**[Упрощаем отладку с помощью атрибутов](http://andrey.moveax.ru/post/csharp-debugger-attributes)**

9 июля 2013  [Теория и практика](http://andrey.moveax.ru/category/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F-%D0%B8-%D0%BF%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0)  [Комментарии (4)](http://andrey.moveax.ru/post/csharp-debugger-attributes#comment)

Давайте рассмотрим несколько атрибутов, способных облегчить отладку .NET приложений.

**[DebuggerDisplay]**

Этот атрибут позволяет отобразить значения ключевых свойств экземпляра класса вместо типа в окнах Watch. Для этого необходимо добавить его в описание класса, указав имена этих свойств в фигурных скобках в качестве параметра. Например:

[?](http://andrey.moveax.ru/post/csharp-debugger-attributes)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | [DebuggerDisplay("{Date} => {Value}")]  internal class ChartPoint  {      public DateTime Date { get; set; }      public float Value { get; set; }      public string Hint { get; set; }  } |

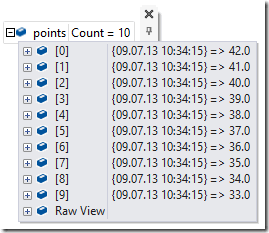
И тогда вместо стандартного

[DebuggerDisplay]

отладчик покажет следующее:

[DebuggerDisplay]

Кроме того, такое решение удобно для перечислений. Например:



**[DebuggerBrowsable]**

Это атрибут, определяющий что будет отображено в окне отладчика. Позволяет указать одно из трех состояний, определенных в перечислении *DebuggerBrowsableState*:

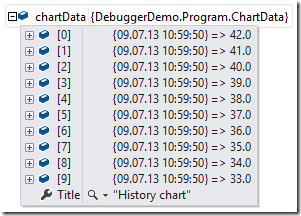
* *Never* – не показывать элемент (полезен при создании библиотечных классов и компонент);
* *Collapsed* – показывать свернутым (поведение по умолчанию);
* *RootHidden* – не показывать сам элемент, но показывать содержащуюся в нем коллекцию.

Посмотрим пример:

[?](http://andrey.moveax.ru/post/csharp-debugger-attributes)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | internal class ChartData  {      [DebuggerBrowsable(DebuggerBrowsableState.Never)]      private int \_chartId;        public string Title { get; set; }        [DebuggerBrowsable(DebuggerBrowsableState.RootHidden)]      public ChartPoint[] Points { get; set; }  } |

В данном случае, поле *\_chartId* не будет отображено в окне отладчика, а точки *Points* будут показаны сразу на уровне экземпляра. Обратите внимание, что *[DebuggerDisplay]*, указанный для класса *ChartPoint*, по-прежнему сразу отображает значения указанных свойств:



**[DebuggerStepThrough]**

*DebuggerStepThroughAttribute* позволяет указать метод, в который отладчик не должен заходить в пошаговом режиме. Однако установленные внутри точки останова сработают (опция отладчика "Just My Code" должна быть выключена).

**[DebuggerHidden]**

Этот атрибут полностью скрывает элемент от отладчика. Использование его со свойством аналогично *[DebuggerBrowsable(DebuggerBrowsableState.Never)]*. При добавлении его к методу, отладчик в пошаговом режиме не только проигнорирует сам метод, но и установленные внутри точки останова.

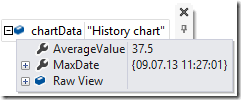
**[DebuggerTypeProxy]**

*DebuggerTypeProxyAttribute* позволяет задать объект, который будет использоваться для отображения данных экземпляра в отладке. Это позволяет кардинально изменить вид объекта в отладчике.

Прокси это обычный класс, но с единственным условием – параметром его конструктора должен быть экземпляр исходного типа. Например, с помощью данной техники отобразим для *ChartData* максимальную дату и среднее значение:

[?](http://andrey.moveax.ru/post/csharp-debugger-attributes)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | internal class ChartDataDebugView  {      private readonly ChartData \_chartData;        public ChartDataDebugView(ChartData chartData)      {          this.\_chartData = chartData;      }        public DateTime MaxDate      {          get { return this.\_chartData.Points.Max(p => p.Date); }      }        public float AverageValue      {          get { return this.\_chartData.Points.Average(p => p.Value); }      }  }    [DebuggerTypeProxy(typeof(ChartDataDebugView))]  [DebuggerDisplay("{Title}")]  internal class ChartData  {      private int \_chartId;      public string Title { get; set; }      public ChartPoint[] Points { get; set; }  } |



Ссылка *Raw View* позволяет посмотреть оригинальный экземпляр. Также обратите внимание, что *[DebuggerDisplay]* в данном случае работает со свойствами исходного типа.

**[DebuggerVisualizer]**

И в завершении, стоит упомянуть еще один атрибут – *[DebuggerVisualizerAttribute](http://msdn.microsoft.com/en-us/library/system.diagnostics.debuggervisualizerattribute.aspx" \o "DebuggerVisualizerAttribute Class" \t "_blank),* который позволяет указать собственный класс для отображения текущих значений. В данном случае весь вывод информации, включая диалоговое окно, полностью ложиться на разработчика.

Для упрощения этой задачи можно использовать абстрактный базовый класс *[DialogDebuggerVisualizer](http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.debuggervisualizers.dialogdebuggervisualizer(v=vs.110).aspx" \o "DialogDebuggerVisualizer Class" \t "_blank)*, реализующий окно вывода. В этом случае, необходимо переопределить его метод *Show()* для отображения информации. Простейшую реализацию такого подхода можно [посмотреть в MSDN](http://msdn.microsoft.com/en-us/library/microsoft.visualstudio.debuggervisualizers.dialogdebuggervisualizer.show.aspx).