Комментарии

Многострочные комментарии

В языке C# при составлении программы используются символы комментариев:
/* - начало комментария
***/ - конец комментария**
Вся последовательность символов, заключенная между символами начала и конца комментария, является комментарием. Такие комментарии могут занимать несколько строк.

```
int i; /* переменная i
будет использоваться как параметр цикла */
```

Многострочные комментарии удобно использовать при отладке, когда нужно временно не выполнять часть кода программы с целью найти ошибку.

Однострочные комментарии

Язык C# также поддерживает написание коротких (однострочных) комментариев. Для этого используются символы //. В этом случае комментарием является все, что расположено после символов // и до конца строки. Текст следующей строки считается кодом программы.

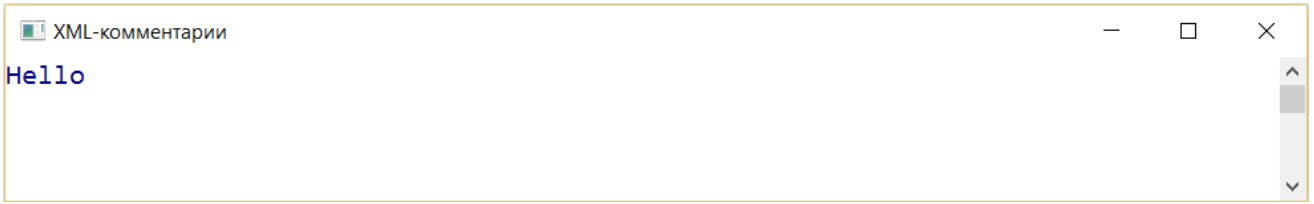
```
int i; // параметр внешнего цикла
int j; // параметр внутреннего цикла
```

XML-комментарии

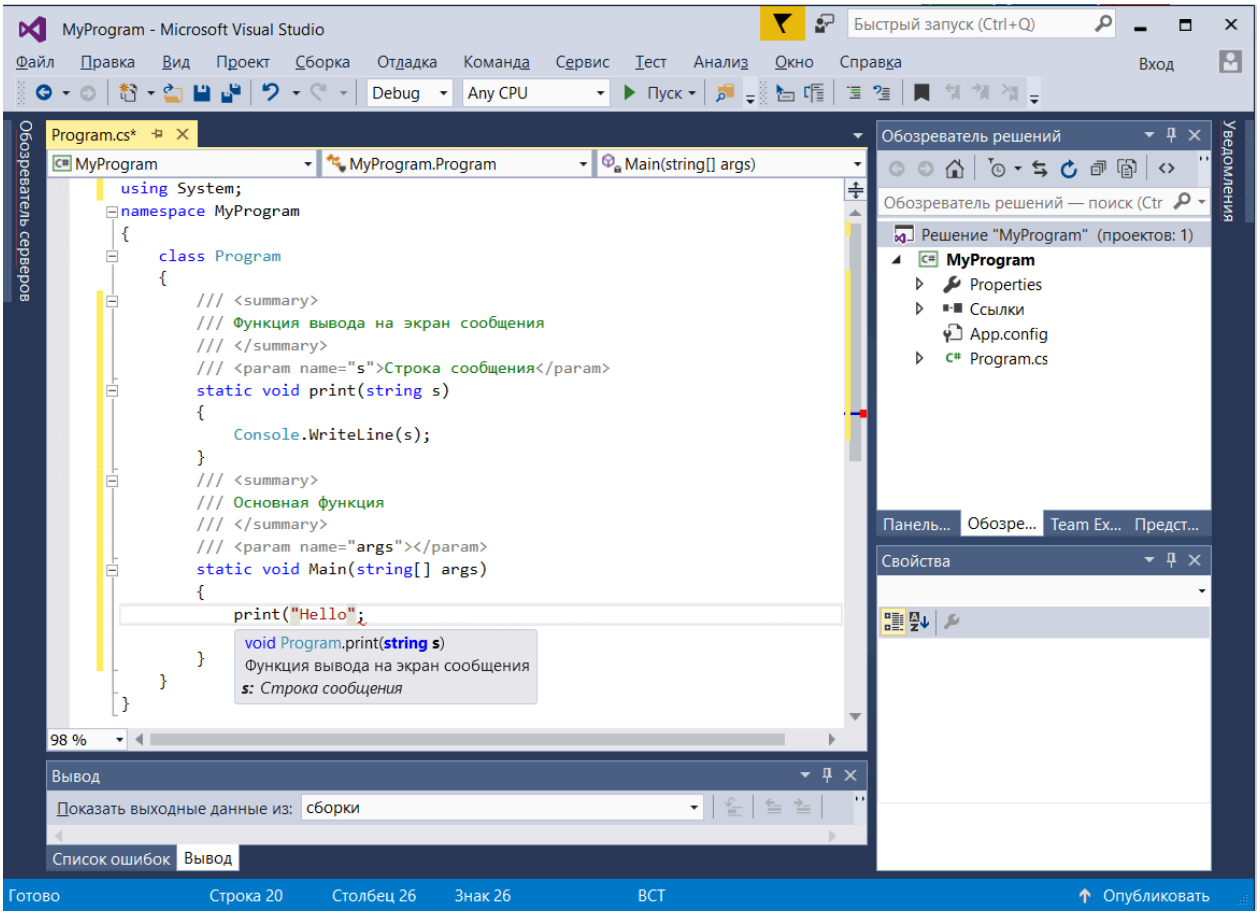
Среда разработки Microsoft Visual Studio позволяет создавать документацию для кода программ путем включения XML-элементов в специальные поля комментариев. Такие комментарии начинаются с тройного символа слеша /// и располагаются непосредственно перед блоком кода, к которому они относятся. Каждая новая строка XML-комментариев начинается с символов ///.

```
using System;
namespace MyProgram
{
  class Program
  {
    /// <summary>
```

```
    /// Функция вывода на экран сообщения
    /// </summary>
    /// <param name="s">Строка сообщения</param>
    static void print(string s)
    {
      Console.WriteLine(s);
    }
    /// <summary>
    /// Основная функция
    /// </summary>
    /// <param name="args"></param>
    static void Main(string[] args)
    {
      Console.Title = "XML-комментарии";
      print("Hello");
      Console.ReadKey();
    }
  }
}
```



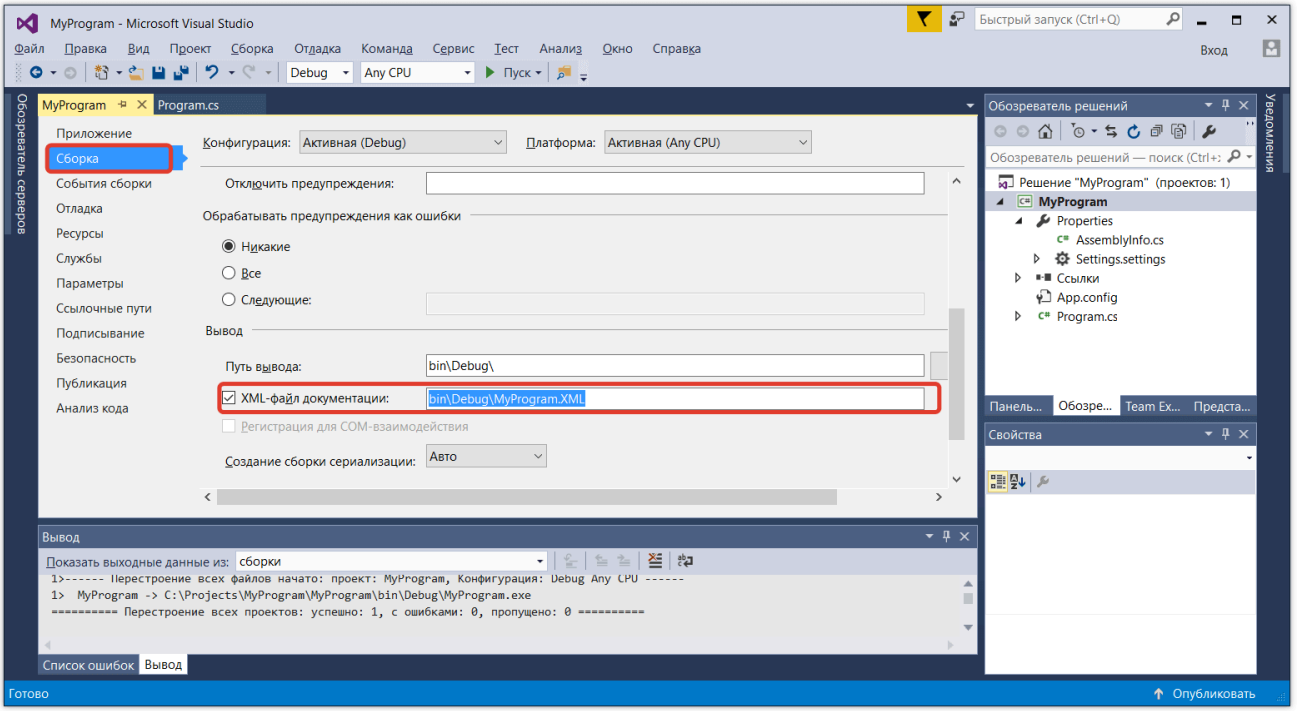
При этом при использовании функции print() автоматически высвечивается описание этой функции и ее аргументов.



При составлении XML-комментариев можно использовать следующие XML-теги.

- <c> - содержит текст, который нужно представить как код.
- <code> - помечает несколько строк как код.
- <example> - содержит описание примера кода, может содержать вложенный тег <code>.
- <exception> - служит для указания исключений, содержит параметр <exception cref="member">Описание исключения</exception>, который указывает ссылку на исключение.
- <include> - используется для ссылок на комментарии в других файлах <include file='filename' path='tagpath[@name="id"]' />
  - filename - имя файла XML, содержащего документацию, заключенное в одинарные кавычки (' ');
  - path - путь тегов в filename, заключенный в одинарные кавычки (' '), который приводит к тегу name;
  - name - спецификатор имени в теге, который предшествует комментариям; name будет иметь id;
  - id - идентификатор для тега, который предшествует комментариям, заключенный в двойные кавычки (" ").
- <list> - вставляет список в документацию.
- <para> - используется внутри тегов <summary>, <remarks>, <returns> и позволяет структурировать текст.

- <param> - используется в комментариях объявления метода для описания его параметров со следующим синтаксисом:
  <param name="Имя">Описание</param>
  name - имя параметра
- <paramref> - позволяет указать, что слово в комментариях кода, например в блоке <summary> или <remarks>, ссылается на параметр:
  <paramref name="Имя"/>
- <permission> - позволяет документу получить доступ к члену
  <permission cref="Ссылка">Описание</permission>
- <remarks> - используется для добавления сведений о типе, дополняющих сведения, указанные в <summary>
  <remarks>Описание</remarks>
- <returns> - используется в комментариях объявления метода для описания возвращаемого значения:
  <returns>Описание</returns>
- <see> - позволяет указать ссылку из текста:
  <see cref="Ссылка"/>
- <seealso> - позволяет указать текст, который будет отображаться в разделе "См. также":
  <seealso cref="Ссылка"/>
- <summary> - используется для описания назначения блока, которому он предшествует:
  <summary>Описание</summary>
- <typeparam> - используется в комментарии объявления универсального типа или метода для описания параметра типа:
  <typeparam name="Имя">Описание</typeparam>
- <typeparamref> - содержит дополнительные сведения параметрах типа в универсальных типах и методах, использование этого тега позволит пользователям файла документации придать слову определенный формат, например выделить его курсивом:
  <typeparamref name="Имя"/>
- <value> - позволяет описывать представляемое свойством значение:
  <value>Описание</value>
  Чтобы обработать и сохранить комментарии документации в файл, при компиляции необходимо использовать параметр /doc.
  Для этого переходим по правой кнопке мыши в меню **Свойства проекта ⇒ Сборка** и устанавливаем галочку напротив **XML-файл документации**.



При этом в указанной папке генерируется файл XML-документации примерно такого вида:

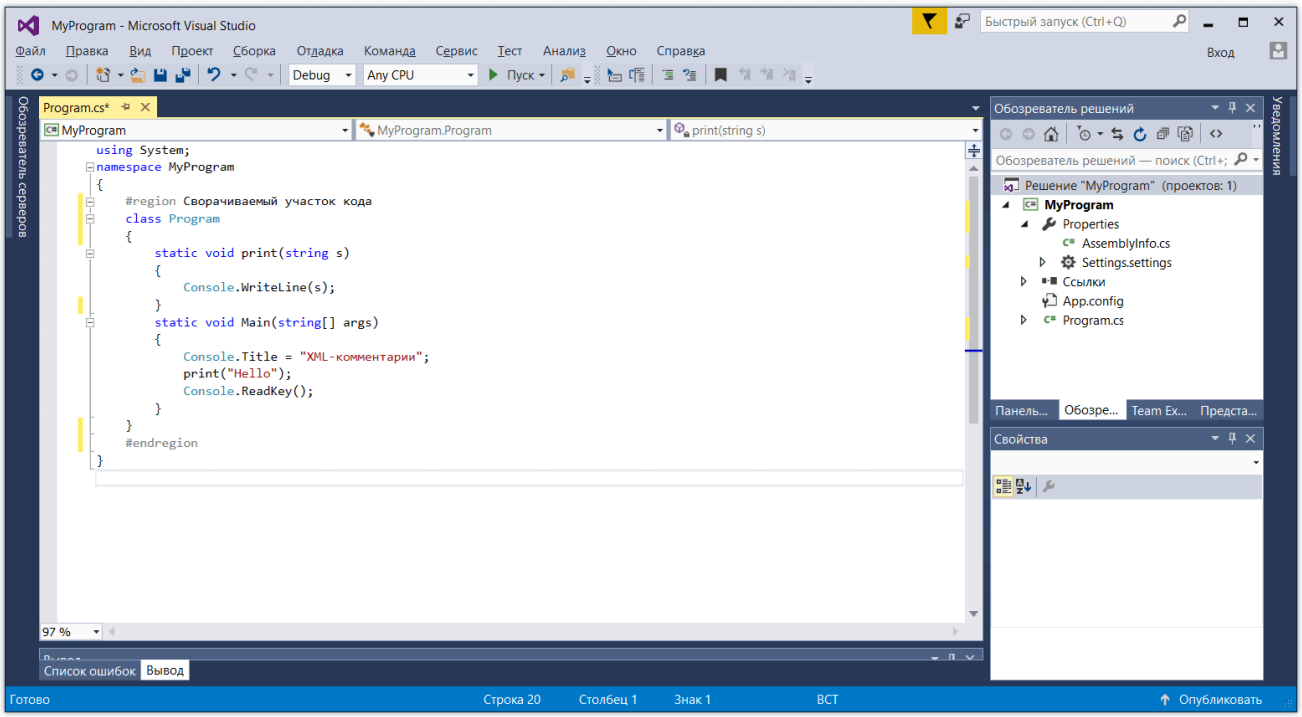
```
<?xml version="1.0"?>
- <doc>
  - <assembly>
      <name>MyProgram</name>
    </assembly>
  - <members>
    - <member name="M:MyProgram.Program.print(System.String)">
        <summary> Функция вывода на экран сообщения </summary>
        <param name="s">Строка сообщения</param>
      </member>
    - <member name="M:MyProgram.Program.Main(System.String[])">
        <summary> Основная функция </summary>
        <param name="args"/>
      </member>
    </members>
</doc>
```

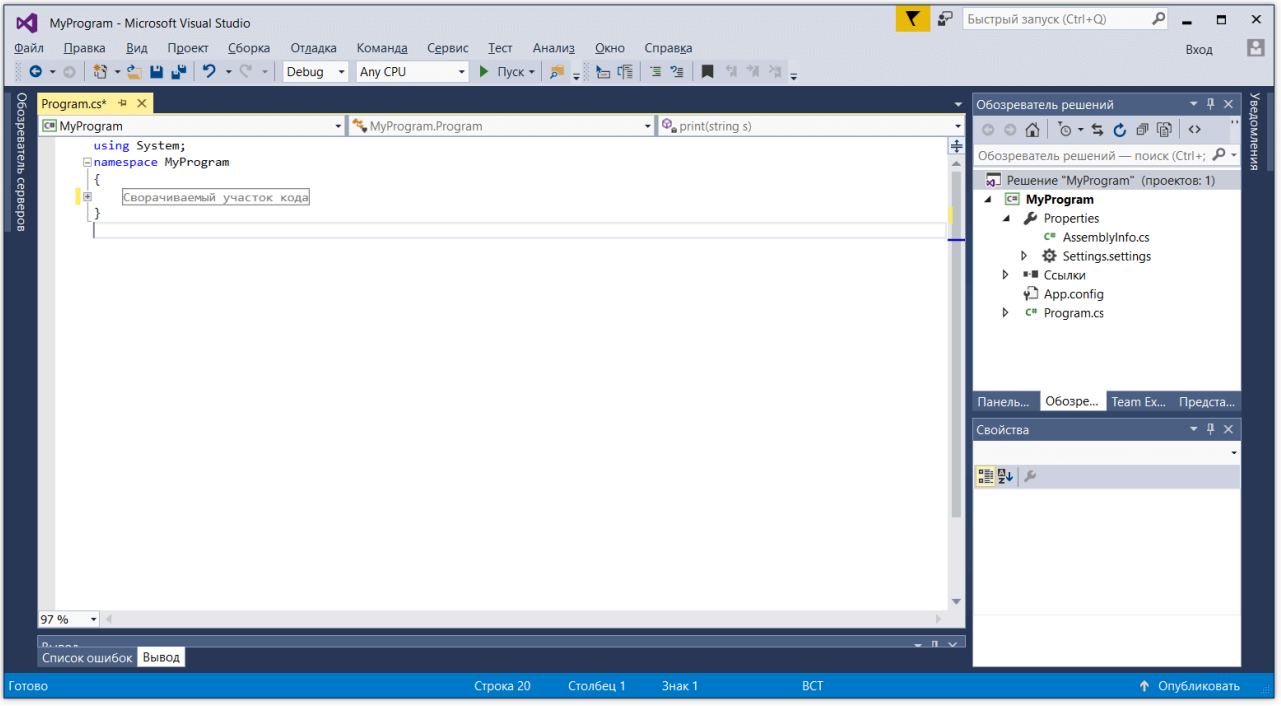
Сворачивание участков кода

Среда разработки Microsoft Visual Studio позволяет сворачивать блоки кода, ограниченные фигурными скобками. Эта возможность применима к описанию структур, классов, функций и методов и т. п.

Однако язык C# предусматривает директиву, позволяющую сворачивать участки кода. Это — директива #region. В конце блока сворачиваемого блока кода, обозначенного директивой #region, должна присутствовать директива #endregion:

```
#region Описание
...
#endregion
```





Создание документированного компонента

Перед разработчиками всегда будет ставиться задача по созданию компонентов для других разработчиков команды или организации.

Прилагаемая к компоненту документация помогает прояснить то, как и когда использовать компонент, и снижает общие затраты на поддержку. Создание документированного компонента включает несколько шагов:

1. Создание компонента
2. Создание XML файла
3. Тестирование компонента в сценарии развертывания

Шаг 1, создание компонента

Поначалу название этого шага может звучать пугающе, примерно как указание "снимите крышу, отложите" в качестве шага 1 в плане реконструкции вашего дома. Однако процесс создания компонента хорошо описан в нескольких разборах программ, приведенных в MSDN Library & VS documentation; поэтому мы не будем на этом останавливаться, а возьмем простой пример и используем его, чтобы показать, как добавить XML документацию.

Окончательной целью этого пошагового руководства является создание двух файлов. Первый - .DLL, которую будет использовать потребитель компонента; второй - .XML файл, содержащий всю документацию для этого компонента. Структура компонента-образца следующая:
Код:

```
using System;

namespace XMLDeploy1
{
 public class Temperature
 {
  public static int CelsiusToFahrenheit(int degreesCelsius)
  {
   return ((int)((9/5)*degreesCelsius) + 32);
  }

  public static int FahrenheitToCelsius(int degressFahrenheit)
  {
   return ((int)((5/9)*(degressFahrenheit - 32)));
  }
 }
}
```

Обратите внимание, что в компоненте пропущена очень важная часть - документация! Несмотря на сравнительную простоту компонента, в нем есть

три момента, где добавленная документация будет чрезвычайно полезна для потребителя компонента. Эти три момента - это документация уровня класса, документация уровня метода и документация уровня параметра.

Чтобы добавить документацию уровня класса, наберите /// в строке над
Код:
```
public class Temperature
```

Затем между автоматически сгенерированными тегами <summary> опишите то, для чего предназначен класс Temperature. Например:
Код:
```
/// <summary>
/// Класс temperature предоставляет функции для преобразования
/// показаний различных шкал температур.
/// </summary>
public class Temperature
```

Документация уровня метода и параметра примыкает к данному методу, так же как только что добавленная нами документация примыкала к классу. Например, давайте добавим комментарии к методу
Код:
```
public static int CelsiusToFahrenheit(int degreesCelsius)
```

Наберите /// перед описанием метода, обратите внимание, что при этом были автоматически сгенерированы и добавлены в исходный код три тега. Первый, <summary>, удобен для описания того, что делает метод. Второй, , используется для описания того, что необходимо передать в 'degreesCelsius'. И третий тег, <returns>, подходит для описания возвращаемого значения метода. Документированный метод выглядит примерно так:
Код:
```
/// <summary>
/// Преобразовывает градусы Цельсия в градусы по Фаренгейту
/// </summary>
/// Degrees Celsius
/// <returns>Возвращает градусы по Фаренгейту</returns>
public static int CelsiusToFahrenheit(int degreesCelsius)
```

После применения этих же шагов для создания документации уровня метода и параметра ко второму методу класса, наш пример выглядит так:
Код:
```
using System;
```

```
namespace XMLDeploy1
{
 /// <summary>
 /// Класс temperature предоставляет функции для преобразования
 /// показаний различных шкал температур.
 /// </summary>
 public class Temperature
 {
  /// <summary>
  /// Преобразовывает градусы Цельсия в градусы по Фаренгейту
  /// </summary>
  /// Degrees Celsius
  /// <returns> Возвращает градусы по Фаренгейту </returns>
  public static int CelsiusToFahrenheit(int degreesCelsius)
  {
   return ((int)((9/5)*degreesCelsius) + 32);
  }

  /// <summary>
```

Шаг 2, создание XML файла

Тег: <c>

Этот тег обеспечивает механизм индикации того, что шрифт фрагмента текста описания должен быть таким же, какой используется в блоке кода. (Для строк кода используйте <code>)

Синтаксис: <c>текст который будет отформатирован так же как и код</c>

Пример:
Код:

```
/// <remarks>Класс <c>Point</c> моделирует координаты точки в 2-х
/// мерном пространстве.</remarks>
public class Point
{
 // ...
}
```

Тег: <code>
Этот тег используется, чтобы применить определенный шрифт к одной или более строкам исходного кода или выходных данных программы. (Для маленьких фрагментов кода в комментариях используйте <c>.)

Синтаксис: <code>исходный код или программный вывод</code>

Пример:
Код:

```
/// <summary>Этот метод изменяет положение точки
/// на  x- и y-смещения.
/// <example>Например:
/// <code>
///       Point p = new Point(3,5);
///       p.Translate(-1,3);
/// </code>
/// результат в  <c>p</c>будем иметь значения (2,8).
/// </example>
/// </summary>
public void Translate(int xor, int yor)
{
  X += xor;
  Y += yor;
}
```

Тег: <example>
Этот тег дает возможность вводить пример кода в комментарий, чтобы определить, как могут использоваться метод или другой член библиотеки. Обычно это также влечет за собой использование тега <code>.

Синтаксис: <example>описание</example>

Пример:
Пример смотрите <code>.

Тег: <exception>
Этот тег обеспечивает возможность документировать исключительные ситуации, которые может формировать метод.

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

cref="member" - Имя члена. Генератор документации проверяет существование данного члена и преобразовывает член в имя канонического элемента в файле документации.

description - Описание обстоятельств, в которых формируется исключительная ситуация.

Пример:

Код:

```
public class DataBaseOperations
{
 /// <exception cref="MasterFileFormatCorruptException">
 /// </exception>
 /// <exception cref="MasterFileLockedOpenException">
 /// </exception>
 public static void ReadRecord(int flag)
 {
  if (flag == 1)
    throw new MasterFileFormatCorruptException();
  else if (flag == 2)
    throw new MasterFileLockedOpenException();
  // ...
 }
}
```

**Тег: <list>
**

Этот тег используется для создания списка или таблицы элементов. Он может содержать блок <list> для определения строки заголовка таблицы или списка определения. (При описании таблицы необходимо предоставить только входные данные для термина.)

Каждый элемент списка задается блоком <i>. При создании списка определения термин и описание должны быть заданы. Однако для таблицы, маркированного списка или нумерованного списка необходимо только, чтобы было задано описание.

Синтаксис: <term>term</term>
<description>description</description>

<term>term</term>
<description>description</description>

...

<term>term</term>
<description>description</description>

term - Определяемый термин.
description - Описание термина.
Или элемент в маркированном списке, или нумерованный список, или определение термина.

Пример:
[/list]
Код:
```
public class MyClass
{
 /// <remarks>Пример маркированного списка:
 /// [li]
 /// [i]
 /// <description>Item 1.</description>
 /// [/i]
 /// [i]
 /// <description>Item 2.</description>
 /// [/i]
 /// [/li]
 /// </remarks>
 public static void Main ()
 {
   // ...
 }
}
```

Тег:
Этот тег предназначен для использования в других тегах, таких как <remarks> или <returns>, и разрешает добавление структуры в текст.

Синтаксис: content

content - Текст абзаца.

Пример:
Код:
```
/// <summary>This is the entry point of the Point class
/// testing program.
/// This program tests each method and operator,
/// and is intended to be run after any non-trvial
/// maintenance has been performed on the Point
/// class.</summary>
public static void Main()
{
 // ...
}
```

Тег:

Этот тег используется для описания параметра метода, конструктора или индексатора.

Синтаксис: description

name - Имя параметра.
description - Описание параметра.

Пример:
Код:
```
/// <summary>This method changes the point's location to
/// the given coordinates.</summary>
/// <c>xor</c> is the new x-coordinate.
/// <c>yor</c> is the new y-coordinate.
public void Move(int xor, int yor)
{
  X = xor;
  Y = yor;
}
```

Тег:
Этот тег используется как признак того, что данное слово является параметром. Файл документации может быть обработан так, чтобы отформатировать этот параметр некоторым определенным способом.

Синтаксис: name - Имя параметра.

Пример:
Код:
```
/// <summary>This constructor initializes the new Point to
///       (,
/// ).</summary>
/// <c>xor</c> is the new Point's x-
/// coordinate.
/// <c>yor</c> is the new Point's y-
/// coordinate.
public Point(int xor, int yor)
{
  X = xor;
  Y = yor;
}
```

Тег:
Этот тег разрешает документирование возможности безопасного доступа к

члену.

Синтаксис: description

cref="member" - Имя члена. Генератор документации проверяет существование данного элемента кода и преобразовывает член в имя канонического элемента в файле документации.
description - Описание доступа к члену.
Пример:
Код:

```
/// Everyone can
/// access this method.
public static void Test()
{
  // ...
}
```

Тег: <remarks>
Этот тег используется для включения обзорной информации о типе. (Для описания типа членов используйте <summary>.)

Синтаксис: <remarks>description</remarks>

description - Текст комментариев.

Пример:
Код:

```
/// <remarks>Class <c>Point</c> models a point in a two-dimensional
/// plane.</remarks>
public class Point
{
  // ...
}
```

Тег: <returns>
Этот тег используется для описания возвращаемого методом значения.

Синтаксис: <returns>description</returns>

description - Описание возвращаемого значения.

Пример:
Код:

```
/// <summary>Report a point's location as a string.</summary>
/// <returns>A string representing a point's location, in the form
/// (x,y), without any leading, training, or embedded
/// whitespace.</returns>
public override string ToString()
{
  return "(" + X + "," + Y + ")";
}
```

Тег: <see>
Этот тег обеспечивает возможность задавать связь в тексте. (Для обозначения текста, который должен появиться в секции See Also, используйте <seealso>.)

Синтаксис: <see cref="member"/>

cref="member" - Имя члена. Генератор документации проверяет существование данного элемента кода и передает член в имя элемента в файле документации.

Пример:
Код:

```
/// <summary>This method changes the point's location to
/// the given coordinates.</summary>
/// <see cref="Translate"/>
public void Move(int xor, int yor)
{
  X = xor;
  Y = yor;
}
/// <summary>This method changes the point's location by
/// the given x- and y-offsets.
/// </summary>
/// <see cref="Move"/>
public void Translate(int xor, int yor)
{
  X += xor;
  Y += yor;
}
```

Тег: <seealso>

Этот тег обеспечивает возможность генерировать элемент для секции See Also. (Используйте <see> для задания связи из текста.)

Синтаксис: <seealso cref="member"/>

cref="member" - Имя члена. Генератор документации проверяет существование данного элемента кода и передает член в имя элемента в файле документации.

Пример:
Код:

```
/// <summary>This method determines whether two Points have the /// same location.
</summary>
/// <seealso cref="operator=="/>
/// <seealso cref="operator!="/>
public override bool Equals(object o)
{
  // ...}
```

Тег: <summary>
Этот тег может использоваться для описания члена для типа. (Используйте <remarks> для описания самого типа.)

Синтаксис: <summary>description</summary>

description - Краткое описание члена.

Пример:
Код:

```
/// <summary>This constructor initializes the new Point to
/// (0,0).</summary>
public Point() : this(0,0)
{
}
```

Тег: <value>
Этот тег обеспечивает возможность описывать свойство.

Синтаксис: <value>property description</value>

property description - Описание свойства.

Пример:
Код:

```
/// <value>Property <c>X</c> represents the point's x-
/// coordinate.</value>
public int X
{
  get { return x; }
  set { x = value; }
}
```