**Практическое руководство. Вставка XML-комментариев для создания документации**

* Статья
* 09.12.2021
* Чтение занимает 2 мин
  + [](https://github.com/mikadumont)
  + [](https://github.com/olprod)

Были ли сведения на этой странице полезными?

Visual Studio может помочь вам документировать элементы кода, такие как классы и методы, автоматически формируя стандартную структуру комментариев для XML-документации. Во время компиляции можно создать XML-файл, содержащий комментарии для документации. Чтобы включить этот параметр, выберите **Generate a file containing API documentation** (Создать файл, содержащий документацию по API) на вкладке **Сборка** > **Выходные данные** в свойствах проекта.

**Совет**

Если вы хотите настроить нестандартное имя и расположение для файла документации, добавьте свойство [**DocumentationFile**](https://docs.microsoft.com/ru-ru/dotnet/core/project-sdk/msbuild-props#documentationfile) в файл *.csproj*, *.vbproj* или *.fsproj*.

Созданный компилятором XML-файл можно распространять вместе со сборкой .NET, чтобы Visual Studio и другие интегрированные среды разработки могли использовать IntelliSense для отображения кратких сведений о типах и элементах. Кроме того, XML-файл можно запускать с помощью таких средств, как [DocFX](https://dotnet.github.io/docfx/) и [Sandcastle](https://www.microsoft.com/download/details.aspx?id=10526), и создавать веб-сайты со справочными сведениями по API.

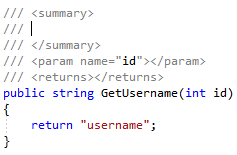
**Примечание**

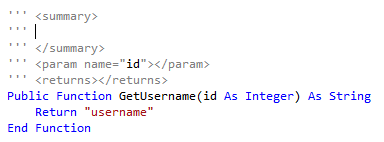
Команда **Вставить комментарий**, которая автоматически вставляет комментарии XML-документации, доступна в [**C#**](https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/xmldoc/) и [**Visual Basic**](https://docs.microsoft.com/ru-ru/dotnet/visual-basic/programming-guide/program-structure/how-to-create-xml-documentation). Однако можно вручную вставить [**комментарии XML-документации в файлы C++**](https://docs.microsoft.com/ru-ru/cpp/build/reference/xml-documentation-visual-cpp) и по-прежнему создавать файлы XML-документации во время компиляции.

**Вставка XML-комментариев для элемента кода**

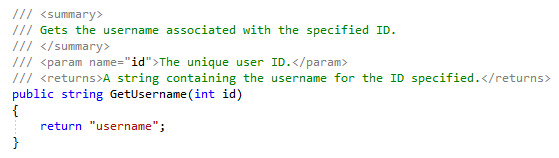
1. Поместите текстовый курсор над элементом, например методом, который нужно задокументировать.
2. Выполните одно из следующих действий.
   * Введите /// в C# или ''' в Visual Basic
   * В меню **Правка** выберите **IntelliSense** > **Вставить комментарий**.
   * Щелкните правой кнопкой мыши или вызовите контекстное меню в редакторе кода или над ним и выберите **Фрагмент кода** > **Вставить комментарий**.

Над элементом кода сразу же создается XML-шаблон. Например, при комментировании метода создается элемент **<summary>** , элемент **<param>** для каждого параметра и элемент **<returns>** для документирования возвращаемого значения.

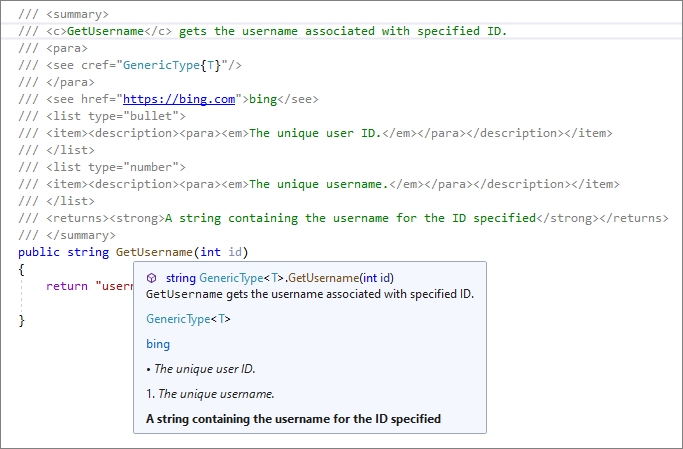




1. Введите описание для каждого XML-элемента, чтобы полностью задокументировать элемент кода.



Стили можно использовать в комментариях XML, которые будут отображаться в виде кратких сведений при наведении указателя мыши на элемент. К этим стилям относятся курсив, полужирный шрифт, маркеры и ссылка, доступная для щелчка.



**Примечание**

Существует [**параметр**](https://docs.microsoft.com/ru-ru/visualstudio/ide/reference/options-text-editor-csharp-advanced?view=vs-2022) для переключения комментариев XML-документации после ввода /// в C# или ''' Visual Basic. В строке меню выберите **Сервис** > **Параметры**, чтобы открыть диалоговое окно **Параметры**. Перейдите в раздел **Текстовый редактор** > **C#** или **Basic** > **Дополнительно**. В разделе **Справка редактора** найдите параметр **Создавать комментарии XML-документации**.

**Комментарии XML-документации**

* Статья
* 09.12.2021
* Чтение занимает 4 мин
  + [](https://github.com/BillWagner)
  + [](https://github.com/olprod)

Были ли сведения на этой странице полезными?

В исходных файлах C# могут находиться структурированные комментарии, используемые для создания документации по API для типов, определенных в этих файлах. Компилятор C# создает файл *XML*, содержащий структурированные данные, представляющие комментарии и сигнатуры API. Другие средства могут обрабатывать эти выходные данные в формате XML и создавать удобочитаемую документацию, например, в виде веб-страниц или PDF-файлов.

Благодаря этому процессу добавление документации по API в код предоставляет множество преимуществ:

* Компилятор C# объединяет структуру кода C# и текст комментариев в XML-документ.
* Компилятор C# проверяет, соответствуют ли комментарии сигнатурам API для соответствующих тегов.
* Средства, обрабатывающие XML-файлы документации, могут определять XML-элементы и атрибуты, характерные для этих средств.

Такие средства как Visual Studio предоставляют IntelliSense для многих распространенных XML-элементов, используемых в комментариях к документации.

В этой статье рассматриваются следующие темы:

* Комментарии к документации и создание XML-файла
* Теги, проверяемые компилятором C# и Visual Studio
* Формат созданного XML-файла

**Создание выходных данных XML-документации**

Вы создаете документацию для кода, указывая ее в специальных полях комментариев, обозначаемых тройными косыми чертами. Поля комментария включают XML-элементы, которые описывают блок кода, следующий за комментариями. Пример:

C#Копировать

/// <summary>

/// This class performs an important function.

/// </summary>

public class MyClass {}

Вы задаете параметр [**DocumentationFile**](https://docs.microsoft.com/ru-ru/dotnet/core/project-sdk/msbuild-props#generatedocumentationfile) или [**DocumentationFile**](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/compiler-options/output#documentationfile), а компилятор находит все поля комментариев с XML-тегами в исходном коде и создает XML-файл документации на основе этих комментариев. Если этот параметр включен, компилятор создаст предупреждение [CS1591](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/compiler-messages/cs1591) для любого открытого видимого члена, объявленного в проекте без комментариев XML-документации.

**Формат комментариев XML**

Для создания комментариев XML-документации необходимо использовать разделители, указывающие начало и конец комментария. С тегами в XML-документации можно использовать следующие разделители:

* Однострочный разделитель ///: в примерах документации и шаблонах проектов C# используется эта форма. Если после разделителя указан пробел, он не включается в выходные данные XML.

**Примечание**

Visual Studio автоматически вставляет теги <summary> и </summary> и устанавливает курсор между ними после ввода разделителя /// в редакторе кода. Вы можете включить или отключить эту функцию в [**диалоговом окне "Параметры"**](https://docs.microsoft.com/ru-ru/visualstudio/ide/reference/options-text-editor-csharp-advanced).

* Многострочные разделители /\*\* \*/: разделители /\*\* \*/ имеют следующие правила форматирования:
  + Строка, содержащая разделитель /\*\*, не обрабатывается как комментарий, если оставшаяся ее часть представляет собой пробелы. Если первый знак после разделителя /\*\* является пробелом, то этот пробел игнорируется, а оставшаяся часть строки обрабатывается. В противном случае весь текст, расположенный в строке после разделителя /\*\*, обрабатывается как часть комментария.
  + Строка, содержащая разделитель \*/, игнорируется, если перед разделителем \*/ стоят только пробелы. В противном случае текст, расположенный в строке до разделителя \*/, обрабатывается как часть комментария.
  + В начале каждой из строк, расположенных после строки, которая начинается с разделителя /\*\*, компилятор выполняет поиск общего шаблона. Шаблон может включать необязательный пробел и знак звездочки (\*), после которого следуют необязательные пробелы. Если компилятор находит общий шаблон в начале каждой строки, которая не начинается с разделителя /\*\* или не заканчивается разделителем \*/, он игнорирует этот шаблон для каждой строки.
  + В следующем комментарии обрабатывается только строка, которая начинается с <summary>. Формат с тремя тегами позволяет создать точно такие же комментарии.

C#Копировать

/\*\* <summary>text</summary> \*/

/\*\*

<summary>text</summary>

\*/

/\*\*

\* <summary>text</summary>

\*/

* + Компилятор обнаруживает в начале второй и третьей строки общий шаблон " \* ". Шаблон не включается в выходные данные.

C#Копировать

/\*\*

\* <summary>

\* text </summary>\*/

* + Компилятор не обнаруживает общий шаблон в следующем комментарии, так как второй знак в третьей строке не является звездочкой. Весь текст во второй и третьей строках обрабатывается как часть комментария.

C#Копировать

/\*\*

\* <summary>

text </summary>

\*/

* + Компилятор не обнаруживает шаблон в следующем комментарии по двум причинам. Во-первых, количество пробелов перед звездочкой не является одинаковым в разных строках. Во-вторых, пятая строка начинается с символа табуляции, который не является пробелом. Весь текст из строк со второй по пятую обрабатывается как часть комментария.

C#Копировать

/\*\*

\* <summary>

\* text

\* text2

\* </summary>

\*/

Для ссылки на XML-элементы (например, если функция обрабатывает определенные XML-элементы, которые требуется включить в комментарии XML-документации) можно использовать стандартный механизм заключения в скобки (&lt; и &gt;). Для ссылки на универсальные идентификаторы в элементах ссылок кода (cref) можно использовать escape-символы (например, cref="List&lt;T&gt;") или фигурные скобки (cref="List{T}"). В особом случае компилятор анализирует фигурные скобки, как угловые, чтобы при ссылке на универсальные идентификаторы сделать комментарий документации менее громоздким.

**Примечание**

Комментарии XML-документации не являются метаданными. Они не включаются в скомпилированную сборку, и поэтому не доступны посредством отражения.

**Средства, принимающие входные данные XML-документации**

Следующие средства создают выходные данные на основе XML-комментариев:

* [DocFX](https://dotnet.github.io/docfx/): *DocFX* — это генератор документации API для .NET, который сейчас поддерживает C#, Visual Basic и F#. Он также позволяет изменять создаваемую справочную документацию. DocFX создает статический веб-сайт HTML на основе исходного кода и файлов Markdown. Кроме того, DocFX предоставляет гибкие возможности по настройке макета и стиля веб-сайта с помощью шаблонов. Можно также создавать пользовательские шаблоны.
* [Sandcastle](https://github.com/EWSoftware/SHFB): *средства Sandcastle* создают файлы справки для библиотек управляемых классов, содержащие как концептуальные страницы справки, так и страницы справки для API. Средства Sandcastle работают из командной строки и не имеют графического интерфейса, функций управления проектами или автоматизированного процесса сборки. Средство Sandcastle Help File Builder предоставляет автономный пользовательский интерфейс и инструменты командной строки для создания файла справки в автоматическом режиме. Также доступен пакет интеграции Visual Studio, позволяющий создавать проекты справки и управлять ими только из Visual Studio.
* [Doxygen](https://github.com/doxygen/doxygen): *Doxygen* создает интерактивный обозреватель документации (в формате HTML) или автономное справочное руководство (в формате LaTeX) на основе набора исходных файлов с комментариями документации. Также поддерживается создание документации в форматах RTF (MS Word), PostScript, PDF с гиперссылками, HTML со сжатием, DocBook и страницы руководства Unix. Doxygen можно настроить для извлечения структуры кода из исходных файлов без комментариев документации.

**Строки идентификаторов**

Каждый тип или член хранится в элементе в выходном XML-файле. Каждый из этих элементов имеет уникальную строку идентификатора, определяющую тип или член. Строка идентификатора должна учитывать операторы, параметры, возвращаемые значения, параметры универсального типа, а также параметры ref, in и out. Чтобы закодировать все эти потенциальные элементы, компилятор следует четко определенным правилам при создании строк идентификаторов. С помощью строки идентификатора программы, обрабатывающие XML-файл, могут определить, к какому элементу метаданных или отражения .NET применяется эта документация.

Компилятор соблюдает следующие правила при формировании строк идентификаторов.

* В строке нет пробелов.
* Первая часть строки определяет тип элемента, использующего один символ, за которым следует двоеточие. Используются следующие типы элементов.

| **ТАБЛИЦА 1** | | |
| --- | --- | --- |
| **Знак** | **Тип члена** | **Примечания** |
| Нет | namespace | Вы не можете присвоить комментарий документации пространству имен, но можете создать для него cref-ссылку, если такие ссылки поддерживаются. |
| T | type | В качестве типа может использоваться класс, интерфейс, структура, перечисление или делегат. |
| C | поле |  |
| С | свойство; | Включает индексаторы или другие индексированные свойства. |
| M | method | Включает специальные методы, такие как конструкторы и операторы. |
| E | event |  |
| ! | текст ошибки | Остальная часть строки предоставляет сведения об ошибке. Компилятор C# создает сведения об ошибках для ссылок, которые не могут быть разрешены. |

* Вторая часть строки содержит полное имя элемента, начиная от корня пространства имен. Имя элемента, включающие типы и пространства имен разделяются точками. Если в имени самого элемента есть точки, они заменяются символами решетки ("#"). Предполагается, что в именах элементов не может содержаться символ решетки. Например, полное имя конструктора для объекта String имеет вид System.String.#ctor.
* Для свойств и методов указывается список параметров, заключенный в круглые скобки. Если параметры не используются, скобки отсутствуют. Параметры разделяются запятыми. Кодирование каждого параметра прямо соответствует тому, как он кодируется в сигнатуре .NET (определения всех элементов в следующем списке, записанных заглавными буквами, см. в разделе [Microsoft.VisualStudio.CorDebugInterop.CorElementType](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.visualstudio.cordebuginterop.corelementtype)):
  + Базовые типы. Обычные типы (ELEMENT\_TYPE\_CLASS или ELEMENT\_TYPE\_VALUETYPE) представляются в виде полного имени типа.
  + Встроенные типы (например, ELEMENT\_TYPE\_I4, ELEMENT\_TYPE\_OBJECT, ELEMENT\_TYPE\_STRING, ELEMENT\_TYPE\_TYPEDBYREF и ELEMENT\_TYPE\_VOID) представляются в виде полного имени соответствующего полного типа. Например, System.Int32 или System.TypedReference.
  + ELEMENT\_TYPE\_PTR представляется в виде "\*" после имени измененного типа.
  + ELEMENT\_TYPE\_BYREF представляется в виде "@" после имени измененного типа.
  + ELEMENT\_TYPE\_CMOD\_OPT представляется в виде "!" и полного имени класса модификатора после имени измененного типа.
  + ELEMENT\_TYPE\_SZARRAY представляется в виде "[]" после типа элемента массива.
  + ELEMENT\_TYPE\_ARRAY представляется как [*нижняя\_граница*:size,*нижняя\_граница*:size], где число запятых на единицу меньше размерности массива, а нижние границы и размер каждого измерения, если они известны, представлены в десятичном формате. Если нижняя граница или размер не указаны, они опускаются. Если нижняя граница и размер для определенной размерности опущены, опускается и символ ":". Например, двухмерный массив со значением 1 для нижних границ без указания размеров обозначается так: "[1:,1:]".
* Только для операторов преобразования (op\_Implicit и op\_Explicit) возвращаемое значение метода кодируется как символ ~, за которым следует тип возвращаемого значения. Например: <member name="M:System.Decimal.op\_Explicit(System.Decimal arg)~System.Int32"> — это тег для оператора приведения public static explicit operator int (decimal value);, объявленного в классе System.Decimal.
* Для универсальных типов в конце имени указывается символ обратного апострофа и число, которое обозначает количество параметров универсального типа. Например: <member name="T:SampleClass``2"> — это тег для типа, который определен как public class SampleClass<T, U>. Для методов, принимающих в качестве параметров универсальные типы, параметры универсального типа указываются в виде чисел, которым предшествуют символы обратного апострофа (например, `0,`1). Каждое число представляет нотацию массива с отсчетом индексации с нуля для параметров универсального типа.
  + ELEMENT\_TYPE\_PINNED представляется в виде "^" после имени измененного типа. Компилятор C# никогда не создает такую кодировку.
  + ELEMENT\_TYPE\_CMOD\_REQ представляется в виде "|" и полного имени класса модификатора после имени измененного типа. Компилятор C# никогда не создает такую кодировку.
  + ELEMENT\_TYPE\_GENERICARRAY представляется в виде "[?]" после типа элемента массива. Компилятор C# никогда не создает такую кодировку.
  + ELEMENT\_TYPE\_FNPTR представляется как "=FUNC:type(*подпись*)", где type обозначает возвращаемый тип, а *подпись* представляет аргументы метода. Если аргументы не используются, скобки опускаются. Компилятор C# никогда не создает такую кодировку.
  + Следующие компоненты сигнатуры не представляются, поскольку они никогда не используются для различения перегруженных методов:
    - соглашение о вызовах;
    - тип возвращаемого значения;
    - ELEMENT\_TYPE\_SENTINEL

Примеры ниже демонстрируют, как создаются строки идентификаторов для класса и его элементов.

C#Копировать

namespace MyNamespace

{

/// <summary>

/// Enter description here for class X.

/// ID string generated is "T:MyNamespace.X".

/// </summary>

public unsafe class MyClass

{

/// <summary>

/// Enter description here for the first constructor.

/// ID string generated is "M:MyNamespace.MyClass.#ctor".

/// </summary>

public MyClass() { }

/// <summary>

/// Enter description here for the second constructor.

/// ID string generated is "M:MyNamespace.MyClass.#ctor(System.Int32)".

/// </summary>

/// <param name="i">Describe parameter.</param>

public MyClass(int i) { }

/// <summary>

/// Enter description here for field message.

/// ID string generated is "F:MyNamespace.MyClass.message".

/// </summary>

public string message;

/// <summary>

/// Enter description for constant PI.

/// ID string generated is "F:MyNamespace.MyClass.PI".

/// </summary>

public const double PI = 3.14;

/// <summary>

/// Enter description for method func.

/// ID string generated is "M:MyNamespace.MyClass.func".

/// </summary>

/// <returns>Describe return value.</returns>

public int func() { return 1; }

/// <summary>

/// Enter description for method someMethod.

/// ID string generated is "M:MyNamespace.MyClass.someMethod(System.String,System.Int32@,System.Void\*)".

/// </summary>

/// <param name="str">Describe parameter.</param>

/// <param name="num">Describe parameter.</param>

/// <param name="ptr">Describe parameter.</param>

/// <returns>Describe return value.</returns>

public int someMethod(string str, ref int nm, void\* ptr) { return 1; }

/// <summary>

/// Enter description for method anotherMethod.

/// ID string generated is "M:MyNamespace.MyClass.anotherMethod(System.Int16[],System.Int32[0:,0:])".

/// </summary>

/// <param name="array1">Describe parameter.</param>

/// <param name="array">Describe parameter.</param>

/// <returns>Describe return value.</returns>

public int anotherMethod(short[] array1, int[,] array) { return 0; }

/// <summary>

/// Enter description for operator.

/// ID string generated is "M:MyNamespace.MyClass.op\_Addition(MyNamespace.MyClass,MyNamespace.MyClass)".

/// </summary>

/// <param name="first">Describe parameter.</param>

/// <param name="second">Describe parameter.</param>

/// <returns>Describe return value.</returns>

public static MyClass operator +(MyClass first, MyClass second) { return first; }

/// <summary>

/// Enter description for property.

/// ID string generated is "P:MyNamespace.MyClass.prop".

/// </summary>

public int prop { get { return 1; } set { } }

/// <summary>

/// Enter description for event.

/// ID string generated is "E:MyNamespace.MyClass.OnHappened".

/// </summary>

public event Del OnHappened;

/// <summary>

/// Enter description for index.

/// ID string generated is "P:MyNamespace.MyClass.Item(System.String)".

/// </summary>

/// <param name="str">Describe parameter.</param>

/// <returns></returns>

public int this[string s] { get { return 1; } }

/// <summary>

/// Enter description for class Nested.

/// ID string generated is "T:MyNamespace.MyClass.Nested".

/// </summary>

public class Nested { }

/// <summary>

/// Enter description for delegate.

/// ID string generated is "T:MyNamespace.MyClass.Del".

/// </summary>

/// <param name="i">Describe parameter.</param>

public delegate void Del(int i);

/// <summary>

/// Enter description for operator.

/// ID string generated is "M:MyNamespace.MyClass.op\_Explicit(MyNamespace.X)~System.Int32".

/// </summary>

/// <param name="myParameter">Describe parameter.</param>

/// <returns>Describe return value.</returns>

public static explicit operator int(MyClass myParameter) { return 1; }

}

}

**Рекомендуемые XML-теги для комментариев к документации по C#**

* Статья
* 07.12.2021
* Чтение занимает 4 мин
  + [](https://github.com/BillWagner)
  + [](https://github.com/olprod)

Были ли сведения на этой странице полезными?

В комментариях к документации по C# используются XML-элементы, что позволяет определить структуру выходной документации. Одним из последствий использования этой функции является то, что в комментарии к документации можно добавить любой допустимый XML-код. Компилятор C# копирует эти элементы в выходной XML-файл. Несмотря на то что в комментариях (включая любой допустимый HTML-элемент) можно использовать любой допустимый XML-код, документирование кода рекомендуется по многим причинам.

Далее приводится ряд рекомендаций, распространенные сценарии использования и вопросы, которые нужно иметь в виду при работе с тегами XML-документации в коде C#. Несмотря на то что теги можно поместить в комментарии к документации, в этой статье описываются рекомендуемые теги для наиболее распространенных конструкций языка. Во всех случаях следует соблюдать следующие рекомендации:

* Для обеспечения согласованности все открытые типы и их члены должны быть задокументированы.
* Закрытые члены можно также задокументировать с помощью XML-комментариев. Однако это приводит к раскрытию внутренних (возможно, конфиденциальных) процессов библиотеки.
* Как минимум, типы и их члены должен иметь тег <summary>, так как его содержимое требуется для IntelliSense.
* Текст документации должны быть написан с использованием законченных предложений, в конце которых ставится точка.
* Разделяемые классы полностью поддерживаются, и данные о документации объединяются в одну запись для этого типа.

Документация XML начинается с ///. При создании проекта шаблоны добавляют несколько строк ///. Обработка этих комментариев имеет некоторые ограничения.

* Документация должна представлять собой XML с правильным форматом. Если XML сформирован неправильно, компилятор выдает предупреждение. Файл документации будет содержать комментарий, в котором сообщается, что обнаружена ошибка.
* Некоторые рекомендуемые теги имеют особые значения.
  + Тег <param> используется для описания параметров. При использовании этого тега компилятор проверяет, что параметр существует и все параметры описаны в документации. При сбое проверки компилятор выдает предупреждение.
  + Атрибут cref может быть присоединен к любому тегу для ссылки на элемент кода. Компилятор проверяет наличие этого элемента кода. При сбое проверки компилятор выдает предупреждение. Компилятор учитывает любые операторы using при поиске типа, описанного в атрибуте cref.
  + Тег <summary> используется технологией IntelliSense в Visual Studio для отображения дополнительных сведений о типе или элементе.

**Примечание**

XML-файл не предоставляет полную информацию о типе и членах (например, он не содержит никаких сведений о типе). Чтобы получить полную информацию о типе или элементе, используйте файл документации вместе с отражением для текущего типа или элемента.

* Разработчики могут создавать собственные наборы тегов. Компилятор скопирует их в выходной файл.

Некоторые из рекомендуемых тегов можно использовать для любого элемента языка. Другие имеют более специализированное использование. Наконец, некоторые теги используются для форматирования текста в документации. В этой статье описываются рекомендуемые теги, упорядоченные по их использованию.

Компилятор проверяет синтаксис элементов, за которыми следует один \* в следующем списке. Visual Studio предоставляет IntelliSense для тегов, проверенных компилятором, и всех тегов, за которыми следует \*\* в следующем списке. Помимо указанных здесь тегов, компилятор и Visual Studio проверяют теги <b>, <i>, <u>, <br/> и <a>. Компилятор также проверяет <tt>, который является устаревшим тегом HTML.

* [Общие теги](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#general-tags), используемые для нескольких элементов. Эти теги являются минимальным набором для любого API.
  + [<summary>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#summary): значение этого элемента отображается в IntelliSense в Visual Studio.
  + [<remarks>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#remarks) \*\*
* [Теги, используемые для членов.](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#document-members) Эти теги используются при документировании методов и свойств.
  + [<returns>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#returns): значение этого элемента отображается в IntelliSense в Visual Studio.
  + [<param>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#param) \*: значение этого элемента отображается в IntelliSense в Visual Studio.
  + [<paramref>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#paramref)
  + [<exception>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#exception) \*
  + [<value>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#value): значение этого элемента отображается в IntelliSense в Visual Studio.
* [Формат вывода документации.](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#format-documentation-output) Эти теги предоставляют направления форматирования для средств, создающих документацию.
  + [<para>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#para)
  + [<list>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#list)
  + [<c>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#c)
  + [<code>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#code)
  + [<example>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#example) \*\*
* [Повторное использование текста документации.](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#reuse-documentation-text) Эти теги предоставляют средства, упрощающие использование комментариев XML.
  + [<inheritdoc>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#inheritdoc) \*\*
  + [<include>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#include) \*
* [Создание ссылок и гиперссылок.](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#generate-links-and-references) Эти теги создают ссылки на другую документацию.
  + [<see>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#see) \*
  + [<seealso>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#seealso) \*
  + [<cref>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#cref-attribute)
  + [<href>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#href-attribute)
* [Теги для универсальных типов и методов](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#generic-types-and-methods). Эти теги используются только для универсальных типов и методов
  + [<typeparam>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#typeparam) \*: значение этого элемента отображается в IntelliSense в Visual Studio.
  + [<typeparamref>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#typeparamref)

**Примечание**

Комментарии документации не применяются к пространству имен.

Чтобы ввести в текст комментария документации угловые скобки, используйте для символов < и > коды HTML &lt; и &gt; соответственно. Это показано в следующем примере.

C#Копировать

/// <summary>

/// This property always returns a value &lt; 1.

/// </summary>

**Общие теги**

**<summary>**

XMLКопировать

<summary>description</summary>

Тег <summary> следует использовать для описания типа или элемента типа. Чтобы добавить дополнительную информацию в описание типа, используйте [<remarks>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#remarks). Чтобы включить средства документации, такие как [DocFX](https://dotnet.github.io/docfx/) и [Sandcastle](https://github.com/EWSoftware/SHFB), для создания внутренних гиперссылок на страницы документации для элементов кода, используйте атрибут [cref](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#cref-attribute). Текст в теге <summary> является единственным источником сведений о типе для технологии IntelliSense и также отображается в окне обозревателя объектов.

**<remarks>**

XMLКопировать

<remarks>

description

</remarks>

Тег <remarks> позволяет добавить сведения о типе или члене типа, дополняющие информацию, указанную с помощью тега [<summary>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#summary). Эти сведения отображаются в окне "Обозреватель объектов". Этот тег может содержать более длинные объяснения. Возможно, вы обнаружите, что использование разделов CDATA для разметки упрощает написание. Такие средства, как [docfx](https://dotnet.github.io/docfx/), обрабатывают текст разметки в разделах CDATA.

**Члены документа**

**<returns>**

XMLКопировать

<returns>description</returns>

Тег <returns> следует использовать в комментариях к объявлению метода для описания возвращаемого значения.

**<param>**

XMLКопировать

<param name="name">description</param>

* name: имя параметра метода. Имя заключается в двойные кавычки (" "). Имена параметров должны соответствовать сигнатуре API. Если один или несколько параметров не охвачены, компилятор выдает предупреждение. Компилятор также выдает предупреждение, если значение name не соответствует формальному параметру в объявлении метода.

Тег <param> следует использовать в комментариях к объявлению метода для описания одного из параметров такого метода. Чтобы задокументировать несколько параметров, используйте несколько тегов <param>. Текст тега <param> отображается в IntelliSense, обозревателе объектов и веб-отчете по комментариям к коду.

**<paramref>**

XMLКопировать

<paramref name="name"/>

* name: имя параметра, на который указывается ссылка. Имя заключается в двойные кавычки (" ").

Тег <paramref> позволяет указать, что слово в комментариях к коду, например в блоке <summary> или <remarks>, ссылается на параметр. В XML-файл такое слово может выделяться особым образом, например курсивом или полужирным шрифтом.

**<exception>**

XMLКопировать

<exception cref="member">description</exception>

* cref = "member": ссылка на исключение, которое доступно из текущей среды компиляции. Компилятор проверяет, существует ли исключение, и приводит member к каноническому имени элемента в выходных XML-данных. member необходимо заключать в двойные кавычки (" ").

Тег <exception> служит для указания возможных исключений. Этот тег может применяться к определениям методов, свойств, событий и индексаторов.

**<value>**

XMLКопировать

<value>property-description</value>

Тег <value> позволяет описывать значение, которое представляется свойством. При добавлении свойства с помощью мастера создания кода из среды разработки Visual Studio .NET для нового свойства будет добавлен тег [<summary>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#summary). Следует вручную добавить тег <value> для описания значения, которое представляется свойством.

**Форматирование выходных данных документации**

**<para>**

XMLКопировать

<remarks>

<para>

This is an introductory paragraph.

</para>

<para>

This paragraph contains more details.

</para>

</remarks>

Тег <para> используется внутри другого тега, например [<summary>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#summary), [<remarks>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#remarks) или [<returns>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#returns), и позволяет добавить к тексту структуру. Тег <para> создает абзац с двойным отступом. Используйте тег <br/>, если хотите добавить в него один абзац.

**<list>**

XMLКопировать

<list type="bullet|number|table">

<listheader>

<term>term</term>

<description>description</description>

</listheader>

<item>

<term>Assembly</term>

<description>The library or executable built from a compilation.</description>

</item>

</list>

Блок <listheader> используется для определения строки заголовка в таблице или списке определений. При определении таблицы необходимо ввести данные для term в заголовке. Каждый элемент в списке указывается в блоке <item>. При создании списка определений необходимо указать одновременно term и description. Тем не менее для таблицы, маркированного или нумерованного списка достаточно ввести только description. Число блоков <item> в списке или таблице не ограничено.

**<c>**

XMLКопировать

<c>text</c>

С помощью тега <c> можно указать, что текст в описании необходимо пометить как код. Чтобы определить несколько строк в качестве кода, используйте тег [<code>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#code).

**<code>**

XMLКопировать

<code>

var index = 5;

index++;

</code>

Тег <code> используется для указания нескольких строк кода. С помощью тега [<c>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#c) можно указать, что однострочный текст в описании необходимо пометить как код.

**<example>**

XMLКопировать

<example>

This shows how to increment an integer.

<code>

var index = 5;

index++;

</code>

</example>

Тег <example> позволяет указать пример использования метода или другого элемента библиотеки. Пример обычно включает использование тега [<code>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#code).

**Повторное использование текста документации**

**<inheritdoc>**

XMLКопировать

<inheritdoc [cref=""] [path=""]/>

Наследование XML-комментариев от базовых классов, интерфейсов и аналогичных методов. Использование inheritdoc позволяет обойтись без копирования и вставки одинаковых XML-комментариев и автоматически поддерживать их синхронизацию. Обратите внимание, что при добавлении тега <inheritdoc> к типу все члены также будут наследовать эти комментарии.

* cref: указывает элемент, из которого следует наследовать документацию. Унаследованные теги не переопределяют уже определенные теги для текущего элемента.
* path: запрос с выражением XPath, в результате которого выводится набор узлов. С помощью этого атрибута можно отфильтровать теги, которые следует включить в наследуемую документацию или исключить из нее.

Добавьте XML-комментарии в базовые классы или интерфейсы, и InheritDoc скопирует их во все реализации классов. Добавьте XML-комментарии в синхронные методы, и InheritDoc скопирует их в асинхронные версии аналогичных методов. Если нужно скопировать комментарии из определенного элемента, укажите нужный элемент в атрибуте cref.

**<include>**

XMLКопировать

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

* filename: имя XML-файла, содержащего документацию. Имя файла может быть квалифицировано с помощью относительного пути к файлу исходного кода. filename необходимо заключать в одинарные кавычки (' ').
* tagpath: путь тегов в filename, который ведет к тегу name. Путь необходимо заключать в одинарные кавычки (' ').
* name: спецификатор имени в теге, предшествующий комментариям. name будет иметь идентификатор id.
* id: идентификатор тега, который предшествует комментариям. Идентификатор заключается в двойные кавычки (" ").

Тег <include> позволяет задать ссылку на комментарии в другом файле, которые описывают типы и элементы вашего исходного кода. Включение внешнего файла является альтернативой размещению комментариев документации непосредственно в файле исходного кода. Помещая комментарии документации в отдельный файл, вы можете реализовать управление их версиями отдельно от версий исходного кода. В этом случае файл исходного кода может быть извлечен для изменения одним пользователем, а файл документации — другим. Тег <include> использует XML-синтаксис XPath. Сведения об использовании тега <include> см. в документации по XPath.

**Создание ссылок и гиперссылок**

**<see>**

C#Копировать

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

// or

/// <see cref="member">Link text</see>

// or

/// <see href="link">Link Text</see>

// or

/// <see langword="keyword"/>

* cref="member": ссылка на член или поле, которые доступны для вызова из текущей среды компиляции. Компилятор проверяет, существует ли элемент кода, и передает member в имя элемента в выходных XML-данных. *member* необходимо заключать в двойные кавычки (" "). Можно указать другой текст ссылки для "cref", используя отдельный закрывающий тег.
* href="link": гиперссылка на заданный URL-адрес. Например, <see href="https://github.com">GitHub</see> формирует гиперссылку с текстом GitHub, которая ведет на сайт https://github.com.
* langword="keyword": ключевое слово языка, например true.

Тег <see> позволяет задать ссылку из текста. С помощью тега [<seealso>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#seealso) можно указать, что текст должен быть размещен в разделе "См. также". Используйте [атрибут cref](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#cref-attribute) для создания внутренних гиперссылок на страницы документации для элементов кода. Вы включаете параметры типа, чтобы указать ссылку на универсальный тип или метод, например cref="IDictionary{T, U}". Кроме того, href является допустимым атрибутом, который будет функционировать как гиперссылка.

**<seealso>**

C#Копировать

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

// or

/// <seealso href="link">Link Text</seealso>

* cref="member": ссылка на член или поле, которые доступны для вызова из текущей среды компиляции. Компилятор проверяет, существует ли элемент кода, и передает member в имя элемента в выходных XML-данных. member необходимо заключать в двойные кавычки (" ").
* href="link": гиперссылка на заданный URL-адрес. Например, <seealso href="https://github.com">GitHub</seealso> формирует гиперссылку с текстом GitHub, которая ведет на сайт https://github.com.

С помощью тега <seealso> можно указать текст, который должен отображаться в разделе **См. также**. Тег [<see>](https://docs.microsoft.com/ru-ru/dotnet/csharp/language-reference/xmldoc/recommended-tags#see) позволяет задать ссылку из текста. Тег seealso нельзя вложить в тег summary.

**Атрибут cref**

Атрибут cref в теге XML-документации означает "кодовая ссылка". Он указывает, что текст внутри тега представляет собой элемент кода, например тип, метод или свойство. Средства создания документации, такие как [DocFX](https://dotnet.github.io/docfx/) и [Sandcastle](https://github.com/EWSoftware/SHFB), используют атрибуты cref для автоматического создания гиперссылок на страницу, где документирован тип или член.

**Атрибут href**

Атрибут href означает ссылку на веб-страницу. Его можно использовать для прямой ссылки на интерактивную документацию по API или библиотеке.

**Универсальные типы и методы**

**<typeparam>**

XMLКопировать

<typeparam name="TResult">The type returned from this method</typeparam>

* TResult: имя параметра типа. Имя заключается в двойные кавычки (" ").

Тег <typeparam> следует использовать в комментариях к объявлению универсального типа или метода для описания параметра типа. Добавьте такой тег для каждого параметра типа универсального типа или метода. Текст для тега <typeparam> будет отображаться в IntelliSense.

**<typeparamref>**

XMLКопировать

<typeparamref name="TKey"/>

* TKey: имя параметра типа. Имя заключается в двойные кавычки (" ").

Используйте этот тег, чтобы разрешить получателям файла документации форматировать слово определенным образом, например курсивным шрифтом.

**Пользовательские теги**

Все описанные выше теги распознаются компилятором C#. Тем не менее пользователь может определять собственные теги. Инструменты, такие как Sandcastle, обеспечивают поддержку дополнительных тегов (например, [<event>](https://ewsoftware.github.io/XMLCommentsGuide/html/81bf7ad3-45dc-452f-90d5-87ce2494a182.htm) и [<note>](https://ewsoftware.github.io/XMLCommentsGuide/html/4302a60f-e4f4-4b8d-a451-5f453c4ebd46.htm)) и даже поддержку [пространств имен документирования](https://ewsoftware.github.io/XMLCommentsGuide/html/BD91FAD4-188D-4697-A654-7C07FD47EF31.htm). Средства создания пользовательской или внутренней документации также можно использовать со стандартными тегами. Кроме того, поддерживается несколько выходных форматов — от HTML до PDF.