XML Documentation Comments Guide

XML Documentation Comments defined by
Microsoft Visual Studio .NET (C# and Visual Basic compilers), Sandcastle and NDoc

Compiled by Wm. Eric Brunsen
Dynicity, LLC
10/3/2007

Table of Contents

Revision History

Date	Version	Author	Changes
August 11, 2007	1.0.0	Eric Brunsen	Initial Release
August 15, 2007	1.1.0	Eric Brunsen	**Changed:** Moved the <see> & <seealso> notes from Document Conventions section to <see> & <seealso> Tags section **Added:** Links to "*Sandcastle Wiki*" and "*Sandcastle on CodePlex*" **Changed:** Added Visual Studio Version to the Tags Listing table **Changed:** Updated URL for download location of this document
October 3, 2007	1.2.0	Eric Brunsen	**Fixed:** Link to <include> tag in help files now correctly links to the <include> tag rather than the <list> tag. **Fixed:** <exception> tag "example" is now correct (it was using the same "example" as the <event> tag. **Fixed:** Link to "Sandcastle Wiki" in the Resources Links section now has the correct URL **Fixed:** The "CDATA" format in the XML Documentation Comments Notes section (#3) is now correct. **Fixed:** Various spelling and grammar errors throughout the document **Added:** Added links to MSDN documentation for XML Documentation Comments for .NET 2.0 (C# and VB.NET) added to the Sources section in the Help files. **Added:** Added note to the XML Documentation Comments Notes section regarding the compiler output when using the /doc compiler switch. **Added:** Added note regarding the Sandcastle API Filter to the <exclude> tag **Added:** Link to dnrTV Sandcastle episode

Document Notes

1. This document is a work in progress, meaning that, even though I have tried my best to verify the validity of the tags and their related information, it is not necessarily complete or correct.

2. I have used as many sources and documents as I was able to find on the web, as well as entries from the "*Microsoft Developer Documentation and Help System*" forum and comments and feedback from individuals.

   For information on the sources that I used, please see the "Resources" section of this document.
   For information on the people who contributed to this document please see the "People" section of this document.

3. I will try to keep this document, the HTML Help file, and the website updated as the XML Documentation Comments tags that are defined by Microsoft and Sandcastle are changed or added.
   The NDoc tags should not change since at this time work on NDoc has been officially stopped.

4. If you find errors or omissions, or have ideas that would make the document more useful please send me an email (the address is in the "Document Links" section of this document).

5. All of the code examples are in C# at this time. If someone wants to add samples in Visual Basic (or any other .NET language) please feel free to send them to me, and I'll get them added.

6. For a list of references on XML Documentation Comments from Microsoft see the "Links" section of this document

7. For a list of Sandcastle related resources, see the "Resources" section of this document

8. If you have questions on the use of a particular tag, the best place for getting answers to your questions is on the "*Microsoft Developer Documentation and Help System*" forum (see the "Links" section of this document for the URL).

XML Documentation Comments Notes

1. All XML/HTML within an XML Documentation Comment must be “well formed.”

2. If you want angle brackets to appear in the text of a documentation comment, use **<** and **>**.
   For example, the string "<text in angle brackets>" will appear as <text in angle brackets>.

3. Another method for inserting angle brackets and other characters that are not valid XML characters is to put them within a “<![CDATA[ ]]>“ XML tag.
   For example: <![CDATA[<text in angle brackets>]]>

4. To reference a generic type as a target in a tag’s cref attribute you can use curly braces around the “T”.
   For example <see cref=“CustomCollection{T}”/>. For a more in depth example see the “Generic Type cref example” section of this document.

5. XML Documentation Comments support all of the common HTML tags such a <p>, <b>, <a>, <img>, etc., so you can use HTML markup to add additional formatting to your XML comments if needed.

6. To enable XML Documentation within Visual Studio 2005, see the entry “How to: Generate XML Documentation for a Project“ in the “Sources and Links” section of this document.

7. XML Documentation Comments cannot be applied to a namespace within the Visual Studio environment, however they can be applied using NDoc or Sandcastle Help File Builder as follows:
   a. NDoc and the Sandcastle Help File Builder do support adding namespace comments using the GUI.
   b. You can also create a custom XML comments file that can be included by the Sandcastle Help File Builder that specifies the project and namespace comments.
   c. NDoc also supports a class with the name **NamespaceDoc** in each namespace. The summary from that class will then be used as the namespace summary. The class itself will not show up in the resulting documentation output.

8. An example of the output of the XML Documentation Comments after compiling using the /doc argument is available in the "<include>" tag example.

Document Conventions

This section of the document describes the format used in this document.

The format of the "Tags Descriptions" portion of the document is as follows:

Tag:	The name of the tag the description applies to
Type:	Whether the tag is a Top-Level or Inline tag
Applies To:	What code elements the tag is used to document
Defined By:	The program in which the tag is defined
Syntax:	The syntax for using the tag
Parameters:	The parameters for the tag and a description of each of the parameters
Remarks:	Remarks about the tag and its usage
Example:	Example of how the tag can be used

Tag Types:

I could not find a definitive definition of the naming for tag types, so for this document, I have defined the tag types as follows:

Top-Level:

This is a tag that appears at the root level of an XML Documentation Comment block (actually it is an element of the <doc> element, but the <doc> element is not shown within the Microsoft Visual Studio .NET environment).

An example of a Top-Level tag is the <summary> tag.

Inline:

This type of XML Documentation Comment is also referred to as a "Block Comment" in some documents.
This is a tag that appears within a Top-Level tag.
An example of an Inline tag is the <list> tag.

Applies To:

The "Applies To" entry indicates the Code Element (Property, Method, Event, Operator, etc.) that a tag is intended to be used with, within the XML Documentation Comments. If you have enabled the "XML Documentation" feature in Visual Studio, you will generally get a warning at compile time if you document a Code Element with a tag that is not valid for that Code Element. For information on turning on the "XML Documentation Feature", see the Notes section of this document.

Defined By:

The “Defined By” entry indicates the environments that support the tag.
The possible values for “Defined By” are: Microsoft; NDoc; and Sandcastle.

Microsoft: These are tags that are utilized by Visual Studio .NET and the C# and Visual Basic compilers. These tags are also supported by NDoc and Sandcastle.

NDoc: These are tags that are utilized by the NDoc documentation compiler but are not utilized by Microsoft or Sandcastle.

Not only does NDoc specify additional tags that are not supported by Microsoft and/or Sandcastle, but it also defines additional attributes for some of the tags that are defined by Microsoft.

Sandcastle: These are tags that are utilized by the Sandcastle documentation compiler but are not utilized by the Microsoft or NDoc compilers.

Sandcastle Helpfile Builder and additional components:
The Code Block Component, used standalone or in conjunction with the Sandcastle Help File Builder, supports several additional attributes on the <code> tag to support code colorization, importing code from external code files, and merging of code blocks from snippets.

Notes: At this time, all of the tags defined by Microsoft are also utilized by the NDoc and Sandcastle compilers, but not all NDoc and Sandcastle tags are utilized by Microsoft.

When the “Defined By” value is listed as “NDoc/Sandcastle”, generally the Sandcastle syntax for the tag is the same as the NDoc syntax. There are a couple of exceptions to this rule (specifically the <see href=““> and <seealso href=““> tags), but as a general rule, they are the same.

Syntax: The “Syntax” entry shows the syntax for the specific tag including attributes and parameters used by the tag.

The “Syntax” entry may be divided into “Microsoft” and “NDoc”, showing the syntax of the command when utilized by Microsoft and NDoc respectively.

Items enclosed within “[ ]” characters are optional.
For example, in the following sample, the “label” value is optional.

```
<see cref="member">[label]</see>
```

Items separated by a “|” character are the list of items that may be used.
For example, in the following sample the “type” value must be either “caution”, “inheritinfo” or “implementnotes”.

```
<note type="caution" | "inheritinfo" | "implementnotes">description</note>
```

Parameters: The “Parameters” entry lists the parameters and the possible values for the parameter and a short description of the parameter.

The “Parameters” entry may be divided into “Microsoft” and “NDoc”, showing the parameters used by the Microsoft and NDoc utilized versions of the tag.

Remarks: The “Remarks” entry lists general information regarding the tag and its usage.
The “Remarks” entry may be divided into “Microsoft” and “NDoc”, with the remarks under each of the headings coming from their respective online help documents.

Example: The “Example” generally shows a small code example with the current tag being applied to it. There may be more than one example if there are multiple syntaxes for a tag, or if the tag is utilized by more than one compiler.

Tags List

Tag Name	Defined By				Compiler Verifies Syntax
	Microsoft		NDoc	Sandcastle	
	VS 2003	VS 2005			
<c>	X				
<code>	X				
<event>			X		
<example>	X				
<exception>	X				X
<exclude>			X	X[1]	
<filterpriority>		X			
<include>	X				X
<list>	X				
<note>			X	X	
<overloads>			X	X	
<para>	X				
<param>	X				X
<paramref>	X				
<permission>	X				X
<preliminary>			X		
<remarks>	X				
<returns>	X				
<see>	X				X
<seealso>	X				X
<summary>	X				
<threadsafety>			X	X	
<typeparam>		X			X
<typeparamref>		X			
<value>	X				

1. When used with the Sandcastle Help File Builder.
(Not supported by Sandcastle directly. Sandcastle uses an API filter instead.)

XML Documentation Comments Matrix

	Defined By			Tag Type		Applies To											
Tag	Microsoft	NDoc	Sandcastle	Top-Level	Inline	Namespace	Class	Structure	Interface	Enum	Delegate	Field	Constant	Property	Method	Event	Operator
<c>	X				X		X	X	X	X	X	X	X	X	X	X	X
<code>	X				X									X	X	X	X
<event>	X			X											X		X
<example>	X			X			X	X	X	X	X	X	X	X	X	X	X
<exception>	X			X										X	X	X	X
<exclude>		X	X	X			X	X	X	X	X	X	X	X	X	X	X
<include>	X			X			X	X	X	X	X	X	X	X	X	X	X
<list>	X				X		X	X	X	X	X	X	X	X	X	X	X
<note>		X	X		X		X	X	X	X	X	X	X	X	X	X	X
<overloads>		X	X	X										X	X	X	X
<para>	X				X												
<param>	X			X							X			X	X		
<paramref>	X				X		X	X	X	X	X	X	X	X	X	X	X
<permission>	X			X								X	X	X	X	X	X
<preliminary>		X	X	X			X	X	X	X	X	X	X	X	X	X	X
<remarks>	X			X			X	X	X	X	X	X	X	X	X	X	X
<returns>	X			X											X		
<see>	X			X	X		X	X	X	X	X	X	X	X	X	X	X
<seealso>	X			X	X		X	X	X	X	X	X	X	X	X	X	X
<summary>	X			X		X[1]	X	X	X	X	X	X	X	X	X	X	X
<threadsafety>		X	X	X			X	X									
<typeparam>	X			X											X		
<typeparmref>	X				X										X		
<value>	X			X										X			

1. NDoc and Sandcastle Help File Builder support adding namespace comments using the GUI

Tag Descriptions

Tag: **<c>**

Type: Inline

Applies To: Can be used inline within any other markup.

Defined By: Microsoft

Syntax: **<c>text</c>**

Parameters:

text:

The text you would like to indicate as code.

Remarks:

- The <c> tag gives you a way to indicate that text within a description should be marked as code.
- Use the <code> tag to indicate multiple lines as code.

Example:

```
/// <summary>
///   <c>Increment</c> method increments the stored number by one.
/// </summary>
public void Increment()
{ number++; }
```

Tag: **<code>**

Type: Inline

Applies To: Can be used inline within any other markup.

Defined By: Microsoft, NDoc, Sandcastle

Syntax:

Microsoft/Sandcastle:

```
<code>content</code>
```

NDoc:

```
<code [lang="language" escaped="true|false"]>content</code>
```

Sandcastle Help File Builder (using Code Block Component):

```
<code [source="SourceFile"
        region="Region Name"
        lang="langId"
        numberLines="true|false"
        outlining="true|false"
        tabSize="###"
        title="Code Block Title"]>content</code>

Snippet merging:
        <code title="Nested code tag example" lang="VB.NET">
          <code source="..\Class1.vb" region="Snippet #1" />
          <code source="..\Class1.vb" region="Snippet #2" />
        </code>
```

Parameters:

Microsoft:

content:

The text you want marked as code.

NDoc extension:

lang="language"

Applies a filter for this language. (Optional)
See the “NDoc Language ID” table below.

escaped="true "

Escapes all reserved characters within content. (Optional)

Tag: <code> [continued]

Sandcastle Help File Builder (using Code Block Component extensions):

source="SourceFile"

- Allows importing code from an external source file.
- Unlike <include> this imports live, buildable code.
- It can also be used to import XML which works around encoding issues that make it difficult to specify well-formed XML in a <code> tag with Sandcastle.

region="Region Name"

When used with the **source** attribute, this limits the import to a specific section of code delimited with a #region/#endregion preprocessor attribute.

lang="langId"

- The language ID to use for colorizing the code block.
- See the “Sandcastle Help File Builder Language ID” table below.

numberLines="true|false"

Enables number lines within the code block.

outlining="true|false"

Enables collapsible regions for #region and #if preprocessor sections within the code block.

tabSize="###"

Replace tab characters (\x09) with the specified number of spaces to preserve formatting.

title="Code Title"

- The title that should appear over the code block.
- If omitted, the language name is used (i.e. C# or VB.NET).
- For a blank title, specify an empty string (title="").

Tag: <code> [continued]

Remarks:

Microsoft:

- The <code> tag gives you a way to indicate multiple lines as code.
- Use the <c> tag to indicate that a single line of text within a description should be marked as code.

NDoc:

- A language filter can be attached using the optional lang attribute.
- Standard languages are Visual Basic, C#, C++ and JScript.
- Multiple languages can be specified as a comma separated list such as "Visual Basic, C#, C++".
- The escaped attribute can be used to escape all reserved characters within the text.
- Note: All content within XML comments must be well-formed!!!

Language	Lang value
C#	lang="C#"
C++	lang="C++"
VB.NET	lang="Visual Basic"
Jscript	lang="JScript"

NDoc Language ID table

Sandcastle Help File Builder (using Code Block Component extensions):

- If a **lang** attribute is specified on a code block for C#, VB.NET, C++, or J#, the code block will be connected to the page's language filter.
- Unlike NDoc, only one language ID is supported in the attribute.
- Due to the way Sandcastle processes the comments, including XML directly in the code comments will result in a loss of all inter-element spacing and line feeds. Encode the angle brackets using **<** and **>** to avoid this issue.
- For larger examples where this is impractical, use the **source** attribute to import content from an XML file so that formatting can be preserved.
- Nested code tags can be used to merge multiple snippets into a single code example. Nested <code> tags should have only **source** and **region** attributes. All other attributes should be placed on the parent <code> tag.

Tag: <code> [continued]

Language ID (case-insensitive)	Language Syntax Used
cs, C#, CSharp	C#
cpp, C++, CPlusPlus	C++
c	C
jscript, JavaScript	JavaScript
vbnet, VB.NET, VB	VB/VB.NET
vbscript	VBScript
xml, xsl	XML
jsharp, J#	J#
sql, sql server, sqlserver	SQL script
Anything else (i.e. "none")	No highlighting

Sandcastle Help File Builder Language ID table

Tag: <code> [continued]

Example:

Microsoft/Sandcastle:

See the <example> tag for an example of how to use the <code> tag.

NDoc:

```
/// <summary>
///   The GetZero method.
/// </summary>
/// <example>
///         This sample shows how to call the GetZero method.
///   <code lang="C#">
///         class TestClass
///         {
///               static int Main()
///               { return GetZero(); }
///         }
///   </code>
/// </example>
public static int GetZero()
{ return 0; }


/// <summary>
///   Load the configuration from the specified configuration file.
/// </summary>
/// <example>
///   The configuration file should contain the following elements:
///         <code escaped="true">
///               <config>
///                     <reportServer name="TestServer" />
///                     <outputPath location="C:\Test" />
///               </config>
///         </code>
/// </example>
public void LoadConfig( string file )
{
      // load config file …
}
```

Sandcastle Help File Builder (using Code Block Component extensions):

```
/// <summary>
///   The GetZero method.
/// </summary>
/// <example>
///   Uses SHFB code colorization (C#), default title.
///   <code lang="cs">
///         class TestClass
///         {
///           static int Main()
///           { return GetZero(); }
///         }
///   </code>
/// </example>
public static int GetZero()
{ return 0; }
```

Tag: <code> [continued]

```
/// <summary>
///   The GetZero method.
/// </summary>
///   Uses SHFB code colorization (C#), user-defined title.
///   <code lang="cs" title="GetZero Demo">
///   class TestClass
///   {
///       static int Main()
///       { return GetZero(); }
///   }
///   </code>
/// </example>
public static int GetZero()
{ return 0; }


/// <summary>
///   The GetZero method.
/// </summary>
///   Import the code from a source code file
///   <code lang="cs" source="..\Code\GetZeroDemo.cs" />
/// </example>
public static int GetZero()
{ return 0; }


/// <summary>
///   The GetZero method.
/// </summary>
///   Import the code from a region within the source code file
///   <code lang="cs" source="..\Code\GetZeroDemo.cs"
///       region="Main Method" />
/// </example>
public static int GetZero()
{ return 0; }


/// <summary>
///   The GetZero method.
/// </summary>
///   Merge various snippets into a single example
///   <code title="Nested code tag example" lang="VB.NET">
///     <code source="..\Class1.vb" region="Snippet #1" />
///     <code source="..\Class1.vb" region="Snippet #2" />
///   </code>
/// </example>
public static int GetZero()
{ return 0; }
```

Tag: **<event>**

Type: Top-Level

Applies To: Method

Defined By: NDoc

Syntax: **<event cref="member">description</event>**

Parameters:

cref="member":

- A reference to an event that is available from the current compilation environment.
- The compiler checks that the given event exists and translates “member” to the canonical element name in the output XML.
- member must appear within double quotation marks (" ").

description:

A description of the event.

Remarks:

- This section defines the events that can be raised by the method.
- There can be multiple <event> tags per method.

Example:

```
/// <summary>
///   Summary for the class
/// </summary>
/// <event cref="SampleEventHandler">Sample Event</event>
public class Publisher
{
  // Declare the delegate (if using non-generic pattern).
  public delegate void SampleEventHandler( object sender, SampleEventArgs e );

  // Declare the event.
  public event SampleEventHandler SampleEvent;

  // Wrap the event in a protected virtual method
  // to enable derived classes to raise the event.
  protected virtual void RaiseSampleEvent()
  {
       // Raise the event by using the () operator.
       SampleEvent( this, new SampleEventArgs( "Hello" ) );
  }
}
```

Tag: **<example>**

Type: Top-Level

Applies To: All types and members.

Defined By: Microsoft

Syntax: **<example>description</example>**

Parameters:

description:

A description of the code sample.

Remarks:

- The <example> tag lets you specify an example of how to use a method or other library member.
- This commonly involves using the <code> tag.

Example:

```
/// <summary>
///   The GetZero method.
/// </summary>
/// <example>
///         This sample shows how to call the GetZero method.
///   <code>
///         class TestClass
///         {
///               static int Main()
///               { return GetZero(); }
///         }
///   </code>
/// </example>
public static int GetZero()
{ return 0; }
```

Tag: **<exception>**

Type: Top-Level

Applies To: Property, Method, Event, Operator, Type Conversion

Defined By: Microsoft

Syntax: **<exception cref="member">description</exception>**

Parameters:

cref="member":

- A reference to an exception that is available from the current compilation environment.
- The compiler checks that the given exception exists and translates “member” to the canonical element name in the output XML.
- member must appear within double quotation marks (" ").

description:

A description of the exception.

Remarks:

- The <exception> tag lets you specify which exceptions can be thrown.
- This tag can be applied to definitions for methods, properties, events, and indexers.

Example:

```
/// comment for class
public class EClass : System.Exception
{
    // comment for class
}

/// comment for class
class TestClass
{
    /// <exception cref="System.Exception">Thrown when...</exception>
    public void DoSomething()
    {
        try
        {
        }
        catch (EClass)
        {
    }
}
```

Tag: **<exclude>**

Type: Top-Level

Applies To: All Types and Members.

Defined By: NDoc, Sandcastle Help File Builder

Syntax: **<exclude/>**

Parameters:

None

Remarks:

- This tag takes precedence over any visibility options.
- Note that Sandcastle does not support this tag. Instead, it uses an API filter in the MRefBuilder.config file to specify which namespaces, types, and members should be excluded.
- The Sandcastle Help File Builder will use this tag if it finds it and will automatically add the marked members to the API filter. It also provides support for the API filter directly (the ApiFilter property in version 1.5.1.0 and after).

  The dnrTV (dot Net Rocks TV) Sandcastle episode contains a short explanation of the APIFilter element and its usage (for links to the dnrTV Sandcastle episode, see the Resources section of this document).

  The API Filter offers the ability to filter namespaces, types and members so that their HTML topics are not generated by the BuildAssembler.exe utility. Topics are filtered by their API names in the apiFilter element of the MRefBuilder.config file.

  The MRefBuilder.exe program outputs an xml file with reflection information for a set of input assemblies, which is then processed using XSL transformations and input into the BuildAssembler.exe utility to generate the HTML topics.

  The API Filter is used to prevent specific reflection information from being added to the reflection file, therefore BuildAssembler.exe will not generate the corresponding HTML topics.

  Generic types and members can be filtered by referencing their names using compiler syntax (e.g., MyGenericType`1). Individual method overloads cannot be filtered since signatures are not currently supported.

Example:

```
/// <summary>
///   The GetZero method.
/// </summary>
/// <exclude/>
public static int GetZero()
{ return 0; }
```

Tag: **<filterpriority>**

Type: Top-Level

Applies To: Method

Defined By: Microsoft

Syntax: **<filterpriority>value</ filterpriority >**

Parameters:

value:

- Used by the Visual Basic editor in order to determine whether a member should appear on the Common or All tab when the completion list is shown.

Remarks:

None

Example:

```
/// <summary>
///   Loads the XML.
/// </summary>
/// <filterpriority>1</filterpriority>
public void LoadXML( string XML )
{
      if (string.IsNullOrEmpty(XML))
      {throw new ArgumentNullException();}
}
```

Tag: **<include>**

Type: Top-Level

Applies To: All Types and Members.

Defined By: Microsoft

Syntax: **<include file='filename' path='tagpath[@name="id"]' />**

Parameters:

filename:

- The name of the file containing the documentation.
- The file name can be qualified with a path.
- Enclose filename in single quotation marks (' ').

tagpath:

- The path of the tags in filename that leads to the tag name.
- Enclose the path in single quotation marks (' ').

name:

The name specifier in the tag that precedes the comments; name will have an id.

id:

- The ID for the tag that precedes the comments.
- Enclose the ID in double quotation marks (" ").

Remarks:

- The <include> tag lets you refer to comments in another file that describe the types and members in your source code. The comments are imported into the generated XML comments file at build time by the compiler.
- This is an alternative to placing documentation comments directly in your source code file.
- The <include> tag uses the XML XPath syntax.
- Refer to XPath documentation for ways to customize your <include> use.

Tag: <include> [continued]

Example:

NOTE: This is a multi-file example.

The first file, which uses <include>, is listed below:

```
/// <include file='XML_include_tag.doc' path='MyDocs/MyMembers[@name="test"]/*' />
class Test
{
	static void Main()
	{ }
}

/// <include file='XML_include_tag.doc' path='MyDocs/MyMembers[@name="test2"]/*' />
class Test2
{
	public void Test()
	{ }
}
```

The second file, XML include tag.doc, contains the following documentation comments:

```
<MyDocs>
	<MyMembers name="test"><summary>The summary for this type.</summary></MyMembers>
	<MyMembers name="test2"><summary>The summary for this other type.</summary></MyMembers>
</MyDocs>
```

Compiler Output:

```
<?XML version="1.0"?>
	<doc>
		<assembly>
			<name>XML_include_tag</name>
		</assembly>

		<members>
			<member name="T:Test">
				<summary>
					The summary for this type.
				</summary>
			</member>
			<member name="T:Test2">
				<summary>
					The summary for this other type.
				</summary>
			</member>
		</members>
</doc>
```

Tag: **<list>**

Type: Inline

Applies To: Can be used inline within any other markup.

Defined By: Microsoft

Syntax:

```
<list type="bullet" | "number" | "table">
        <listheader>
                <term>term</term>
                <description>description</description>
        </listheader>
        <item>
                <term>term</term>
                <description>description</description>
        </item>
</list>
```

Parameters:

type=””

The type of list to be created (bullet, number or table)

term:

The term to that will be defined in description.

description:

Either an item in a bullet or numbered list or the definition of a term.

Remarks:

- The <listheader> block is used to define the heading row of either a table or definition list.
- When defining a table, you only need to supply an entry for term in the heading.
- Each item in the list is specified with an <item> block.
- When creating a definition list, you will need to specify both term and description.
- However, for a table, bulleted list, or numbered list, you only need to supply an entry for description.
- A list or table can have as many <item> blocks as needed.

Example:

```
/// <summary>
///   The GetZero method.
///   <list type="number">
///         <item>
///               <description>Item 1.</description>
///         </item>
///         <item>
///               <description>Item 2.</description>
///         </item>
///   </list>
/// </summary>
public static int GetZero()
{ return 0; }
```

Tag: **<note>**

Type: Inline

Applies To: Can be used inline within any other markup.

Defined By: NDoc, Sandcastle

Syntax: **<note type="caution" | "inheritinfo" | "implementnotes">description</note>**

Parameters:

type="":

Type of note being displayed

description:

Descriptive text for the note tag

Remarks:

The <note> tag produces a formatted note block.

Example:

```
/// <summary>
///          <c>Increment</c> method increments the stored number by one.
///          <note type="caution">
///                note description here
///          </note>
///          <preliminary/>
/// </summary>
public void Increment()
{ number++; }
```

Tag: **<overloads>**

Type: Top-Level

Applies To: Property, Method, Event, Operator.

Defined By: NDoc, Sandcastle

Syntax:

Short Form

<overloads>summary_description</overloads>

Expanded Form

<overloads>
<summary>
summary_description
</summary>
[<remarks>remarks_description</remarks>]
[<example>examples_description</example>]
</overloads>

Parameters: None

Remarks:

- The <overloads> tag provides documentation that applies to all the overloads of a member.
- This information will appear on the Overloads topic page for the members.
- This tag only needs to be specified on the first overload.
- The tag can have two forms:
    1. In the short form, it includes only one or more text blocks that are treated as the summary.
    2. In the expanded form, it can include one or more applicable section tags (summary, remarks and example).

Examples:

Short Form:

```
/// <summary>Decrements the number by 1</summary>
/// <overloads>This method has two overloads</overloads>
public void Dec()
{ number--; }

/// <summary>Decrements the number by amount</summary>
/// <param name="amount">The amount to decrement it by</param>
public void Dec( int amount )
{ number -= amount; }
```

Tag: <overloads> [continued]

Expanded Form:

```
/// <summary>Decrements the number by 1</summary>
/// <overloads>
///   <summary> This method has two overloads</summary>
///   <remarks>remarks description</remarks>
///   <example>code example</example>
/// </overloads>
public void Dec()
{ number--; }

/// <summary> Decrements the amount by number </summary>
/// <param name="amount">The amount to decrement it by</param>
public void Dec( int amount )
{ number -= amount; }
```

Tag: <para>

Type: Inline

Applies To: Can be used inline within any other markup.

Defined By: Microsoft, NDoc

Syntax: Microsoft:

```
<para>content</para>
```

NDoc:

```
<para [lang="language"]>content</para>
```

Parameters:

Microsoft:

content:

The text of the paragraph.

NDoc:

content:

The text of the paragraph.

lang="language"

Applies a filter for this language. (Optional)

Remarks: Microsoft:

- The <para> tag is for use inside a tag, such as <summary>, <remarks>, or <returns>, and lets you add structure to the text.
- As an alternative, you can use the HTML or tags to create paragraphs.

NDoc:

- This tag is for use inside a tag, such as <summary>, <remarks>, or <returns>, and lets you add structure to the text.
- A language filter can be attached using the optional lang attribute.
- Standard languages are Visual Basic, C#, C++ and JScript.
- Multiple languages can be specified as a comma separated list such as "Visual Basic, C#, C++".

Language	Lang value
C#	lang="C#"
C++	lang="C++"
VB.NET	lang="Visual Basic"
Jscript	lang="JScript"

NDoc Language ID table

Tag: <para> [continued]

Example:

Microsoft:

```
/// <summary>
///   Increment method increments the stored number by a specified.
///   <para>
///         This is the first paragraph
///   </para>
///   <para>
///         This is the second paragraph
///   </para>
/// </summary>
public void Increment( int step )
{ number = number + step; }
```

NDoc:

```
/// <summary>
///       Increment method increments the stored number by a specified.
///   <para>
///         This is the first paragraph
///   </para>
///   <para lang="C#">
///         This is the second paragraph
///   </para>
/// </summary>
public void Increment( int step )
{ number = number + step; }
```

Tag: <param>

Type: Top-Level

Applies To: Property, Method, Event, Operator.

Defined By: Microsoft

Syntax: **<param name="name">description</param>**

Parameters:

name:

The name of a method parameter.
Enclose the name in double quotation marks (" ").

description:

A description for the parameter.

Remarks:

- The <param> tag should be used in the comment for a method declaration to describe one of the parameters for the method.
- The text for the <param> tag will be displayed in IntelliSense, the Object Browser, and in the Code Comment Web Report.

Example:

```
///   <summary>
///         Increment method increments the stored number by a specified
///   </summary>
///   <param name="step"> specified step</param>
public void Increment( int step )
{ number = number + step; }
```

Tag: <paramref>

Type: Inline

Applies To: Can be used inline within any other markup.

Defined By: Microsoft

Syntax: **<paramref name="name"/>**

Parameters:

name:

The name of the parameter to refer to. Enclose the name in double quotation marks (" ").

Remarks:

- The <paramref> tag gives you a way to indicate that a word in the code comments, for example in a <summary> or <remarks> block refers to a parameter.
- The XML file can be processed to format this word in some distinct way, such as with a bold or italic font.

Example:

```
///   <summary>
///         Increment method increments the stored number by a specified
///         <paramref name="step"/>.
///   </summary>
///   <param name="step"> specified step</param>
public void Increment( int step )
{ number = number + step; }
```

Tag: <permission>

Type: Top-Level

Applies To: All Members.

Defined By: Microsoft

Syntax: **<permission cref="member">description</permission>**

Parameters:

cref="member":

- A reference to a member or field that is available to be called from the current compilation environment.
- The compiler checks that the given code element exists and translates “member” to the canonical element name in the output XML.
- member must appear within double quotation marks (" ").

description:

A description of the access to the member.

Remarks:

- The <permission> tag lets you document the access of a member.
- The PermissionSet class lets you specify access to a member

Example:

```
///   <summary>
///         Increment method increments the stored number by a specified
/// </summary>
///   <permission cref="System.Security.PermissionSet">
///         Everyone can access this method.
///   </permission>
public void Increment( int step )
{ number = number + step; }
```

Tag: <preliminary>

Type: Top-Level

Applies To: All Types and Members.

Defined By: NDoc / Sandcastle

Syntax: **<preliminary>[description]</preliminary>**

Parameters:

description

An optional textual message or warning that replaces the default preliminary warning.

Remarks:

- The <preliminary> tag marks the documentation for the current item as preliminary.
- If the empty form of this tag is used (<preliminary/>) the default message of

  "[This is preliminary documentation and subject to change.]"

  will be included in the generated help topic.
- If a value is supplied for the content of this tag, that value will appear in the help topic, replacing the default message.
- You can format the message text using the para and list tags, but this is not required.

Sandcastle Help File Builder/NDoc:

- Entire help titles can be marked preliminary using the Preliminary documenter/project setting.

Example:

```
// The class summary will get the default message
/// <preliminary/>
public class MyClass
{
   /// <preliminary>
   ///      <para>
   ///            This method is just for testing right now.
   ///            It might be removed in the near future
   ///      </para>
   /// </preliminary>
   public void Dump ()
   {
      // Code for Dump goes here
   }
}
```

Tag: <remarks>

Type: Top-Level

Applies To: All Types and Members.

Defined By: Microsoft

Syntax: **<remarks>description</remarks>**

Parameters:

description:

A description of the member.

Remarks:

- The <remarks> tag is used to add information about a type, supplementing the information specified with <summary>.
- This information is displayed in the Visual Studio Object Browser.

Example:

```
/// <summary>
///   Increment method increments the stored number by a specified step.
/// </summary>
/// <remarks>
///   You may have some additional information about this class.
/// </remarks>
public void Increment(int step)
{ number = number + step; }
```

Tag: **<returns>**

Type: Top-Level

Applies To: Method.

Defined By: Microsoft

Syntax: **<returns>description</returns>**

Parameters:

description:

A description of the return value.

Remarks:

- The <returns> tag should be used in the comment for a method declaration to describe the return value.
- This information is displayed in the Visual Studio Object Browser.

Example:

```
/// <summary>
///   Increment method increments the stored number by a specified step.
/// </summary>
/// <returns>Returns nothing</returns>
public void Increment(int step)
{ number = number + step; }
```

Tag: **<see>**

Type: Top-Level (Microsoft) or Inline (NDoc/Sandcastle)

Applies To: Can be used inline within any other markup.

Defined By: Microsoft, NDoc, Sandcastle

Syntax:

Microsoft:

```
<see cref="member"/>
```

NDoc/Sandcastle:

```
<see cref="member">[label]</see>
```

OR

```
<see langword="null | sealed | static | abstract | virtual | true | false|other_key_word"/>
```

NDoc Only:

```
<see href="URL">[label]</see>
```

Parameters:

Microsoft:

cref="member":

- A reference to a member or field that is available to be called from the current compilation environment.
- The compiler checks that the given code element exists and passes “member” to the element name in the output XML.
- Place member within double quotation marks (" ").

NDoc Only:

href = "URL":

A reference to a Top-Level resource at the address given by the URL.

NDoc / Sandcastle:

label:

text to display as the link

langword:

- A common .NET language keyword.
- These keywords are highlighted, and, in some cases, expanded into descriptive phrases (see remarks for further details).
- Note that although the syntax above only shows the common keywords, any word specified will be highlighted.

Tag: <see> [continued]

Remarks:

Microsoft:

- The <see> tag lets you specify a link from within text.
- Use <seealso> to indicate that text should be placed in a See Also section.

NDoc:

- Use <seealso> to indicate text that you might want to appear in a See Also section.
- **Note:** As of release 1.3, the MSDN and VS.NET documenters will only create a link on the first occurrence of each unique cref specified within a documentation section; further <see> tags will just be highlighted.
- This improves the readability of the documentation.

langword	keyword expansions
null	null reference (Nothing in Visual Basic)
sealed	sealed (NotInheritable in Visual Basic)
static	static (Shared in Visual Basic)
abstract	abstract (MustInherit in Visual Basic)
virtual	virtual (CanOverride in Visual Basic)

Note: Microsoft defines the <see> tag as both a Top-Level and an Inline tag, while NDoc and Sandcastle define it as an Inline tag.

Note: While not documented as a Top-Level tag, both Sandcastle and NDoc support the <see> tag as a Top-Level tag.

Tag: <see> [continued]

Example:

Microsoft:

Top-Level tag:

```
class Class1
{
      /// <summary>
      ///   Summary of information about this class.
      /// </summary>
      /// <see cref="System.String"/>
      public string GetString()
      {
            // Code for GetString goes here
      }
}
```

Inline tag:

```
/// <summary>
///   Summary of information about this class
///   See <see cref="System.String"/>
///   for more information about the string class
/// </summary>
public string GetString()
{
      // Code for GetString goes here }
}
```

Tag: <see> [continued]

NDoc/Sandcastle:

Inline tag (cref):

```
/// <summary>
///   You may have some primary information about this class.
///         See <see cref="System.String"/>
///         for more information about the string class
/// </summary>
public string GetString()
{
      // Code for GetString goes here
}
```

Inline tag (href, NDoc only):

```
/// <summary>
///   You may have some primary information about this class.
///         See <see href="http://www.someurl.com"/>
///         for more information about the string class
/// </summary>
public string GetString()
{
      // Code for GetString goes here
}
```

Inline tag (langword):

```
/// <summary>
///   You may have some primary information about this class.
///   See <see langword="null"/>
///   for more information about null values
/// </summary>
public string GetString()
{
      // Code for GetString goes here
}
```

Tag: **<seealso>**

Type: Top-Level (Microsoft) or Inline (NDoc/Sandcastle)

Applies To: All Types and Members.

Defined By: Microsoft, NDoc, Sandcastle

Syntax:

Microsoft:

```
<seealso cref="member"/>
```

NDoc/Sandcastle:

```
<seealso cref="member">[label]</seealso>
```

NDoc Only:

```
<seealso href="URL">[label]</seealso>
```

Parameters:

Microsoft:

cref="member":

- A reference to a member or field that is available to be called from the current compilation environment.
- The compiler checks that the given code element exists and passes “member” to the element name in the output XML.
- member must appear within double quotation marks (" ").

NDoc/Sandcastle:

label: text to display as the link

cref = "member":

- A reference to a member or field that is available to be called from the current compilation environment.
- The compiler checks that the given code element exists and passes “member” to the element name in the output XML.
- member must appear within double quotation marks (" ").

NDoc:

href = "URL":

A reference to a Top-Level resource at the address given by the URL.

Tag: <seealso> [continued]

Remarks:

Microsoft:

- The <seealso> tag lets you specify the text that you might want to appear in a See Also section.
- Use <see> to specify a link from within text.

NDoc:

- Use <see> to specify a link from within text.
- Do not include this tag in the <remarks> section.

Note: Microsoft defines the <seealso> tag as both a Top-Level and an Inline tag, while NDoc and Sandcastle define it as an Inline tag.

Note: While not documented as a Top-Level tag, both Sandcastle and NDoc support the <seealso> tag as a Top-Level tag.

Examples:

Microsoft:

Top-Level tag:

```
/// <summary>
///   You may have some primary information about this class.
/// </summary>
/// <seealso cref="System.String"/>
public string GetString()
{
      // Code for GetString goes here
}
```

Inline tag:

```
/// <summary>
///   You may have some primary information about this class.
///   See <seealso cref="System.String"/>
///   for information about the string class
/// </summary>
public string GetString()
{
      // Code for GetString goes here
}
```

Tag: <seealso> [continued]

NDoc/Sandcastle:

Inline tag (cref):

```
/// <summary>
///   You may have some primary information about this class.
///         See <seealso cref="System.String"/>
///         for information about the string class
/// </summary>
public string GetString()
{
      // Code for GetString goes here
}
```

Inline tag (href, NDoc only):

```
/// <summary>
///   You may have some primary information about this class.
///         See < seealso href="http://www.someurl.com"/>
///         for information about the string class
/// </summary>
public string GetString()
{
      // Code for GetString goes here
}
```

Tag: **<summary>**

Type: Top-Level

Applies To: All Types and Members.

Defined By: Microsoft

Syntax: **<summary>description</summary>**

Parameters:

description:

A summary of the object.

Remarks:

- The <summary> tag should be used to describe a type or a type member.
- Use <remarks> to add supplemental information to a type description.
- The text for the <summary> tag is the only source of information about the type in IntelliSense, and is also displayed in the Object Browser.

Example:

```
/// <summary>
///   Gets or sets a
///   <see cref="T:System.Windows.Media.Brush"/>
///   used to fill the area between the borders of a
///   <see cref="T:System.Windows.Controls.InkCanvas"/>
///   see <see langword="null"/> as reference
/// </summary>
public int MyProp
{
      get { return _myProp; }
      set { _myProp = value; }
}
```

Tag: **<threadsafety>**

Type: Top-Level

Applies To: Class, Structure.

Defined By: NDoc, Sandcastle

Syntax: **<threadsafety static="true|false" instance="true|false"/>**

Parameters:

static="true|false"

Indicates whether static members of this class are safe for multi-threaded operations.

instance="true|false"

Indicates whether members of instances of this type are safe for multi-threaded operations.

Remarks:

The <threadsafety> tag is used to describe how a class or structure behaves in multi-threaded scenarios.

Example:

```
/// <threadsafety static="true" instance="false"/>
public class MyClass
{
   /// not safe across threads
   public void InstanceMethod()
   {
         // Code for instanceMethod goes here
   }

   /// safe across threads
   public static void StaticMethod()
   {
         // Code for StaticMethod goes here
   }
}
```

Tag: **<typeparam>**

Type: Top-Level

Applies To: Method (Generic)

Defined By: Microsoft

Syntax: **<typeparam name="name">description</typeparam>**

Parameters:

name:

The name of the type parameter.
Enclose the name in double quotation marks (" ").

description:

A description for the type parameter.

Remarks:

- The <typeparam> tag should be used in the comment for a generic type or method declaration to describe a type parameter.
- Add a tag for each type parameter of the generic type or method.
- For more information, see Generics (C# Programming Guide).
- The text for the <typeparam> tag will be displayed in IntelliSense, the Object Browser code comment web report.

Example:

```
/// comment for class
public class TestClass
{
      /// <summary>
      ///   Creates a new array of arbitrary type <typeparamref name="T"/>
      /// </summary>
      /// <typeparam name="T">The element type of the array</typeparam>
      public static T[] mkArray<T>( int n )
      {
            return new T[n];
      }
}
```

Tag: **<typeparamref>**

Type: Inline

Applies To: Inline within any other markup, for Generic methods

Defined By: Microsoft

Syntax: **<typeparamref name="name"/>**

Parameters:

name:

The name of the type parameter.

Enclose the name in double quotation marks (" ").

Remarks:

- For more information on type parameters in generic types and methods, see Generics.
- Use this tag to enable consumers of the documentation file to format the word in some distinct way, for example in italics.

Example:

```
/// <summary>
///   <para>
///         <typeparam name="T">The element type to swap</typeparam>
///   </para>
///   <para>
///         Swap data of type <typeparamref name="T"/>
///   </para>
/// </summary>
public void Swap<T>( ref T lhs, ref T rhs )
{
      T temp;
      temp = lhs;
      lhs = rhs;
      rhs = temp;
}
```

Tag: **<value>**
Type: Top-Level
Applies To: Property
Defined By: Microsoft
Syntax: **<value>property-description</value>**
Parameters:

property-description:
A description for the property.

Remarks:

- The <value> tag lets you describe the value that a property represents.
- Note that when you add a property via code wizard in the Visual Studio .NET development environment, it will add a <summary> tag for the new property. You should then manually add a <value> tag to describe the value that the property represents.
- Visual Studio doesn't use <value>, but rather uses <summary> for IntelliSense.

Example:

```
/// text for class Employee
public class Employee
{
    private string _name;

      /// <summary>
      ///   The Name property represents the employee's name.
      /// </summary>
      /// <value>
      ///   The Name property gets/sets the _name data member.
      /// </value>
    public string Name
    {
        get {return _name;}
        set {_name = value;}
    }
}

/// text for class MainClass
public class MainClass
{
      /// text for Main
      static void Main()
      {
            // Code for Main goes here
      }
}
```

Generic Type cref Example

The following example shows how to make a cref reference to a generic type.

```
// the following cref shows how to specify the reference, such that,
// the compiler will resolve the reference
/// <see cref="C{T}" />

class A { }

// the following cref shows another way to specify the reference,
// such that, the compiler will resolve the reference
// <summary cref="C&lt;T&gt;">
// the following cref shows how to hard-code the reference
/// <see cref="T:C`1" />

class B { }

/// <summary>
/// A generic class.
/// </summary>
/// <typeparam name="T"></typeparam>
class C<T> { }
```

NOTE:

The second format above (<summary cref="T:C`1"> will vary based on the number of generic parameters. For example, Dictionary<TKey, TValue> would be Dictionary`2.
The first way is easier to specify as you just substitute curly braces for angle brackets.

Sandcastle Test.cs Example

```
using System;
using System.Collections.Generic;
using System.Diagnostics.CodeAnalysis;
using System.Runtime.Serialization;

namespace TestNamespace
{
	/// <summary>
	///		Tests whether sandcastle can handle all c# tags as defined at
	///			http://msdn2.microsoft.com/en-us/library/5ast78ax.aspx.
	///		Comments of method "Increment (int step)" include almost all tags.
	///		Method "Swap" is used to test generics tags, such as "typeparam".
	///		<threadsafety static="true" instance="false"/>
	/// </summary>
	[Serializable()]
	public class StoredNumber
	{
		/// <summary>
		///		Initializes the stored number class with a starting value.
		///	</summary>
		public StoredNumber( int value )
		{ number = value; }

		private int number;

		/// <preliminary>
		///	<para>
		///		This method is just for testing right now.
		///		It might be removed in the near future
		///	</para>
		/// </preliminary>
		public void PreliminaryTest()
		{
		}

		/// <overloads>
		///	<summary>
		///		test overloads tag
		///	</summary>
		/// </overloads>
		public void Dec()
		{ number--; }

		/// <summary>
		///		dec by a specified step
		/// </summary>
		/// <param name="step"></param>
		public void Dec( int step )
		{ number -= step; }
```

XML Documentation Comments Example [continued]

```
/// <summary>
///         <c>Increment</c> method increments the stored number by one.
///         <note type="caution">
///               note description here
///         </note>
///         <preliminary/>
/// </summary>
public void Increment()
{ number++; }

///   <summary>
///         <c>Increment</c> method increments the stored number by a
///         specified <paramref name="step"/>.
///         <list type="number">
///               <item>
///                     <description>Item 1.</description>
///               </item>
///               <item>
///                     <description>Item 2.</description>
///               </item>
///         </list>
///         <para>
///               see <see cref="System.Int32"/>
///         </para>
///         <para>
///               seealso <seealso cref="System.Int64"/>
///         </para>
/// </summary>
/// <remarks>
///   You may have some additional information about this class.
/// </remarks>
/// <example>
///   This sample shows how to call the GetZero method.
///   <code>
///         class TestClass
///         {
///               static int Main()
///               { return GetZero(); }
///         }
///   </code>
/// </example>
/// <exception cref="System.Exception">Thrown when...</exception>
/// <param name="step"> specified step</param>
/// <permission cref="System.Security.PermissionSet">
///   Everyone can access this method.
/// </permission>
/// <returns>Returns nothing</returns>
/// <value>The Name property gets/sets the _name data member.</value>
public void Increment( int step )
{ number = number + step; }
```

XML Documentation Comments Example [continued]

```
            /// <summary>
            ///   <para>
            ///         <typeparam name="T">The element type to swap</typeparam>
            ///   </para>
            ///   <para>
            ///         Swap data of type <typeparamref name="T"/>
            ///   </para>
            /// </summary>
            public void Swap<T>( ref T lhs, ref T rhs )
            {
                  T temp;
                  temp = lhs;
                  lhs = rhs;
                  rhs = temp;
            }

            ///   <summary>Gets the stored number. </summary>
            public int Value
            {
                  get { return (number); }
            }

            private int _myProp;

            ///   <summary>
            ///         Gets or sets a
            ///         <see cref="T:System.Windows.Media.Brush"></see>
            ///         used to fill the area between the borders of a
            ///         <see cref="T:System.Windows.Controls.InkCanvas"></see>.
            ///         see <see langword="null"/> as reference
            ///   </summary>
            public int MyProp
            {
                  get { return _myProp; }
                  set { _myProp = value; }
            }
      }
}
```

Links

.NET Documentation Guidelines (version 2.5)

http://p3net.mvps.org/downloads/docs/DocGuidelines.doc

XML Documentation in C# (Anson Horton, C# Compiler Program Manager)

http://cyrino.members.winisp.net/9112006/XMLDocs.doc

C# XML Documentation (Alan Dean)

http://thoughtpad.net/alan-dean/cs-xml-documentation.html

Microsoft Developer Documentation and Help System forum

http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=66&SiteID=1

NDoc User's Guide

http://ndoc.sourceforge.net/usersguide.html

MSDN XML Documentation Comments Resources

Visual Studio

- How to: Generate XML Documentation for a Project
  http://msdn2.microsoft.com/en-us/library/x4sa0ak0(VS.80).aspx

C#

- C# Programmer's Reference:
  XML Documentation
  Microsoft Visual Studio 2003/.NET Framework 1.1
  http://msdn2.microsoft.com/en-us/library/b2s063f7(VS.71).aspx

- C# Programmer's Reference:
  Recommended Tags for Documentation Comments
  Microsoft Visual Studio 2003/.NET Framework 1.1
  http://msdn2.microsoft.com/en-us/library/5ast78ax(VS.71).aspx

- C# Programming Guide:
  Recommended Tags for Documentation Comments (C# Programming Guide)
  Microsoft Visual Studio 2005/.NET Framework 2.0
  http://msdn2.microsoft.com/en-us/library/5ast78ax(vs.80).aspx

- C# Programming Guide:
  XML Documentation Comments (C# Programming Guide)
  Microsoft Visual Studio 2005/.NET Framework 2.0
  http://msdn2.microsoft.com/en-us/library/b2s063f7(vs.80).aspx

Links [continued]

MSDN XML Documentation Comments Resources [continued]:

Visual Basic

- Documenting Your Code with XML (Visual Basic)
  Microsoft Visual Studio 2005/.NET Framework 2.0
  http://msdn2.microsoft.com/en-us/library/ms172652(VS.80).aspx

- Visual Basic Language Reference
  How to: Create XML Documentation in Visual Basic
  http://msdn2.microsoft.com/en-us/library/xzysab5k(VS.80).aspx

Sandcastle Information

Sandcastle Docs
http://www.sandcastledocs.com/

Sandcastle Blogs
http://blogs.msdn.com/sandcastle/

Sandcastle Wiki
http://www.sandcastledocs.com/

Sandcastle dnrTV video episode
http://www.dnrtv.com/default.aspx?showNum=84

Sandcastle GUI programs

Help File Builder - GUI and console mode builder plus various custom build components including

a. Code Block Component
b. Post Transform Component
c. Show Missing Component
d. Version Info Component

http://www.codeplex.com/SHFB

Other Sandcastle GUI programs
http://www.sandcastledocs.com/Wiki%20Pages/Home.aspx

Sandcastle on CodePlex
http://codeplex.com/Tagging/TagDetail.aspx?Tagname=Sandcastle

NDoc Information

Website
http://ndoc.sourceforge.net

Document Links

Contact Information

Web Site

http://www.dynicity.com/products/XMLDocComments.aspx

Email

mailto://xmldoccomments@dynicity.com

Document Download Location

Windows Installer

http://www.dynicity.com/downloads/XMLDocCommentsGuide.exe

Individuals who contributed to this document

I would like to thank the following individuals who have helped create this document by answering my many questions, fixing my mistakes, making suggestions, and generally making this a better document.

Eric Woodruff
Developer – Sandcastle Help File Builder application and various custom build components
http://www.codeplex.com/SHFB

Anand Raman
Microsoft Group Manager, Developer Division
http://www.sandcastledocs.com/

Anson Horton
Program Manager – Microsoft C# Team
http://blogs.msdn.com/ansonh/default.aspx