you will explore some common controls and painting with acrylic brushes:

1. Open `MainPage.xaml`, set the stack panel's background to use the acrylic system window brush, and add some elements to the stack panel after the button for the user to enter their name, as shown highlighted in the following markup:

```
<StackPanel
  Background="{ThemeResource SystemControlAcrylicWindowBrush}">
  <StackPanel Orientation="Horizontal" Padding="4"
              Background="LightGray" Name="toolbar">
    <Button Margin="6" Padding="6" Name="ClickMeButton"
            Click="ClickMeButton_Click">
      Click Me
    </Button>
    <TextBlock Text="First name:"
      VerticalAlignment="Center" Margin="4" />
    <TextBox PlaceholderText="Enter your name"
      VerticalAlignment="Center" Width="200" />
  </StackPanel>
</StackPanel>
```

2. Run the application by navigating to **Debug | Start Without Debugging**, and note the tinted acrylic material showing the green tent and orange sunset over the mountains of one of the standard Windows 10 wallpapers through the app window background, as shown in the following screenshot:



Acrylic uses a lot of system resources, so if an app loses focus, or your device is low on battery, then acrylic is disabled automatically.

Exploring Reveal

Reveal is built-in for some controls, such as `ListView` and `NavigationView`, that you will see later. For other controls, you can enable it by applying a theme style. First, we will add some XAML to define a calculator user interface made up of a grid of buttons. Then, we will add an event handler for the page's `Loaded` event so that we can apply the Reveal theme style and other properties by enumerating the buttons instead of having to manually set attributes for each one in XAML:

1. Open `MainPage.xaml`, add a new horizontal stack panel under the one used as a toolbar, and add a grid with buttons to define a calculator, as shown highlighted in the following markup:

```
<StackPanel
  Background="{ThemeResource SystemControlAcrylicWindowBrush}">
  <StackPanel Orientation="Horizontal" Padding="4"
              Background="LightGray" Name="toolbar">
    <Button Margin="6" Padding="6" Name="ClickMeButton"
            Click="ClickMeButton_Click">
      Click Me
    </Button>
    <TextBlock Text="First name:"
               VerticalAlignment="Center" Margin="4" />
    <TextBox PlaceholderText="Enter your name"
             VerticalAlignment="Center" Width="200" />
  </StackPanel>
  <StackPanel Orientation="Horizontal">
    <Grid Background="DarkGray" Margin="10"
          Padding="5" Name="gridCalculator">
      <Grid.ColumnDefinitions>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
        <ColumnDefinition/>
      </Grid.ColumnDefinitions>
      <Grid.RowDefinitions>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
        <RowDefinition/>
      </Grid.RowDefinitions>
      <Button Grid.Row="0" Grid.Column="0" Content="X" />
      <Button Grid.Row="0" Grid.Column="1" Content="/" />
      <Button Grid.Row="0" Grid.Column="2" Content="+" />
      <Button Grid.Row="0" Grid.Column="3" Content="-" />
      <Button Grid.Row="1" Grid.Column="0" Content="7" />
      <Button Grid.Row="1" Grid.Column="1" Content="8" />
      <Button Grid.Row="1" Grid.Column="2" Content="9" />
      <Button Grid.Row="1" Grid.Column="3" Content="0" />
      <Button Grid.Row="2" Grid.Column="0" Content="4" />
```

```
      <Button Grid.Row="2" Grid.Column="1" Content="5" />
      <Button Grid.Row="2" Grid.Column="2" Content="6" />
      <Button Grid.Row="2" Grid.Column="3" Content="." />
      <Button Grid.Row="3" Grid.Column="0" Content="1" />
      <Button Grid.Row="3" Grid.Column="1" Content="2" />
      <Button Grid.Row="3" Grid.Column="2" Content="3" />
      <Button Grid.Row="3" Grid.Column="3" Content="=" />
    </Grid>
  </StackPanel>
</StackPanel>
```

2. In the `Page` element, add a new event handler for `Loaded`, as shown highlighted in the following markup:

```
<Page
...
  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
  Loaded="Page_Loaded">
```

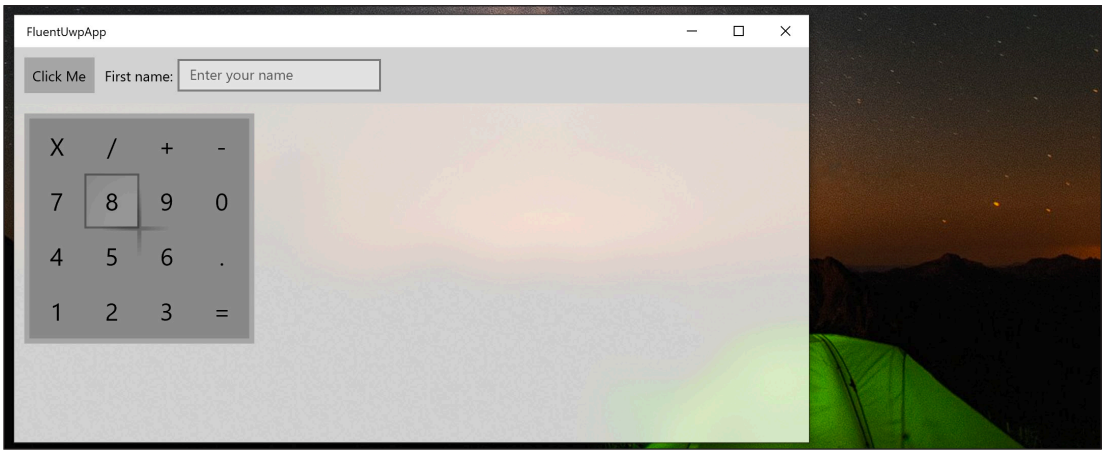
3. Right-click `Page_Loaded` and select **Go To Definition** or press *F12*.
4. Add statements to the `Page_Loaded` method to loop through all of the calculator buttons, setting them to be the same size, and apply the Reveal style, as shown in the following code:

```
private void Page_Loaded(object sender, RoutedEventArgs e)
{
  Style reveal = Resources.ThemeDictionaries[
    "ButtonRevealStyle"] as Style;

  foreach (Button b in gridCalculator.Children.OfType<Button>())
  {
    b.FontSize = 24;
    b.Width = 54;
    b.Height = 54;
    b.Style = reveal;
  }
}
```

5. Run the application by navigating to **Debug | Start Without Debugging** and note the calculator buttons start with a flat gray user interface.

6. When the user moves their mouse pointer over the bottom-right corner of the `8` button, we see that Reveal lights it up, and parts of the surrounding buttons light up too, as shown in the following screenshot:



7. Close the app.

The implementation of the Reveal effect has subtlety changed over the recent biannual updates to Windows 10, so the effect in your app may look slightly different.

Installing more controls

In addition to dozens of built-in controls, you can install additional ones as NuGet packages and one of the best is the **UWP Community Toolkit**.

More Information: You can read more about the UWP Community Toolkit at the following link: `https://docs.microsoft.com/en-us/windows/communitytoolkit/`

One of the toolkit controls is an editor for Markdown.

More Information: You can read about the Markdown project at the following link: `https://daringfireball.net/projects/markdown/`

Let's install the **UWP Community Toolkit** now and explore some of its controls:

1. In the **FluentUwpApp** project, right-click on **References**, and select **Manage NuGet Packages....**
2. Click **Browse**, search for `Microsoft.Toolkit.Uwp.UI.Controls`, and click on **Install**.

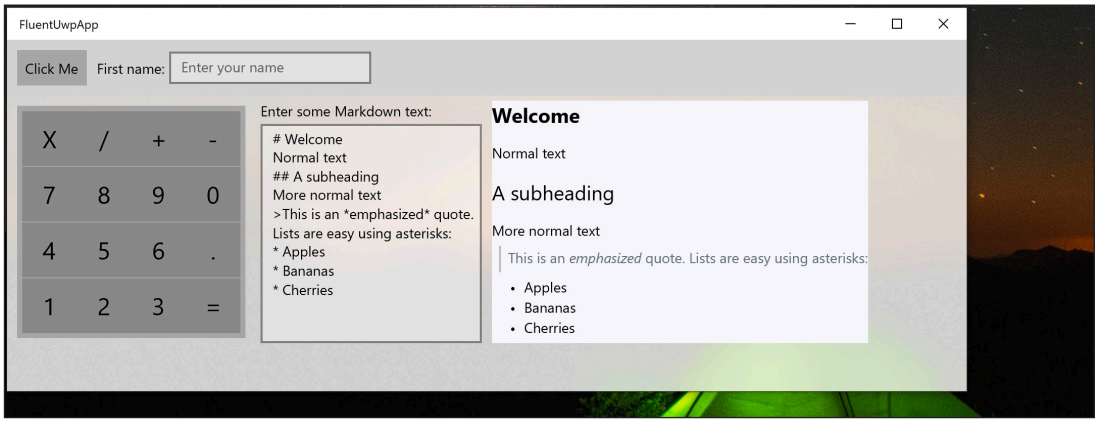
3. Review the changes and accept the license agreement.
4. Navigate to **Build | Build FluentUwpApp**. This will restore packages if necessary.
5. Open `MainPage.xaml`, and in the `Page` element, import the toolkit namespace as a prefix named `kit`, as shown highlighted in the following markup:

```
<Page
...
  xmlns:kit="using:Microsoft.Toolkit.Uwp.UI.Controls"
  mc:Ignorable="d"
  Background="{ThemeResource ApplicationPageBackgroundThemeBrush}"
  Loaded="Page_Loaded">
```

6. Inside the second horizontal stack panel and after the calculator grid, add a textbox and a markdown text block that are data bound together so that the text in the textbox becomes the source of the markdown control, as shown highlighted in the following markup:

```
<StackPanel Orientation="Horizontal">
  <Grid Background="DarkGray" Margin="10"
        Padding="5" Name="gridCalculator">
  ...
  </Grid>
  <TextBox Name="markdownSource" Text="# Welcome"
           Header="Enter some Markdown text:"
           VerticalAlignment="Stretch" Margin="5"
           AcceptsReturn="True" />
  <kit:MarkdownTextBlock Margin="5"
    Text="{Binding ElementName=markdownSource, Path=Text}"
    VerticalAlignment="Stretch" HorizontalAlignment="Stretch" />
</StackPanel>
```

7. Run the application by navigating to **Debug | Start Without Debugging** and note that the user can enter Markdown syntax in the textbox, and it is rendered in the Markdown text block, as shown in the following screenshot:



Using resources and templates

When building graphical user interfaces, you will often want to use a resource, such as a brush to paint the background of controls or an instance of a class to perform custom conversions. These resources can be defined in a single place and shared throughout the app.

Sharing resources

A good place to define shared resources is at the app level, so let's see how to do that:

1. In **Solution Explorer**, open the `App.xaml` file.
2. Add the following markup inside the existing `Application` element to define a linear gradient brush with a key of rainbow, as shown highlighted in the following markup:

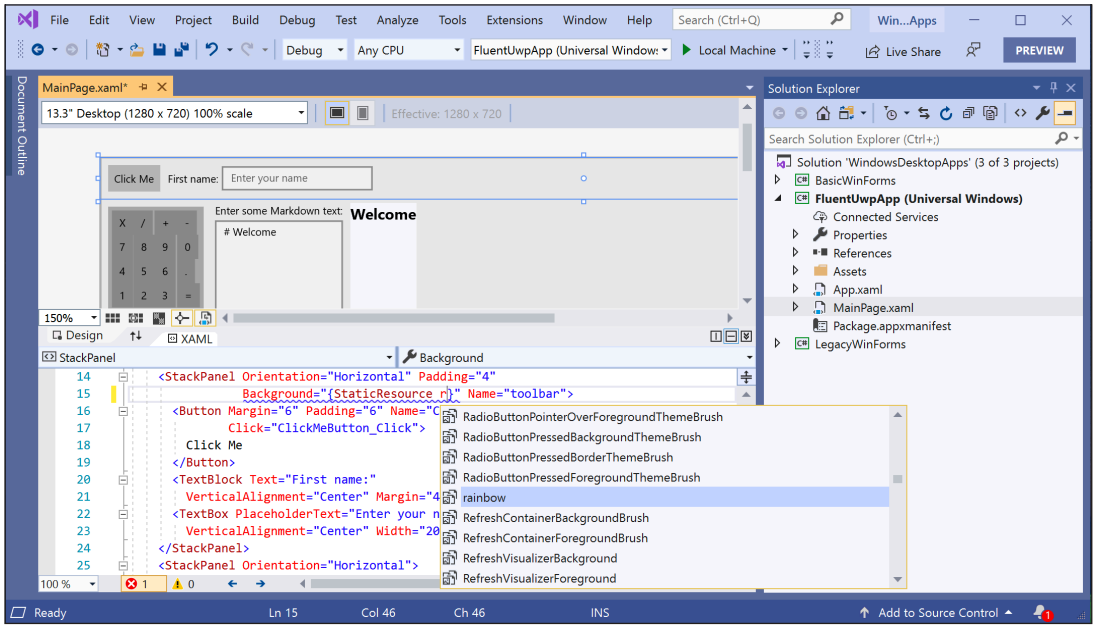
```
<Application
  x:Class="FluentUwpApp.App"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:local="using:FluentUwpApp">

  <Application.Resources>
    <LinearGradientBrush x:Key="rainbow">
      <GradientStop Color="Red" Offset="0" />
      <GradientStop Color="Orange" Offset="0.1" />
      <GradientStop Color="Yellow" Offset="0.3" />
      <GradientStop Color="Green" Offset="0.5" />
      <GradientStop Color="Blue" Offset="0.7" />
      <GradientStop Color="Indigo" Offset="0.9" />
      <GradientStop Color="Violet" Offset="1" />
    </LinearGradientBrush>
  </Application.Resources>
</Application>
```

3. Navigate to **Build | Build FluentUwpApp**.
4. In `MainPage.xaml`, modify the `StackPanel` element named `toolbar` to change its background from `LightGray` to the static resource `rainbow` brush, as shown highlighted in the following markup:

```
<StackPanel Orientation="Horizontal" Padding="4"
  Background="{StaticResource rainbow}" Name="toolbar">
```

5. As you enter the reference to a static resource, IntelliSense will show your `rainbow` resource and the built-in resources, as shown in the following screenshot:



Good Practice: A resource can be an instance of any object. To share it within an application, define it in the `App.xaml` file and give it a unique key. To set an element's property with a resource, use `{StaticResource key}`.

Resources can be defined and stored inside any element of XAML, not just at the app level. For example, if a resource is only needed on `MainPage`, then it can be defined there. You can also dynamically load XAML files at runtime.

More Information: You can read more about the Resource Management System at the following link: `https://docs.microsoft.com/en-us/windows/uwp/app-resources/`

Replacing a control template

You can redefine how a control looks by replacing its default template. The default control template for a button is flat and transparent.

One of the most common resources is a style that can set multiple properties at once. If a style has a unique key, then it must be explicitly set, as we did earlier with the linear gradient. If it doesn't have a key, then it will be automatically applied based on the `TargetType` property:

1. In `App.xaml`, define a control template inside the `Application.Resources` element and note that the `Style` element will automatically set the `Template` property of all controls that are `TargetType`, that is, buttons, to use the defined control template, as shown highlighted in the following markup:

```
<Application.Resources>
  <LinearGradientBrush x:Key="rainbow">
  ...
  </LinearGradientBrush>
  <ControlTemplate x:Key="DarkGlassButton" TargetType="Button">
    <Border BorderBrush="#FFFFFFFF"
            BorderThickness="1,1,1,1" CornerRadius="4,4,4,4">
      <Border x:Name="border" Background="#7F000000"
              BorderBrush="#FF000000"
              BorderThickness="1,1,1,1"
              CornerRadius="4,4,4,4">
        <Grid>
          <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="*"/>
          </Grid.RowDefinitions>
          <Border Opacity="0"
                  HorizontalAlignment="Stretch" x:Name="glow"
                  Width="Auto" Grid.RowSpan="2"
                  CornerRadius="4,4,4,4">
          </Border>
          <ContentPresenter HorizontalAlignment="Center"
                            VerticalAlignment="Center"
                            Width="Auto"
                            Grid.RowSpan="2" Padding="4"/>
          <Border HorizontalAlignment="Stretch" Margin="0,0,0,0"
                  x:Name="shine" Width="Auto"
                  CornerRadius="4,4,0,0">
            <Border.Background>
              <LinearGradientBrush EndPoint="0.5,0.9"
                                   StartPoint="0.5,0.03">
                <GradientStop Color="#99FFFFFF" Offset="0"/>
                <GradientStop Color="#33FFFFFF" Offset="1"/>
              </LinearGradientBrush>
            </Border.Background>
          </Border>
        </Grid>
      </Border>
    </Border>
  </ControlTemplate>
  <Style TargetType="Button">
```

```
    <Setter Property="Template"
            Value="{StaticResource DarkGlassButton}" />
    <Setter Property="Foreground" Value="White" />
  </Style>
</Application.Resources>
```

2. Run the application and note the black glass effect on the button in the toolbar.

The calculator buttons are not affected at runtime by this black glass effect because we replace their styles using code after the page has loaded.

Using data binding

When building graphical user interfaces, you will often want to bind a property of one control to another, or to some data.

Binding to elements

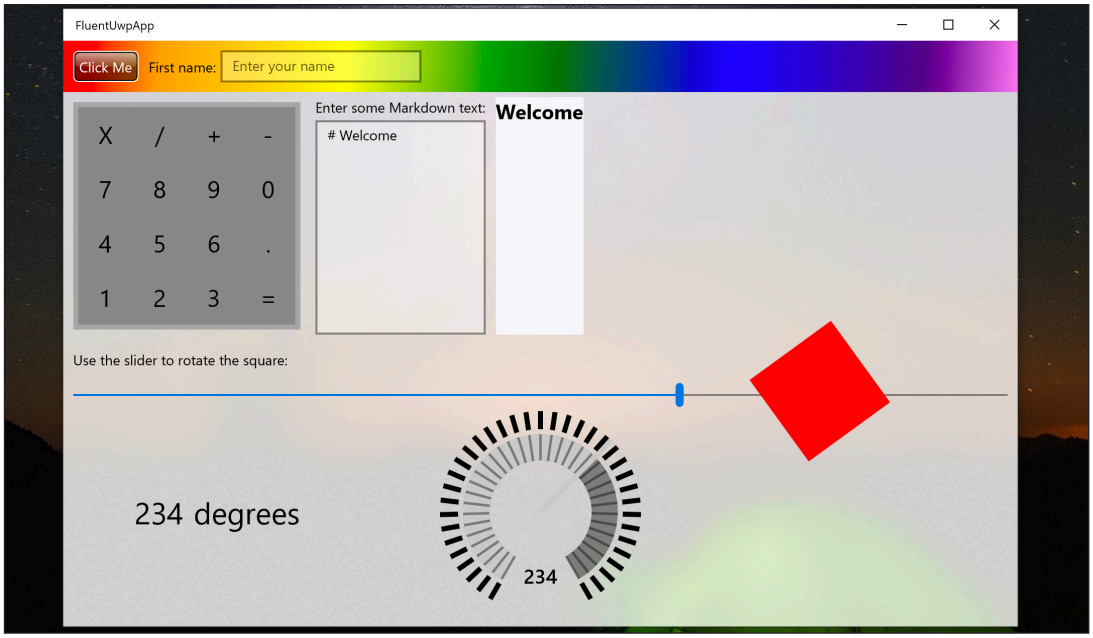
The simplest type of binding is between elements:

1. In `MainPage.xaml`, after the second horizontal stack panel and inside the outer vertical stack panel, add a text block for instructions, a slider for selecting a rotation, a grid containing a stack panel and text blocks to show the selected rotation in degrees, a radial gauge from the UWP Community Toolkit, and a red square to rotate, as shown in the following markup:

```
<TextBlock Margin="10">
  Use the slider to rotate the square:
</TextBlock>
<Slider Value="180" Minimum="0" Maximum="360"
        Name="sliderRotation" Margin="10,0" />
<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition />
    <ColumnDefinition />
    <ColumnDefinition />
  </Grid.ColumnDefinitions>
  <StackPanel Orientation="Horizontal"
              VerticalAlignment="Center"
              HorizontalAlignment="Center">
    <TextBlock FontSize="30"
      Text="{Binding ElementName=sliderRotation, Path=Value}" />
    <TextBlock Text="degrees" FontSize="30" Margin="10,0" />
  </StackPanel>
  <kit:RadialGauge Grid.Column="1" Minimum="0" Maximum="360"
    Value="{Binding ElementName=sliderRotation, Path=Value}"
```

```
      Height="200" Width="200" />
    <Rectangle Grid.Column="2" Height="100" Width="100" Fill="Red">
      <Rectangle.RenderTransform>
        <RotateTransform
          Angle="{Binding ElementName=sliderRotation, Path=Value}" />
      </Rectangle.RenderTransform>
    </Rectangle>
  </Grid>
```

2. Note that the text of the text block, the value of the radial gauge, and the angle of the rotation transform are all bound to the slider's value using the `{Binding}` markup extension that's specific to the name of the element and the name, also known as the path, of a property to bind to.
3. Run the app and then click, hold, and drag the slider to rotate the red square, as shown in the following screenshot:



Practicing and exploring

Test your knowledge and understanding by answering some questions, get some hands-on practice, and explore this appendix's topics with deeper research.

Exercise: B.1 – Test your knowledge

Answer the following questions:

1. .NET 5 is cross-platform. Windows Forms and WPF apps can run on .NET 5. Can those apps therefore run on macOS and Linux?
2. How does a Windows Forms app define its user interface and why is this a potential problem?
3. How can a WPF or UWP app define its user interface and why is this good for developers?
4. List five layout containers and how their child elements are positioned within them.
5. What is the difference between `Margin` and `Padding` for an element like a `Button`?
6. How are event handlers attached to an object using XAML?
7. What do XAML styles do?
8. Where can you define resources?
9. How can you bind the property of one element to a property on another element?
10. Why might you implement `IValueConverter`?

Exercise: B.2 – Explore topics

Use the following links to read more about this appendix's topics:

- **Enable your device for development**: `https://docs.microsoft.com/en-us/windows/uwp/get-started/enable-your-device-for-development`
- **Get started with Windows 10 apps**: `https://docs.microsoft.com/en-us/windows/uwp/get-started/`
- **Getting Started with the Windows Community Toolkit**: `https://docs.microsoft.com/en-us/windows/communitytoolkit/getting-started`
- **Design and code Windows apps**: `https://docs.microsoft.com/en-us/windows/uwp/design/`
- **Develop UWP apps**: `https://docs.microsoft.com/en-us/windows/uwp/develop/`

Summary

In this appendix, you learned that .NET 5 supports older technologies for building Windows desktop applications, including Windows Forms and WPF.

You learned how to build a graphical user interface using XAML and the new Fluent Design System, including features such as acrylic material and Reveal lighting.

You also learned how to share resources in an app, how to replace a control's template, and how to bind to data and controls.