**Coding Documentation Standard**

| **Tag** | **Usage** | **Description** |
| --- | --- | --- |
| summary | <summary>Your summary</summary> | Provide a short description of the type or member.  *The text typed between the opening and closing summary tags is displayed in Visual Studio's IntelliSense when you hover over the type.* |
| param | <param name='name'>Description.</param>  'name' must be surrounded by single quotes. | Provide a description of the related parameter.  *The text typed between the opening and closing param tags is displayed when writing the client code.* |
| returns | <returns>Description.</returns> | Provide a description of what the method returns to the client. |
| remarks | <remarks>Your remarks.</remarks> | Provide more detailed information about the type or member that may be important to someone using the type.  *This is a good place to document things like whether the type is thread safe or not.* |
| para | <para>Your text.</para> | Allows you to control the format of the documentation output by inserting or wrapping segments within paragraphs.  *You will nest para tags within other tags.* |
| c | <c>Your code sample.</c> | Indicates a short segment of code.  *You will use the c tag to draw attention to particular words, phrases or code in the resulting documentation.  You will nest c tags within other tags.* |
| paramref | <paramref name="name"/>  Include the closing tag, and you must use double quotes around the "name" of the parameter. | Allows you to link the word to the parameter with the same name.  The paramref tag causes the contained word to be linked to the param description in the resulting documentation.  <param name='myParm'>Desc</param>  <remarks>Determines if <paramref name=”myParm”/> is either <c>null</c> or <c>empty</c>.</remarks>  *You will find this will be one of the most often used tags.* |
| see | <see cref="member"/>  Include the closing tag, and you must use double quotes around the linked "member". | Allows you to create a reference to any type, member, or field that can be used within the class without causing a compilation error (this includes .NET types and your custom types as well).  *One really neat integration feature with this tag is that the references integrate directly into the Visual Studio help system.*  <returns>A <see cref=”System.String”/> containing the user's unique identifier.</returns>  *When you click the link on String, you will see the actual Microsoft documentation for the String type.* |
| exception | <exception cref="member">  Include the closing tag, and you must use double quotes around the linked "member". | Allows you to document the type of exceptions your member will throw.  <exception cref="ArgumentException"> If <paramref name="firstName"/> is <c>empty</c>.</exception>  *When you click on the member link, you will see the actual Microsoft documentation for the exception type.  It is not uncommon for your methods to use more than one <exception> tag.* |
| code | <code>Your code sample.</code> | The code tag is similar to the <c> tag, but it is intended to illustrate a true code usage example.  The code tag is generally enclosed within an <example> tag set (see below).  *Use the code tag in concert with the example tag.* |
| example | <example>Your example.</example> | Provides a means to illustrate code examples.  *Generally an example tag set will encompass a code tag set.  Use the example tag in concert with the code tag.* |

**Documenting your Code**

Documenting source code should be a standard part of the development process. If you get in the habit of documenting your source as you write it, you will find that you can produce fully documented code much faster than if you write code and try and go back and instrument it with documentation at a later time.

| **Type / Member** | **Tags** |
| --- | --- |
| Classes | Summary, Remarks and Example (if necessary) |
| Methods | Summary, Remarks, Param, Returns and Exception |
| Properties | Summary, Value, Remarks and Exception  *In the summary tag, indicate whether the property is read/write, read-only or write-only.  <summary>Gets or sets the first name of the user.<summary>  If you add this information to the summary tag, the developer will be able to see this information in IntelliSense when they are consuming the library ... a big help.* |
| Fields | Summary |
| Constants | Summary and Value  *Although there is a specific Value tag, indicate in the Summary tag the value of the constant, and you will be able to see the assigned value in IntelliSense during development. I have found that this removes the need for me to go and find the constant to determine what value is assigned because I can simply hover over the constant, and IntelliSense will do the work for me (you gotta' try this one).*  <summary>Stored procedure to access all states. Value: MyStoredProcName <summary> |
| Delegates | Summary, Remarks, Param and Returns |

**Sample**

namespace Mike.Elliott.Articles.XML.Documentation

{

using System;

/// <summary>

/// The <c>DocumentationSample</c> type

/// demonstrates code comments.

/// </summary>

/// <remarks>

/// <para>

/// The <c>DocumentationSample</c> type

/// provides no real functionality;

/// however, it does provide examples of

/// using the most common, built in

/// <c>C#</c> xml documentation tags.

/// </para>

/// <para>

/// <c>DocumentationSample</c> types are not

/// safe for concurrent access by

/// multiple threads.

/// </para>

/// </remarks>

public class DocumentationSample

{

/// <summary>

/// Initializes a new instance of a

/// <c>DocumentationSample</c> type.

/// </summary>

/// <example>The following is an example of initializing a

/// <c>DocumentationSample</c> type:

/// <code>

/// // Create the type.

/// DocumentationSample ds = new DocumentationSample();

///

/// if ( null == ds )

/// return;

///

/// return ds.MyMethod( “someString” );

/// </code>

/// </example>

public DocumentationSample() { … }

/// <**summary**>Causes something happen.</**summary**>

/// <**param** name="someValue">

/// A <see cref="String"/> type representing a value.

/// </**param**>

/// <**exception** cref="ArgumentNullException">

/// if <**paramref** name="someValue"/> is <c>null</c>.

/// </**exception**>

/// <**exception** cref="ArgumentException">

/// if <**paramref** name="someValue"/> is <c>empty</c>.

/// </**exception**>

/// <**returns**><**paramref** name="someValue"/>

/// as passed in.</**returns**>

public string MyMethod( string someValue )

{

if ( null == someValue )

throw new ArgumentNullException( "Your message." );

if ( 0 >= someValue.Length )

throw new ArgumentException( "Your message." );

return someValue;

}

}

