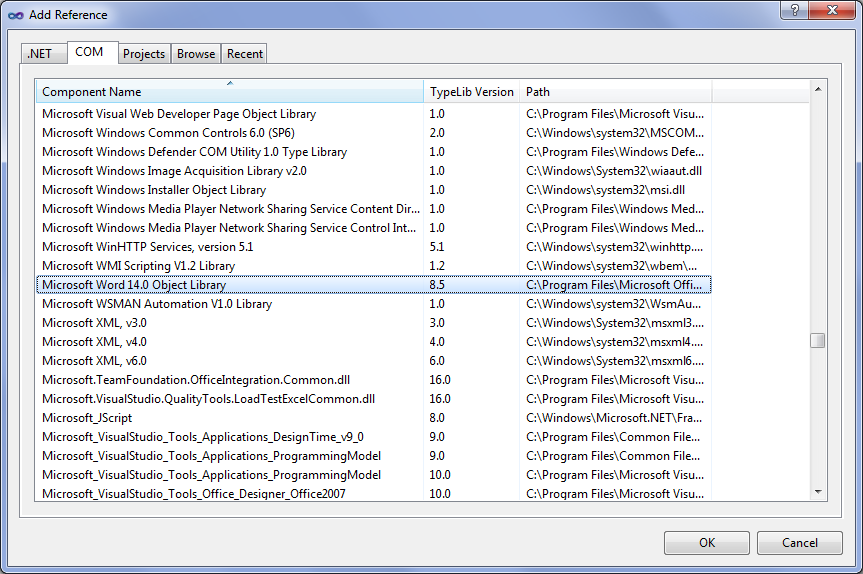
Обработка событий от COM в .NET с помощью IConnectionPoint.

Работа с COM из .NET организована достаточно просто и обычно не вызывает сложностей. Нам просто нужно щелкнуть по Add Reference на проекте, перейти во вкладку COM и выбрать необходимую библиотеку. Visual Studio автоматически сгенерирует для нас управляемую обертку вокруг COM библиотеки (см. Primary Interop Assembly). С помощью этой обертки мы легко можем взаимодействовать с COM, вызывая его методы и подписываясь на события с помощью стандартных .NET делегатов.

Итак, в моем примере я решил написать простое консольное приложение которое отслеживает все открываемые документы Microsoft Word. Для этого я добавлю ссылку на Microsoft Word Object Library.



А затем напишу простой и понятный код, в котором сначала я создам объект Microsoft.Office.Interop.Word.Application, а затем подпишусь на событие DocumentOpen.

using System;

using Microsoft.Office.Interop.Word;

using WordApp = Microsoft.Office.Interop.Word.Application;

namespace UsingWordByCom

{

public class Program

{

public static void Main(string[] args)

{

Console.WriteLine("Monitoring all opened documents by Microsoft Word.");

Console.WriteLine("(Using .NET delegates)");

Console.WriteLine();

// Create Microsoft.Office.Interop.Word.Application:

var wordApp = new WordApp();

// subscribe to DocumentOpen event:

wordApp.DocumentOpen += new ApplicationEvents4\_DocumentOpenEventHandler(wordApp\_DocumentOpen);

Console.WriteLine("Press <Enter> for exit.");

Console.WriteLine();

Console.ReadKey();

// unsubscribe:

wordApp.DocumentOpen -= wordApp\_DocumentOpen;

}

/// <summary>

/// DocumentOpen event handler

/// </summary>

/// <param name="doc">opened document</param>

private static void wordApp\_DocumentOpen(Document doc)

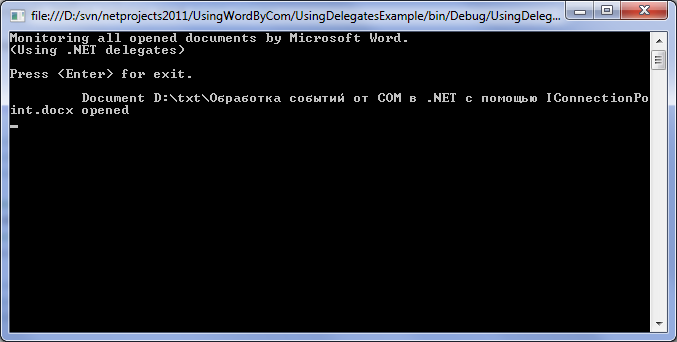
{

Console.WriteLine("\t Document {0} opened", doc.FullName);

}

}

}



Это самый распространённый способ работы с COM из .NET и в большинстве случаев он удовлетворяет нужды разработчика, которому нужно просто вызвать несколько методов или подписаться на событие. Однако использование делегатов для подписки на события от COM влечет за собой некоторые издержки. При каждой подписке на событие создается отдельный класс приемника событий (Event Sink). Таким образом, если мы подпишемся на десяток событий, то у нас будут созданы десять разных приемников. Более того при вызове даже одного события COM, .NET все равно осуществит 10 вызовов на делегаты по умолчанию (см. http://support.microsoft.com/kb/811645).

Альтернативным способом является использование классической модели COM и connection point для подписки на события. Для этого нам необходимо явно завести свой собственный класс приемника событий (Event Sink), создать объект этого класса и передать ссылку на него в COM. Ниже приведен код демонстрирующий работу с connection point.

using System;

using System.Runtime.InteropServices;

using System.Runtime.InteropServices.ComTypes;

using Microsoft.Office.Interop.Word;

using WordApp = Microsoft.Office.Interop.Word.Application;

namespace UsingConnectionPointExample

{

internal class Program

{

[ComVisible(true)]

[ClassInterface(ClassInterfaceType.None)]

public class WordAppEventsSink : ApplicationEvents4

{

/// <summary>

/// DocumentOpen event handler

/// </summary>

/// <param name="doc">opened document</param>

public void DocumentOpen(Document doc)

{

Console.WriteLine("\t Document {0} opened", doc.FullName);

}

#region All another events sink methods

public void DocumentBeforeClose(Document Doc, ref bool Cancel)

{

// do nothing

}

. . .

#endregion

}

private static void Main(string[] args)

{

Console.WriteLine("Monitoring all opened documents by Microsoft Word.");

Console.WriteLine("(Using classic COM connection point)");

Console.WriteLine();

// Create Microsoft.Office.Interop.Word.Application:

var wordApp = new WordApp();

var sink = new WordAppEventsSink(); // create event sink object

var connectionPointContainer = (IConnectionPointContainer)wordApp;

Guid guid = typeof(ApplicationEvents4).GUID; // got guid of Microsoft Word Object Library

// getting connection point object:

IConnectionPoint connectionPoint;

connectionPointContainer.FindConnectionPoint(ref guid, out connectionPoint);

// subscribe to all events through our WordAppEventsSink object:

int cookie;

connectionPoint.Advise(sink, out cookie);

// now we listening...

Console.WriteLine("Press <Enter> for exit.");

Console.WriteLine();

Console.ReadKey();

//after all we use cookie for unsubscribe

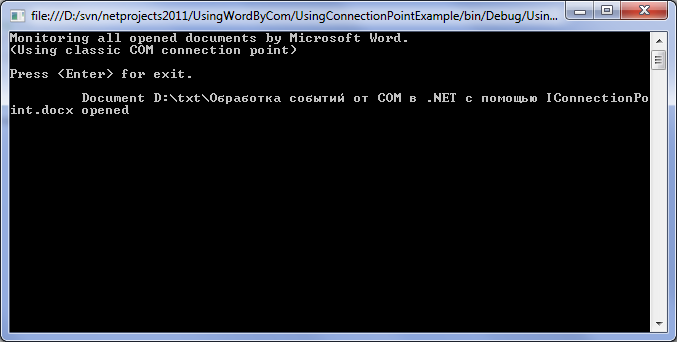
connectionPoint.Unadvise(cookie);

}

}

}

Обратите внимание что класс WordAppEventsSink унаследован от интерфейса ApplicationEvents4 из пространства имен Microsoft.Office.Interop.Word. Реализовав этот интерфейс мы «подписываемся» сразу на все события от COM-библиотеки. Использование connection point позволяет уйти от создания избыточных временных приемников событий таким образом увеличив производительность. Плюс эта модель более гибкая и может быть предпочтительнее в некоторых сценариях взаимодействия. По традиции за гибкость решения мы расплачиваемся читабельностью кода.



Дополнительные материалы по теме:

<http://blogs.msdn.com/b/rcook/archive/2012/05/09/9023741.aspx>

<http://support.microsoft.com/kb/811645>

<http://www.codeproject.com/Articles/990/Understanding-Classic-COM-Interoperability-With-NE>