XML Documentation Comments in C#

XML documentation comments are a special kind of comment, added above the definition of any user-defined type or member. They are special because they can be processed by the compiler to generate an XML documentation file at compile time. The compiler-generated XML file can be distributed alongside your .NET assembly so that Visual Studio and other IDEs can use IntelliSense to show quick information about types or members. Additionally, the XML file can be run through tools like DocFX and Sandcastle to generate API reference websites.

XML documentation comments, like all other comments, are ignored by the compiler.

You can generate the XML file at compile time by doing one of the following:

- If you are developing an application with .NET Core from the command line, you can add a **GenerateDocumentationFile** element to the **<PropertyGroup>** section of your **.csproj project file**. You can also specify the path to the documentation file directly using **DocumentationFile** element. The following example generates an XML file in the project directory with the same root filename as the assembly:

  ```
  <GenerateDocumentationFile>true</GenerateDocumentationFile>
  ```

  This is equivalent to the following:

  ```
  <DocumentationFile>bin\$(Configuration)\$(TargetFramework)\$(AssemblyName)
                    .xml </DocumentationFile>
  ```

- If you are developing an application using Visual Studio**, right-click on the project** and **select Properties**. In the properties dialog**, select the Build tab**, and **check XML documentation file**. You can also change the location to which the compiler writes the file.
- If you are compiling a .NET application from the command line, add the **-doc compiler option** when compiling.

XML Documentation Tags

1. **<summary>**
   The <summary> tag adds brief information about a type or member.

2. **<remarks>**
   The <remarks> tag supplements the information about types or members that the <summary> tag provides.
3. **<returns>**
   The <returns> tag describes the return value of a method declaration.
4. **<value>**
   The <value> tag is similar to the <returns> tag, except that you use it for properties.
5. **<example>**
   You use the <example> tag to include an example in your XML documentation. This involves using the child <code> tag.

6. **<para>**
   You use the <para> tag to format the content within its parent tag. <para> is usually used inside a tag, such as <remarks> or <returns>, to divide text into paragraphs. You can format the contents of the <remarks> tag for your class definition.
7. **<c>**
   Still on the topic of formatting, you use the <c> tag for marking part of text as code. It's like the <code> tag but inline. It's useful when you want to show a quick code example as part of a tag's content.
8. **<exception>**
   By using the <exception> tag, you let your developers know that a method can throw specific exceptions. The cref attribute represents a reference to an exception that is available from the current compilation environment. This can be any type defined in the project or a referenced assembly. The compiler will issue a warning if its value cannot be resolved.
9. **<see>**
   The <see> tag lets you create a clickable link to a documentation page for another code element. The cref is a required attribute that represents a reference to a type or its member that is available from the current compilation environment. This can be any type defined in the project or a referenced assembly.
10. **<seealso>**
    You use the <seealso> tag in the same way you do the <see> tag. The only difference is that its content is typically placed in a "See Also" section.
11. **<param>**
    You use the <param> tag to describe a method's parameters. The parameter the tag describes is specified in the required name attribute.
12. **<typeparam>**
    You use <typeparam> tag just like the <param> tag but for generic type or method declarations to describe a generic parameter.
13. **<paramref>**
    Sometimes you might be in the middle of describing what a method does in what could be a <summary> tag, and you might want to make a reference to a parameter. The <paramref> tag is great for just this. Like the <param> tag, the parameter name is specified in the required name attribute.
14. **<typeparamref>**
    You use <typeparamref> tag just like the <paramref> tag but for generic type or method declarations to describe a generic parameter.
15. **<list>**
    You use the <list> tag to format documentation information as an ordered list, unordered list, or table. You can make an ordered list or table by changing the type attribute to number or table, respectively.
16. **<inheritdoc>**
    You can use the <inheritdoc> tag to inherit XML comments from base classes, interfaces, and similar methods. This eliminates unwanted copying and pasting of duplicate XML comments and automatically keeps XML comments synchronized.
17. **<include>**

The <include> tag lets you refer to comments in a separate XML file that describe the types and members in your source code, as opposed to placing documentation comments directly in your source code file.

Example

C# Code

```
/*
    The main Math class
    Contains all methods for performing basic math functions
*/
/// <summary>
/// The main <c>Math</c> class.
/// Contains all methods for performing basic math functions.
/// <list type="bullet">
/// <item>
/// <term>Add</term>
/// <description>Addition Operation</description>
/// </item>
/// <item>
/// <term>Subtract</term>
/// <description>Subtraction Operation</description>
/// </item>
/// <item>
/// <term>Multiply</term>
/// <description>Multiplication Operation</description>
/// </item>
/// <item>
/// <term>Divide</term>
/// <description>Division Operation</description>
/// </item>
/// </list>
/// </summary>
/// <remarks>
/// <para>This class can add, subtract, multiply and divide.</para>
/// <para>These operations can be performed on both integers and doubles.</para>
/// </remarks>
public class Math
{
// Divides a double by another and returns the result
    /// <summary>
    /// Divides a double <paramref name="a"/> by another double <paramref
name="b"/> and returns the result.
    /// </summary>
    /// <returns>
    /// The quotient of two doubles.
    /// </returns>
    /// <example>
    /// <code>
    /// double c = Math.Divide(4.5, 5.4);
    /// if (c > 1.0)
```

```
    /// {
    ///     Console.WriteLine(c);
    /// }
    /// </code>
    /// </example>
    /// <exception cref="System.DivideByZeroException">Thrown when <paramref
name="b"/> is equal to 0.</exception>
    /// See <see cref="Math.Divide(int, int)"/> to divide integers.
    /// <seealso cref="Math.Add(double, double)"/>
    /// <seealso cref="Math.Subtract(double, double)"/>
    /// <seealso cref="Math.Multiply(double, double)"/>
    /// <param name="a">A double precision dividend.</param>
    /// <param name="b">A double precision divisor.</param>
    public static double Divide(double a, double b)
    {
        return a / b;
    }
}
```