ГЛАВА 7

Расширение IDE Visual Studio

В этой главе:

- Расширение при помощи макросов
- Встраиваемые модули Visual Studio

По мере того как пакет Microsoft Visual Studio с годами прогрессировал, разработчики проводили в окне этого приложения все большую часть своего рабочего времени. В эпоху Microsoft Visual C++ 6.0 многие из нас пользовались лишь базовыми возможностями редактирования, построения и отладки. Для отслеживания ошибок, создания блок-схем и сложного редактирования мы применяли другие инструменты. С появлением Visual Studio 2005 стало возможным с пользой проводить в одном-единственном приложении весь рабочий день. Здесь есть все: начиная от систем управления версиями и отслеживания ошибок Visual Studio Team System и заканчивая конструктором классов и веб-развертыванием. У нас даже есть веб-обозреватель, позволяющий нам проводить «исследования», все так же создавая впечатление продуктивной работы. По моему мнению, для того чтобы вообще не выходить из него, в пакете Visual Studio не хватает всего трех компонентов: интеграции электронной почты, интеграции обмена короткими сообщениями через Интернет и интеграции игры Halo. Если когда-нибудь эти три дополнительных модуля будут написаны, вы вполне можете забыть, как нажимать клавишное сочетание Alt+Tab.

Без сомнения, история расширения Visual Studio практически бесконечна. К сожалению, очень немногие разработчики осознают, какая мощь находится у них под руками. Бессчетное количество раз я слышал, как разработчики говорили, что если бы Visual Studio умела делать *нечто*, то это была бы величайшая среда для разработки на планете. Практически всегда возможность, о которой они просили, было относительно просто реализовать, учитывая расширяемость пакета. Многие разработчики жалуются, что пользователи совершенно не хотят тратить время на то, чтобы хоть что-то узнать о компьютерах. Я всегда парирую: «Не могу поверить, как много разработчиков практически ничего не знают о своих средах для разработки». Разработчикам не нужно знать все ключи реестра, используемые Visual Studio, но они должны знать способы расширения среды, чтобы уметь решать проблемы и добавлять отсутствующие возможности, необходимые в их лаборатории по разработке программного обеспечения.

Функциональность расширяемости строится на четырех элементах: макросы, встраиваемые модули, мастера и партнерская программа Visual Studio Industry Partner (VSIP). Visual Studio позволяет создавать макросы при помощи встроен-

ного редактора Visual Studio Macros IDE. Этот редактор выглядит и работает так же, как среда Visual Studio, поэтому, как только вы начинаете писать свои макросы, все ваши усилия по изучению среды окупаются. Среда носит название Visual Studio for Applications (VSA) и ее можно интегрировать в ваши продукты.

Единственное ограничение VSA заключается в том, что макросы можно создавать только на языке Microsoft Visual Basic. Так как предполагается, что Microsoft .NET не должна ничего знать о языке, я не понимаю, почему Microsoft настолько ограничила среду и не реализовала в ней поддержку языка C#. Фактически, это ограничение означает, что Microsoft совершенно неважно, что вы выбрали для себя в качестве основного языка C#, — возможно, потому что отличаетесь пылкой любовью к точкам с запятой, — и вам все равно придется изучить Visual Basic, чтобы писать макросы.

Второй вариант — использовать встраиваемые модули и мастера. Макросы отлично подходят для небольших и не связанных с пользовательским интерфейсом задач, а встраиваемые модули — это компоненты COM, позволяющие писать настоящие расширения IDE. Например, можно создавать собственные окна инструментов, добавлять страницы свойств в диалоговое окно Options (Параметры) и реагировать на команды меню из встраиваемых модулей. Любой, кому вы предоставляете свой макрос, может увидеть его исходный код, но встраиваемые модули распространяются в двоичной форме, и для написания их можно использовать любой язык, поддерживающий COM.

Мастера наиболее полезны для задач, в которых пользователя необходимо провести через последовательность шагов, обязательных для выполнения задачи. Превосходный пример — мастер Smart Device Application, который проводит вас через процесс создания приложения Smart Device («умное устройство»). Однако с появлением шаблонов Visual Studio Template необходимость в мастерах значительно сократилась.

Последний вариант расширения — это пакеты Visual Studio Industry Partner (VSIP). Практически любые идеи, которые у вас могут возникнуть относительно расширения Visual Studio, можно реализовать в форме макроса или встраиваемого модуля. Однако если вам необходим совершенно новый тип проекта, редактор, обрабатывающий новый язык, или отладчик, который может отлаживать двоичные файлы, выполняясь в другой операционной системе, то для этого придется прибегнуть к VSIP. Дополнительную информацию о VSIP вы можете получить на веб-узле http://msdn.microsoft.com/vstudio/partners. Очень хорошо, что для получения VSIP SDK больше не нужно платить за лицензию. Теперь вы можете бесплатно загрузить SDK, просто приняв условия лицензионного соглашения.

Как несложно догадаться, VSIP — это очень объемная тема, которая сама по себе заслуживает отдельной книги, поэтому я не буду касаться ее. В этой главе я ставлю своей целью познакомить вас с макросами и встраиваемыми модулями, представив несколько реальных инструментов, которые помогли мне ускорить процесс разработки. Увидев, что умеют делать эти инструменты, вы получите хорошее представление об испытаниях и несчастьях, с которыми столкнетесь, попытавшись написать собственный «инструмент, без которого я не могу жить». Учитывая тот факт, что очень немногие разработчики вообще пишут мастера, их

я тоже обсуждать не буду. Что касается макросов и встраиваемых модулей, то я не буду описывать обычные шаги вроде «щелкните эту кнопку мастера, и увидите, как на экране появится встраиваемый модуль», которые обычно вы встречаете в книгах. Я предполагаю, что вы прочитали документацию по Visual Studio, поэтому потрачу время на то, чтобы обратить ваше внимание на ловушки и проблемы, с которыми довелось встретиться мне, чтобы избавить вас от потери времени, которое я потратил в попытках заставить все работать.

Сначала я рассмотрю несколько трюков, относящихся к макросам, которые вы не найдете в документации, и познакомлю вас с CommenTater — отличным макросом, проверяющим, что в проект включены самые новые комментарии из вашей XML-документации. Первый встраиваемый модуль, WhoAmI, — это чрезвычайно простой инструмент, показывающий вам учетную запись пользователя и разрешения, с которыми выполняется данный экземпляр Visual Studio. Я посвятил этому достаточно большую часть введения и повторю здесь еще раз: если вы разрабатываете программное обеспечение с привилегиями администратора, то вы совершенно, абсолютно неправы. Второй встраиваемый модуль, Hidden Settings, демонстрирует, как создавать собственные страницы свойств в диалоговом окне Options (Параметры), используя новую возможность регистрации управляемого кода Visual Studio 2005. Модуль Hidden Settings предоставляет возможность обращаться к различным скрытым и не документированным настройкам Visual Studio.

Последний встраиваемый модуль, SettingsMaster, — самый амбициозный. SettingsMaster предоставляет пакет параметров проекта .NET, позволяющий менять настройки одного или всех проектов в решении автоматически, экономя время на изменении их вручную. Например, для того чтобы изменить определения всех проектов при помощи SettingsMaster, нужно всего лишь щелкнуть кнопку. Помимо этого, в SettingsMaster входят конфигурации, меняющие настройки сборки на те, которые я рекомендовал в разделе «Запланируйте время для построения систем отладки» в главе 2. Вооружившись модулем SettingsMaster, вы сможете с легкостью координировать проекты своей команды.

Расширение при помощи макросов

Для того чтобы я мог рассказать вам о макросе CommenTater и других макросах из проекта Wintellect.VSMacros (который вы найдете в каталоге .\Macros исходного кода для этой книги), вам нужно загрузить его в окно Macro Explorer в Visual Studio. Макросы относятся к индивидуальным настройкам каждого пользователя, поэтому они не устанавливаются во время установки WintellectToolsInstall.MSI. Чтобы добавить файл Wintellect.VSMacros в интегрированную среду разработки (Integrated Development Environment, IDE), сначала откройте окно Macro Explorer (в меню View (Вид) выберите пункт Other Windows?Macro Explorer (Другие окна ▸ Macro Explorer) или нажмите клавишное сочетание Alt+F8, если используется раскладка клавиатуры по умолчанию), щелкните правой кнопкой мыши на элементе Macros в древовидном представлении и в открывшемся контекстном

меню выберите команду Load Macro Project (Загрузить проект макроса). Найдите и загрузите файл .\Macros\Wintellect.VSMacros.

Я также хочу потратить немного времени на обсуждение некоторых ключевых моментов, касающихся макросов, и некоторых проблем, с которыми вы столкнетесь. Самое главное — даже если вы полагаете, что у вас есть величайшая в мире идея для встраиваемого модуля и вам не терпится начать, прежде чем приступать к построению встраиваемых модулей, потратьте достаточно времени на написание макросов. Так как макросы могут обращаться к тем же объектам и свойствам, что и встраиваемые модули, они дают вам наилучшую возможность узнать обо всех тонкостях объектной модели Visual Studio. Как вы увидите далее в этой главе, у объектной модели есть множество хитростей, и заставить встраиваемые модули работать иногда бывает довольно сложно. Макросы намного проще писать и отлаживать, поэтому для начала нужно потренироваться на них в качестве опытного образца.

Перед тем как щелкнуть на команде Macro (Макрос) в меню Tools (Сервис), сначала прочитайте документацию по макросам и объектной модели. Описание макросов можно найти в документации по Visual Studio 2005, выполнив поиск по словам «Visual Studio Macros». Важную информацию об объектной модели вы найдете, выполнив поиск в справке по фразе «Automation Object Model Chart».

Потратив некоторое время на документацию и узнав, что представляют собой разнообразные объекты, можно попробовать записать собственный макрос, чтобы посмотреть на объекты в действии. Помните, что запись работает в основном в редакторах кода (включая диалоговые окна Find/Replace (Поиск и замена)), в обозревателе решений (Solution Explorer), а также в макросах запоминается активация окон. Невозможно записать такие вещи, как построение веб-формы или формы Windows с элементами управления. Обязательно ознакомьтесь с примерами макросов, предоставленными Microsoft, которые автоматически загружаются в окно Macro Explorer в составе проекта макросов `Samples`. Примеры макросов отлично демонстрируют использование объектной модели для устранения неполадок. Макрос `MakeAddinFromMacroProj` (из проекта макросов `MakeAddin`) — один из моих любимых, так как он умеет из макросов создавать встраиваемые модули. Он показывает, какую мощь предлагает нам сегодня Visual Studio.

Еще один мой любимчик — это модуль `Debugger`, который сообщает вам практически все, что можно сделать с объектом `Debugger`. Множество трюков, относящихся к объекту `Debugger`, вы найдете в записи блога Джима Грисмера (Jim Griesmer) по адресу http://blogs.msdn.com/jimgries/archive/2005/12/12/501492.aspx. Джим — разработчик в команде по созданию отладчика в корпорации Microsoft, и он рассказывает о своих любимых макросах отладчика, которые вам действительно стоит добавить в коллекцию.

Существует два способа выполнить макрос: дважды щелкнуть на имени функции макроса в окне Macro Explorer или воспользоваться возможностями окна Command (Команда). Как только вы начнете ввод имени макроса, откроется всплывающее окно системы IntelliSense, показанное на рис. 7.1, в котором можно выбрать и запустить нужный макрос.

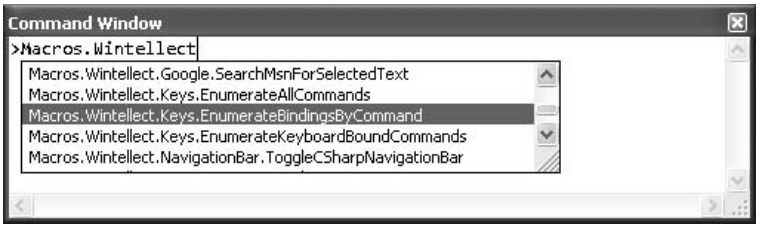

Рис. 7.1. Список макросов во всплывающем окне системы IntelliSense в окне Command (Команда)

Если вы намерены использовать для запуска макросов или любых других встроенных команд окно Command (Команда), то встроенная команда `alias` позволяет переопределять имена команд, создавая более короткие псевдонимы, чтобы каждый раз для запуска макроса вам не приходилось печатать что-то вроде этого:

```
Macros.Wintellect.CommenTater.AddNoCommentTasksForSolution
```

Для создания псевдонима для показанного выше имени выполните в окне Command (Команда) следующую команду:

```
alias CTforSLN Macros.Wintellect.CommenTater.AddNoCommentTasksForSolution
```

Также можно удалить псевдоним, передав параметр `/d` перед определенным ранее псевдонимом.

Параметры макросов

Не все знают, что во всплывающем окне системы IntelliSense, которое вы видите в окнах Macro Explorer и Command (Команда), перечисляются только макросы, не принимающие никакие параметры. Это имеет смысл в окне Macro Explorer, поскольку довольно сложно передать параметры, дважды щелкнув на элементе в окне. Однако если вы используете окно Command (Команда), то, вероятно, хотите передавать макросам какие-то параметры. Трюк заключается в том, чтобы объявить подпроцедуру макроса так, как будто она принимает один необязательный строковый параметр:

```
Sub ParamMacro(Optional ByVal theParam As String = "")
```

Изучая эту строку объявления, вы, вероятно, гадаете, как же передать макросу несколько параметров. Оказывается, когда вы передаете макросу параметры, среда макросов в Visual Studio считает их одной строкой, и разбиение этой строки на элементы для извлечения из нее отдельных параметров — ваша работа. К счастью, для того чтобы успешно выполнять синтаксический разбор, необходимо всего лишь немного знать о регулярных выражениях. Еще лучше то, что я уже сделал эту работу, так что вам нужно просто скопировать мой код.

В файле Wintellect.VSMacros в модуле `Utilities` есть функция `SplitParameters`, которая считает запятые разделителями параметров и возвращает отдельные параметры в виде строкового массива. Если вы взглянете на текст функции, то увидите, что всю работу выполняет регулярное выражение, написанное Майклом Ашем (Michael Ash). Я взял это регулярное выражение с великолепного веб-

узла RegExLib.com — это отличный ресурс, где можно найти все необходимо, что связано с регулярными выражениями.

В модуле `Utilities` вы также обнаружите несколько созданных мной удобных оберток вокруг объектов окон Command (Команда) и Output (Вывод). Они значительно упрощают отображение результатов работы макросов, и вам не приходится вручную разбираться с содержимым этих окон. Если вы просмотрите макросы в файле Wintellect.VSMacros, то увидите, что я постоянно использую эти обертки.

Отладка макроса

Так как VSA — это тоже Visual Studio, но в немного отличной форме, все трюки отладки, за исключением элементов, относящихся исключительно к C#, о которых я говорил в главе 5, «Расширенные возможности отладки с использованием Visual Studio», также распространяются на отладку макросов. Однако первое, что вы заметите при разработке макросов и встраиваемых модулей, — это то, что среда разработки Visual Studio пожирает исключения как сумасшедшая. Хотя вполне разумно, что IDE не должна пропускать исключения, чтобы они не нарушили работу самой среды, эта IDE поглощает так много необработанных исключений, что вы можете даже не осознавать, что ваш код выбрасывает исключения. Когда я разрабатывал свой первый макрос, я двадцать минут просто сидел и думал, почему же установленная мной контрольная точка никогда не срабатывает.

В итоге, для того чтобы гарантировать, что никаких сюрпризов не будет, я открыл диалоговое окно Exceptions (Исключения) в Macro IDE, выбрал в древовидном представлении узел Common Language Runtime Exceptions и установил флажок Thrown (Выброшено). Этот трюк заставляет отладчик макросов останавливаться на любом исключении. Правильные настройки показаны на рис. 7.2. Вероятно, при этом вам придется останавливаться в отладчике намного чаще, но, по крайней мере, когда дело дойдет до вашего кода, вы не будете сталкиваться с неприятными сюрпризами.

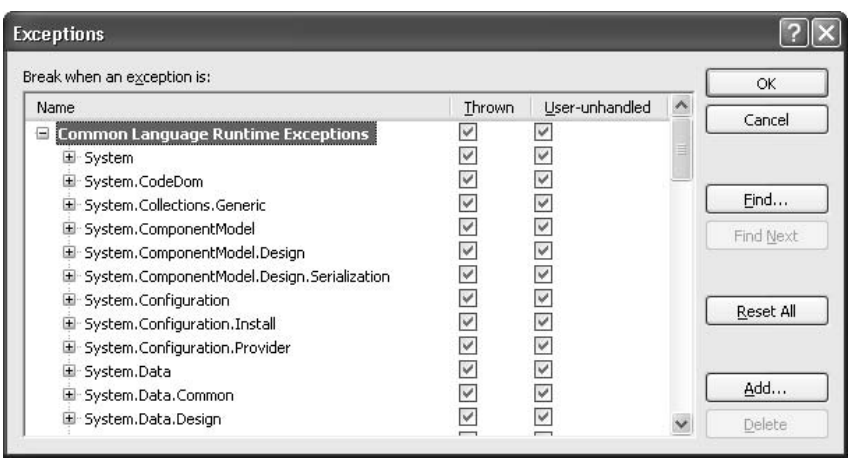

Рис. 7.2. Настройка остановки на всех исключениях в диалоговом окне Exceptions (Исключения)

Однако я также заметил, что, даже если настроить в диалоговом окне Exceptions (Исключения) остановку на любом исключении, отладчик не всегда будет это делать. Чтобы заставить отладчик макросов работать правильно, нужно после изменения настроек диалогового окна Exceptions (Исключения) установить где-нибудь в коде контрольную точку.

Элементы кода

Один из самых замечательных аспектов Visual Studio заключается в том, что при помощи объектной модели можно с легкостью обращаться ко всем программным конструкциям в исходных файлах. Тот факт, что теперь у нас есть возможность запросто добавлять, менять и удалять такие элементы, как методы, в любых языках, поддерживаемых Visual Studio, без необходимости самостоятельно выполнять синтаксический разбор, открывает огромные перспективы для всевозможных уникальных инструментов, которые могли бы никогда не быть созданы из-за того, что синтаксический разбор слишком сложен. Каждый раз, когда я применяю элементы кода для манипулирования кодом в файле, я удивляюсь, насколько же удивительна эта возможность.

Чтобы показать вам, насколько просто в действительности использовать элементы кода, я создал макрос, показанный в листинге 7.1, который выводит все элементы кода активного документа. Это может быть неочевидно, но этот код работает для любого языка, поддерживаемого Visual Studio. В выводе сообщается имя элемента кода и его тип. По типам элементы кода могут делиться на методы, переменные и т. д. Код макроса вы найдете в модуле `Examples` файла Wintellect.VSMacros вместе с поддерживающим его модулем Utilities в исходном коде для данной книги.

Листинг 7.1. Макрос DumpActiveDocCodeElements

```
' Выводит все элементы кода для открытого документа проекта.
Public Sub DumpActiveDocCodeElements()
    ' Куда попадает вывод. Обратите внимание, что класс OutputPane
    ' берется из проекта макросов Utilities.
    Dim ow As OutputPane = New OutputPane("Open Doc Code Elements")
    ' Очистка панели вывода.
    ow.Clear()

    ' Проверка наличия открытого документа.
    Dim doc As Document = DTE.ActiveDocument
    If (doc Is Nothing) Then
        ow.WriteLine("No open document")
        Exit Sub
    End If

    ' Получение модели кода для документа. Вы должны заставить элемент
    ' проекта докопаться до элементов кода
    Dim fileMod As FileCodeModel = Doc.ProjectItem.FileCodeModel

    If (fileMod IsNot Nothing) Then
```

```
        DumpElements(ow, fileMod.CodeElements, 0)
    Else
        ow.WriteLine("Unable to get the FileCodeModel!")
    End If
End Sub

Private Sub DumpElements(ByVal ow As OutputPane, _
                         ByVal elements As CodeElements, _
                         ByVal indentLevel As Integer)
    Dim currentElement As CodeElement2
    For Each currentElement In elements

        Dim i As Integer = 0

        While (i < indentLevel)
            ow.OutPane.OutputString(" ")
            i = i + 1
        End While

        ' Если какое-то исключение обращается к свойству FullName,
        ' вероятно, это неименованный параметр.
        Dim name As String
        Try
            name = currentElement.FullName
        Catch e As COMException
            name = "'Empty Name'"
        End Try
        Dim kind As vsCMElement2 = CType(currentElement.Kind, vsCMElement2)
        ow.WriteLine(Name + "(" + currentElement.Kind.ToString() + ")")

        Dim childElements As CodeElements = Nothing

        childElements = currentElement.Children

        If (Not (childElements Is Nothing)) Then
            If (childElements.Count > 0) Then
                DumpElements(ow, childElements, indentLevel + 1)
            End If
        End If
    Next
End Sub
```

CommenTater

Как вы уже могли догадаться, когда я рассказывал о пользовательских правилах Code Analysis/FxCop, которые я предоставляю в составе исходного кода для данной книги, у меня есть один пунктик: комментарии в моей XML-документации должны присутствовать для всех открытых элементов и должны всегда быть

самыми свежими. Одна из моих самых любимых возможностей Visual Studio заключается в том, что если ввести символы /// или ‘’ над логической структурой, то для этого элемента автоматически будет вставлен подходящий комментарий, зависящий от языка. Есть две причины, по которым очень важно всегда заполнять комментарии для XML-документации. Во-первых, это помогает внедрять единый стандарт комментирования среди всех членов команды, и, если уже на то пошло, во всей вселенной программирования .NET. Во-вторых, система IDE IntelliSense автоматически распознает любые добавляемые вами тэги `<summary>` и `<param>`, что приходится очень кстати, когда вашим кодом начинают пользоваться другие люди, так как это дает им намного больше ценной информации об элементе. Если код входит в состав проекта, то для того чтобы воспользоваться преимуществами комментариев в XML-документации, ничего делать не приходится. Если вы предоставляете решение только в виде двоичного файла, то комментарии можно собирать в XML-файл во время компиляции, что позволяет все так же предоставлять пользователям удобные подсказки. Все, что нужно сделать, — это поместить результирующий XML-файл в тот же каталог, где находится двоичный файл, и Visual Studio автоматически подберет комментарии, чтобы выводить их в подсказках IntelliSense. Кстати, обозреватель объектов (Object Browser) также использует этот XML-файл.

Я настоятельно рекомендую вам прочитать все о тэгах комментариев для XML-документации в документации по Visual Studio, чтобы вы как можно лучше комментировали свой код. Файлы создавать очень просто, всего лишь устанавливая параметр `/DOC` для компиляторов C# и Visual Basic, и это следует делать для каждой собираемой вами конфигурации. Чтобы вручную включить параметр `/DOC` для не-ASP.NET проектов на C#, в свойствах проекта перейдите на вкладку Build (Сборка) и в разделе Output (Вывод) установите флажок XML documentation file (XML-файл документации). По умолчанию файл записывается по пути вывода (параметр Output path в том же разделе), который сразу же жестко кодируется. Это означает, что если изменить путь вывода, то Visual Studio продолжит записывать XML-файл документации в исходный каталог. К счастью, для того чтобы справиться с этим, нужно просто сбросить и снова установить флажок XML documentation file (XML-файл документации), и последнее изменение пути вступит в силу. На рис. 7.3 показан правильно настроенный раздел Output (Вывод).

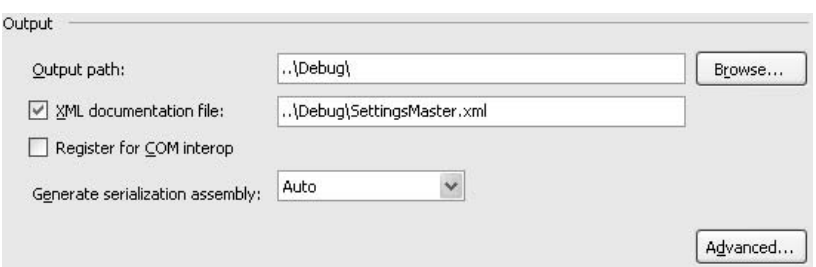

Рис. 7.3. Настройка XML-файла документации для проектов C#

Как обычно, не-ASP.NET проекты на Visual Basic немного отличаются. В свойствах проекта перейдите на вкладку Compile (Компиляция) и установите флажок Generate XML Documentation File (Создавать XML-файл документации). Visual Basic делает все правильно и всегда помещает XML-файл документации в тот же каталог, где создается сборка. В свете безрадостной перспективы вручную настраивать создание файла документации в каждой конфигурации во всех ваших проектах вы должны с энтузиазмом принять предложение воспользоваться модулем SettingsMaster, который сделает всю работу за вас.

Так как комментарии для создания документации настолько важны, мне хотелось иметь какой-то способ автоматического добавления их в мой код .NET на C# и Visual Basic. Хотя выдающийся встраиваемый модуль Роланда Вайгельта (Roland Weigelt) GhostDoc (http://www.roland-weigelt.de/ghostdoc/) превосходно подходит для автоматизации комментариев одного метода, мне нужен был инструмент, который мог бы взглянуть на все методы, которые я забыл документировать, и добавить специальные метки, позволяющие мне находить места, где комментарии были пропущены. Как раз когда я размышлял над тем, как можно было бы показать пользователю все пропущенные комментарии, я наткнулся на очень приятную возможность окна Task List (Список задач) в IDE. Если в комбинированном поле окна Task List (Список задач) выбрать значение Comments (Комментарии), то в окне будут автоматически выведены все комментарии в коде, содержащие символы TODO. Если открыть из меню Tools (Сервис) диалоговое окно Options (Параметры), а затем перейти на страницу свойств Task List (Список задач) раздела Environment (Среда), то вы увидите, что список прочих маркеров, которые IDE умеет искать в комментариях, также включает HACK, MISSING и UNDONE. Можно также добавлять собственные маркеры, такие как FIX, UPDATE и CHTO_ZA_IDIOT_ETO_NAPISAL (в маркерах нельзя использовать пробелы). Моя идея заключалась в том, чтобы создать макрос или встраиваемый модуль, который бы добавлял отсутствующие комментарии для документации и вставлял в них слово TODO, что позволяло бы с легкостью находить их и проверять, что комментарии заполнены правильно. Результатом этих размышлений стал встраиваемый модуль CommenTater. Далее показан метод, который был обработан при помощи CommenTater:

```
/// <summary>
/// TODO - Add Test function summary comment
/// </summary>
/// <remarks>
/// TODO - Add Test function remarks comment
/// </remarks>
/// <param name="x">
/// TODO - Add x parameter comment
/// </param>
/// <param name="y">
/// TODO - Add y parameter comment
/// </param>
/// <returns>
/// TODO - Add return comment
```

```
/// </returns>
public static int Test ( Int32 x , Int32 y )
{
    return ( x ) ;
}
```

Хотя это превосходная идея — показывать элементы TODO в окне Task List (Список задач), есть небольшая проблема, связанная с языком C# в Visual Studio 2005: элементы TODO не отображаются. В предыдущих версиях Visual Studio все работало и для C#, но в Visual Studio 2005 поддерживается только Visual Basic. Надеюсь, что в очередном выпуске пакета обновления это будет исправлено.

Visual Studio делает перебор элементов кода в исходном файле тривиальной задачей, поэтому я очень обрадовался, когда представил, что все, что мне нужно было сделать, — это пройтись по элементам кода, проверить строки над методами или свойствами и, если комментарии для документации в них отсутствуют, вставить комментарий для данного метода или свойства. Но после этого я обнаружил, что у всех элементов кода есть свойство с именем DocComment, возвращающее фактические соответствующие им комментарии для документации. Я немедленно сказал мысленное спасибо разработчикам за заботу и за то, что они действительно сделали элементы кода полезными вещами. Теперь мне всего лишь нужно было установить подходящее значение свойства DocComment.

Открыв макросы для этой книги, дважды щелкните пункт Wintellect, затем правой кнопкой мыши щелкните на модуле CommentTater и в контекстном меню выберите команду Edit (Правка). Исходный код для макроса слишком объемен, чтобы включать его в книгу, поэтому просто следите за ходом рассуждения по файлу, пока я рассказываю об идеях, лежащих в основе реализации макроса. Моя главная идея — создать две функции, AddNoCommentTasksForSelectedProjects и CurrentSourceFileAddNoCommentTasks. Исходя из имен функций, вы можете догадаться, на каком уровне они должны работать. Большей частью базовый алгоритм работает так же, как и пример в листинге 7.1, то есть я просто прохожу по всем элементам кода и обрабатываю свойство DocComment для каждого типа элементов кода.

Я сразу же столкнулся с проблемой, которую считаю недостатком дизайна объектной модели элементов кода. Свойство DocComment – это не общее свойство класса CodeElement, которое можно подставлять в качестве базы для любого общего элемента кода. Поэтому мне пришлось преобразовывать общий объект CodeElement в фактический тип, основанный на свойстве Kind. Вот почему в функции RecurseCodeElements есть большой оператор Select…Case.

Вторую проблему я создал себе сам. По какой-то причине я никогда не задумывался о том, что со свойством DocComment для логической структуры нужно работать как с полностью сформированным XML-фрагментом. Я составлял нужную мне строку комментария, а когда пытался присвоить ее свойству DocComment, выбрасывалось исключение ArgumentException. Это приводило меня в замешательство, так как я считал, что свойство DocComment должно было быть доступно и для чтения, и для записи, но оно вело себя так, как будто бы разрешалось только чтение. Вероятно, у меня был спазм мозга, и поэтому я не понимал, что причина, почему выбрасывалось исключение, заключалась в том, что я не оборачивал мои

комментарии для XML-документации в необходимый элемент <doc></doc>. Конечно же, полагая, что свойство DocComment доступно только для чтения, я реализовал все обновления комментариев для документации, используя текстовую вставку. Однако после того как спазм мозга прошел, я догадался, что мне всего лишь нужно было вставлять в свойство DocComment правильно сформированную XML-строку и оставлять задачу вставки на усмотрение Visual Studio. Помимо этого, я хочу упомянуть о еще одной странности свойства DocComment: в C# к свойству DocComment правильно добавляется корневой элемент <doc></doc>, но в Visual Basic .NET этого не происходит.

В листинге 7.2 показана функция ProcessFunctionComment, и, как вы видите, я пользуюсь преимуществом наличия XML-классов в библиотеке Microsoft .NET Framework, для выполнения всей тяжелой работы, необходимой для получения существующих строк комментариев, и создания их новых версий. Функция ProcessFunctionComment может менять порядок следования комментариев для документации; мне пришлось выбрать конкретный порядок помещения в файл отдельных узлов. Кроме этого, я форматирую комментарии так, как мне это нравится, поэтому CommenTater может менять ваше тщательное форматирование комментариев, но вы не потеряете никакую информацию.

Листинг 7.2. Функция ProcessFunctionComment из модуля CommenTater

```
' Выполняет работу по проверке существующего комментария для функции, чтобы
' удостовериться, что все правильно. При этом порядок комментариев может
' меняться, поэтому может понадобиться исправить его.
Private Sub ProcessFunctionComment(ByVal func As CodeFunction)

    Debug.Assert("" <> func.DocComment, """""" <> Func.DocComment")

    ' Содержит исходный комментарий для документации.
    Dim xmlDocOrig As New XmlDocument
    ' МНЕ НРАВИТСЯ ЭТО! Если присвоить свойству PreserveWhitespace значение
    ' true, то класс XmlDocument сохраняет большую часть форматирования...
    xmlDocOrig.PreserveWhitespace = True

    ' Это раздражает. Так как Visual Basic не добавляет корневой элемент
    ' <doc>, мне приходится добавлять его, чтобы работал метод LoadXml.
    Dim xmlString As String = func.DocComment
    If (func.Language = CodeModelLanguageConstants.vsCMLanguageVB) Then
        Dim sb As StringBuilder = New StringBuilder()
        sb.AppendFormat("<doc>{0}</doc>", xmlString)
        xmlString = sb.ToString()
    End If

    ' Говоря о противоречиях... Несмотря на то, что я сохраняю все
    ' форматирование любого внутреннего кода XML, при установке свойства C#
    ' DocComment пустые строки CRLF удаляются. Visual Basic сохраняет
    ' форматирование целиком.
```

продолжение

Листинг 7.2. (*продолжение*)

```
    xmlDocOrig.LoadXml(xmlString)

    Dim rawXML As New StringBuilder

    ' Имя функции правильно отформатировано для XML.
    Dim FName As String = ProperlyXMLizeFuncName(func.Name)

    ' Получение сводного узла.
    Dim node As XmlNode
    Dim nodes As XmlNodeList = xmlDocOrig.GetElementsByTagName("summary")

    If (0 = nodes.Count) Then
        rawXML.Append(SimpleSummaryComment(func, FName, "function"))
    Else
        rawXML.AppendFormat("<summary>{0}", vbCrLf)
        For Each node In nodes
            rawXML.AppendFormat("{0}{1}", node.InnerXml.Trim(), vbCrLf)
        Next
        rawXML.AppendFormat("</summary>{0}", vbCrLf)
    End If

    ' Получение узла с примечаниями.
    nodes = xmlDocOrig.GetElementsByTagName("remarks")
    If (nodes.Count > 0) Then
        rawXML.AppendFormat("<remarks>{0}", vbCrLf)
        For Each node In nodes
            rawXML.AppendFormat("{0}{1}", node.InnerXml.Trim(), vbCrLf)
        Next
        rawXML.AppendFormat("</remarks>{0}", vbCrLf)
    ElseIf (True = addRemarksToFunctions) Then
        rawXML.AppendFormat("<remarks>{0}TODO - Add {1} function " + _
                                "remarks comment{0}</remarks>", _
                            vbCrLf, FName)
    End If

    ' Получение всех параметров, описанных в комментариях для документации.
    nodes = xmlDocOrig.GetElementsByTagName("param")

    ' У функции есть параметры?
    If (0 <> func.Parameters.Count) Then

        ' Забрасываю все существующие параметры в комментариях для
        ' документации в хэш-таблицу с именем параметра в качестве ключа.
        Dim existHash As New Hashtable
        For Each node In nodes
            Dim paramName As String
            Dim paramText As String
```

```
            paramName = node.Attributes("name").InnerXml
            paramText = node.InnerText.Trim()
            existHash.Add(paramName, paramText)
        Next

        ' Цикл прохождения по параметрам.
        Dim elem As CodeElement
        For Each elem In func.Parameters
            ' Этот уже находится в хэше среди ранее внесенных параметров?
            If (True = existHash.ContainsKey(elem.Name)) Then
                rawXML.AppendFormat("<param name=""{0}"">{1}{2}{1}" + _
                                                        "</param>{1}", _
                                    elem.Name, _
                                    vbCrLf, _
                                    existHash(elem.Name))
                ' Избавиться от этого ключа.
                existHash.Remove(elem.Name)
            Else
                ' Был добавлен новый параметр.
                rawXML.AppendFormat("<param name=""{0}"">{1}TODO - Add " + _
                                    "{0} parameter comment{1}</param>{1}", _
                                    elem.Name, vbCrLf)
            End If
        Next

        ' Если в хэш-таблице что-нибудь осталось, то параметр был либо
        ' удален, либо переименован. Я перечисляю все, что осталось, с
        ' использованием элементов TODO, чтобы пользователь мог удалить
        ' эти комментарии вручную.
        If (existHash.Count > 0) Then
            Dim keyStr As String
            For Each keyStr In existHash.Keys
                Dim desc As Object = existHash(keyStr)
                rawXML.AppendFormat("<param name=""{0}"">{1}{2}{1}{3}" + _
                                                    "{1}</param>{1}", _
                                    keyStr, _
                                    vbCrLf, _
                                    desc, _
                                    "TODO - Remove param tag")
            Next
        End If
    End If

    ' Если необходимо, обработать возвращаемые значения.
    If ("" <> func.Type.AsFullName) Then
        nodes = xmlDocOrig.GetElementsByTagName("returns")
        ' Добавить узлы для возвращаемых значений, но только если таковые
        ' имеются.
```

продолжение

Листинг 7.2. (*продолжение*)

```
        If (0 = nodes.Count) Then
            If (String.IsNullOrEmpty(func.Type.AsFullName) = False) And _
             (String.Compare(func.Type.AsFullName, "System.Void") <> 0) Then
             rawXML.AppendFormat("<returns>{0}TODO - Add return comment" + _
                                                    "{0}</returns>", _
                                 vbCrLf)
            End If
        Else
            rawXML.AppendFormat("<returns>{0}", vbCrLf)

            For Each node In nodes
                rawXML.AppendFormat("{0}{1}", node.InnerXml.Trim(), vbCrLf)
            Next

            rawXML.Append("</returns>")
        End If
    End If

    ' Скопировать существующие узлы с примерами, если таковые имеются.
    nodes = xmlDocOrig.GetElementsByTagName("example")
    If (nodes.Count > 0) Then
        rawXML.AppendFormat("<example>{0}", vbCrLf)
        For Each node In nodes
            rawXML.AppendFormat("{0}{1}", node.InnerXml.Trim(), vbCrLf)
        Next
        rawXML.AppendFormat("</example>{0}", vbCrLf)
    End If

    ' Скопировать существующие узлы с разрешениями, если таковые имеются.
    nodes = xmlDocOrig.GetElementsByTagName("permission")
    If (nodes.Count > 0) Then
        For Each node In nodes
            rawXML.AppendFormat("<permission cref=""{0}"">{1}", _
                                node.Attributes("cref").InnerText, _
                                vbCrLf)
            rawXML.AppendFormat("{0}{1}", node.InnerXml.Trim(), vbCrLf)
            rawXML.AppendFormat("</permission>{0}", vbCrLf)
        Next
    End If

    ' Наконец, исключения.
    nodes = xmlDocOrig.GetElementsByTagName("exception")

    If (nodes.Count > 0) Then
        For Each node In nodes
            rawXML.AppendFormat("<exception cref=""{0}"">{1}", _
                                node.Attributes("cref").InnerText, _
                                vbCrLf)
```

```
            rawXML.AppendFormat("{0}{1}", node.InnerXml.Trim(), vbCrLf)
            rawXML.AppendFormat("</exception>{0}", vbCrLf)
        Next
    End If
    func.DocComment = FinishOffDocComment(func, rawXML.ToString())
End Sub
```

После того как обновление комментариев для документации заработало, я подумал, что было бы здорово сделать шаг вперед и реализовать код для обработки контекста отмены действий. Это позволило бы вам в случае ошибки при помощи одного нажатия клавишного сочетания Ctrl+Z восстанавливать все изменения, внесенные CommenTater. В предыдущих версиях в контекстах отмены действий существовала ошибка, которая проявлялась при глобальной отправке, позволяющей одной операцией отменять все изменения, сделанные макросом. К счастью, разработчики исправили эту ошибку, и теперь я могу предложить возможность глобальной отмены. Если вам нравится идея отмены индивидуальных операций, то присвойте полю individualUndo наверху файла значение True.

Еще одно усовершенствование, которое мне пришлось сделать, чтобы заставить макрос работать не только в Visual Studio .NET 2003, но и в Visual Studio 2005, заключалось в том, что я перекроил перечисление проектов. Предыдущие версии Visual Studio не поддерживали папки с проектами, поэтому мой макрос находил только файлы, находившиеся на верхнем уровне. Так как уровней вложенности папок может быть произвольное количество, мне пришлось всего лишь реализовать рекурсивный просмотр всех элементов проекта, для которых свойство FileCodeModel было не пустым. Если перед вами стоит задача перечислять элементы проекта, то обязательно взгляните на макрос ListProj в файле Samples.VSMacros — там вы найдете в точности то, что вам необходимо.

Помимо поля individualUndo, о котором я упомянул выше, есть еще два необязательных поля, которые вы можете захотеть изменить. По умолчанию макрос CommenTater добавляет комментарии только для открытых элементов в сборке. Для того чтобы отсутствующие комментарии добавлялись для всех методов, присвойте полю VisibleOnly наверху файла значение False. Последнее необязательное поле называется addRemarksToFunctions. Как можно понять из названия, если значение поля равно True, то макрос проверяет в комментариях для всех методов наличие раздела <remarks>...</remarks>, и если этот раздел отсутствует, он добавляется с использованием элемента TODO.

Я использую макрос CommenTater постоянно, и я уверен, что вы также найдете его очень полезным для разработки. Как и в любом коде, в нем еще достаточно пространства для усовершенствований. Мне хотелось бы интегрировать в код автоматическое генерирование документации <exception>...</exception>. Так как объекты CodeElement не могут проникать в методы, это требует выполнения синтаксического анализа фактического текста метода или свойства. Звучит как хорошая задача для регулярного выражения, не так ли? Тони Чоу (Tony Chow) уже представил ее решение в одной из статей в MSDN Magazine (http://msdn.microsoft.com/msdnmag/issues/05/07/XMLComments/default.aspx). Также можно было бы реализовать еще одну возможность, но на нее может потребоваться чуть больше усилий — хорошо было бы позволять пользователям указывать, какое форматирование

они хотели бы применять в своих комментариях для XML-документации. Мне нравится мой стиль, когда все маркеры элементов и внутренний текст находятся на отдельных строках, но другим он может быть не по вкусу. Если у вас есть идеи относительно каких-либо еще возможностей, то не стесняйтесь и пишите мне о них — я обязательно рассмотрю все предложения.

Больше макросов для вас

Просматривая файл Wintellect.VSMacros, вы встретие множество других модулей, содержащих макросы, которые будут вам полезны в ежедневной разработке. В главе 5, в разделе «Расширенные контрольные точки и как их использовать» я упомянул модуль `BreakPointHelper`, содержащий два макроса, `SetBreakpointsOnDocMethods` и `RemoveBreakpointsOnDocMethods`. Эти два макроса проходят по элементам кода в активном документе, и один из них устанавливает контрольную точку в точке входа каждого метода и свойства в файле, а другой, соответственно, удаляет. Эти макросы пользуются преимуществами свойства `BreakPoint.Tag` для идентификации контрольных точек, установленных макросом `SetBreakpointsOnDocMethods`, чтобы макрос `RemoveBreakpointsOnDocMethods` удалял только эти контрольные точки. Таким образом, никакие из существующих установленных вами контрольных точек этими макросами не удаляются.

Также в разделе «Трассировочные точки» пятой главы я обсудил модуль `TracePointMacros`. Макрос в этом модуле демонстрирует, как обойти крупнейшее ограничение трассировочных контрольных точек, заставляя IDE думать, что она останавливается в контрольной точке, но одновременно используя другой поток для того, чтобы приказать отладчику продолжать исполнение.

Пометки исправлений в редакторе Visual Studio великолепны — небольшой желтый прямоугольник рядом со строкой указывает, что вы изменили ее содержимое, но еще не сохранили новую версию. Небольшой зеленый прямоугольник указывает на уже сохраненные изменения. К сожалению, не существует способа удалить эти пометки в редакторе, чтобы увидеть изменения между двумя вариантами, и поэтому приходится закрывать окно редактора и снова открывать его. Но когда вы повторно открываете окно, курсор не перескакивает на место последнего изменения в документе, а остается на первой строке в первом столбце. Так как очень часто возникает необходимость проверять отдельные изменения кода, модуль `RevMarks` содержит единственный макрос под названием `ClearRevMarks`, который сохраняет текущее местоположение и выделенный фрагмент файла, закрывает окно исходного кода, заново его открывает и помещает курсор на старое место, восстанавливая выделение. Когда я начал писать этот макрос, я думал, что он займет не больше десяти строк, но в Visual Studio все обычно оказывается не так просто. Вы сможете увидеть, через какие испытания мне пришлось пройти, если прочитаете исходный код макроса.

Модуль `Keys` родился из моего разочарования, когда я искал клавишное сочетание, привязанное к определенной команде. Страница свойств Keyboard (Клавиатура), которая открывается из узла Environment (Среда) диалогового окна Options (Параметры), превосходно подходит для привязки команды к клавишному сочетанию, но на ней невозможно найти клавишные сочетания, выполняющие различ-

ные команды в разных редакторах или контекстах. Первый макрос, EnumerateAllCommands, перечисляет все команды в IDE в алфавитном порядке, а также все привязанные к ним клавишные сочетания. Так как разнообразные инструменты в IDE могут включать команды с одинаковыми именами, например Edit.Breakline, то в строку с указанием клавишного сочетания также включается название инструмента, для которого оно определено. Например, в следующем фрагменте вывода видно, что команде Edit.Breakline соответствует четыре разных привязки, а соответствующие инструменты перечисляются слева от символа ::.

```
Edit.BreakLine (Windows Forms Designer::Enter |
                Text Editor::Shift+Enter |
                Text Editor::Enter |
                Report Designer::Enter)
```

Два других макроса в модуле Keys, EnumerateBindingsByCommand и EnumerateKeyboardBoundCommands схожи в том, что они показывают только команды, привязанные к клавишным сочетаниям. Макрос EnumerateBindingsByCommand выводит список, отсортированный по клавишным сочетаниям, а макрос EnumerateKeyboardBoundCommands сортирует вывод по команде. Я, как настоящий фанат клавиатуры, выучил огромное количество новых клавишных сочетаний, изучая вывод этих двух макросов. Мой карпальный туннельный синдром обостряется при использовании мыши, так что чем больше я держу пальцы на клавиатуре в Visual Studio, тем здоровее остаюсь.

Мы все хотим, чтобы среда Visual Studio работала как можно быстрее, и Роланд Вайгельт представил в своем блоге по адресу http://weblogs.asp.net/rweigelt/archive/2006/05/16/446536.aspx макрос по названием ToggleCSharpNavigationBar, который включает и выключает отображение полосы навигации наверху редактора C#. Роланд рассказывает, что при выключении отображения навигационной полосы, включающей два комбинированных поля наверху окна редактора, которые позволяют находить классы и методы в файле, редактор начинает работать быстрее. Хотя я не проводил никакие тесты производительности, мне тоже кажется, что скорость работы возрастает. Так как я никогда не использую полосы навигации, то их отсутствие на экране для меня не проблема. Роланд любезно разрешил мне включить его макрос в модуль NavigationBar.

Два других модуля, SelectedSearch и FormatToHtml, принадлежат Джеффу Атвуду (Jeff Atwood) и взяты из его знаменитого блога CodingHorror.com, на который вы просто обязаны подписаться. В записи Google Search VS.NET (http://www.codinghorror.com/blog/archives/000428.html) он представляет отличный макрос, который при помощи поисковой машины Google выполняет поиск текста, выделенного в окне исходного кода или окне Output (Вывод). Так как функция поиска в библиотеке MSDN для Visual Studio 2005 работает очень медленно и не слишком хорошо, превосходный макрос Джеффа SearchGoogleForSelectedText значительно облегчает поиск. Я добавил в модуль SelectedSearch два макроса: SearchGoogleMicrosoftForSelectedText, который выполняет поиск выделенного текста в разделе Google, относящемся к Microsoft, и SearchMsnForSelectedText, который выполняет поиск выделенного текста в MSN. Я привязал эти макросы к клавишным сочетаниям Alt+G и Alt+M и совершенно счастлив.

Если вы когда-либо вставляли код из редактора Visual Studio в Microsoft Word, редактор HTML или собственный блог, то знаете, что вместо ожидаемого кода HTML вы получаете какой-то уродливый текст в формате Rich Text Format. Джеффу хотелось видеть чистый и опрятный HTML-код, поэтому он написал несколько превосходных макросов, позволяющих помещать в буфер обмена чистый HTML (http://www.codinghorror.com/blog/archives/000429.html). Для большей части кода вы будете использовать макросы `Modern` и `ModernText`, так как они создают HTML-код с элементами `<div>` и `<span>`. Если вы работаете в среде, в которой эти коды отсутствуют, то используйте макросы `Legacy` и `LegacyText`. Для человека, подобного мне, кто помещает в документацию тонны кода, эти макросы просто бесценны. Я также благодарен Джеффу за разрешение включить их в исходный код для данной книги.

Хотя я использую макросы Джеффа, необходимо также упомянуть, что многим людям очень нравится встраиваемый модуль Колина Колера (Colin Coller) CopySourceAsHtml (CSAH), который можно загрузить с веб-узла http://www.jtleigh.com/people/colin/software/CopySourceAsHtml. Встраиваемый модуль CSAH поддерживает фоновые цвета, настройки нумерации строк и подсвечивание синтаксиса. Он поставляется с полным исходным кодом и представляет собой отличный пример того, как надо писать хорошие встраиваемые модули.

Встраиваемые модули Visual Studio

Макросы превосходно подходят на небольших изолированных задач, но если у вас более сложный пользовательский интерфейс или высокие требования к вводу данных, а также если вы хотите защитить исходный код, то тогда вашим выбором должны стать встраиваемые модули. Хотя написать макрос намного проще, встраиваемые модули позволяют справляться со следующими задачами, недоступными макросам:

- добавление собственных окон инструментов и диалоговых окно в интегрированную среду разработки;
- добавление собственных командных панелей (то есть меню и панелей инструментов) в интегрированную среду разработки;
- добавление собственных страниц свойств в диалоговое окно Options (Параметры).

Как вы через мгновение увидите, разработка и отладка встраиваемых модулей намного труднее, чем создание макросов, поэтому я настоятельно рекомендую перед тем, как браться за встраиваемые модули, пытаться реализовать необходимое при помощи макроса.

По существу, встраиваемые модули — это объекты COM, которые подключаются к интегрированной среде разработки. Если вы беспокоились о том, что можете утерять все знания о COM, полученные за последние несколько лет, то не бойтесь — они вам еще понадобятся в мире встраиваемых модулей. Интересно, что, так как управляемые языки поддерживают COM, вы можете свободно писать встраиваемые модули на Visual Basic и C#. Хотя мне нравится язык C++,

повышение производительности в .NET мне нравится еще больше, поэтому в этой главе я сконцентрируюсь на вопросах, связанных с написанием встраиваемых модулей на управляемых языках.

Как обычно, вы должны начать свое путешествие в мир встраиваемых модулей с прочтения документации. Во-вторых, вам необходимо посетить страницу http://go.microsoft.com/fwlink/?linkid=57538, с которой можно загрузить образцы автоматизации Visual Studio 2005. Там вы найдете множество прекрасных примеров, демонстрирующих, как выполнять многие распространенные задачи, касающиеся разработки встраиваемых модулей и мастеров.

Те из нас, кто был вынужден писать встраиваемые модули в предыдущих версиях Visual Studio, обнаружат несколько очень приятных изменений в модели Visual Studio 2005. По-моему мнению, лучшее среди них — это новая великолепная модель регистрации. Больше не нужно использовать реестр для того, чтобы сообщать Visual Studio о существовании нового встраиваемого модуля. Просто поместите регистрационный файл, имя которого заканчивается на .AddIn, в подходящий каталог, и Visual Studio с готовностью начнет использовать ваш встраиваемый модуль. Для пользователя, зарегистрированного в данный момент в системе, файлы .AddIn находятся в каталоге C:\Documents and Settings*имя_пользователя*\My Documents\Visual Studio 2005\Addins. Для всех пользователей общее местоположение C:\Documents and Settings\All Users\Application Data\Microsoft\MSEnvShared\Addins.

Второе грандиозное изменение заключается в том, что страницы параметров можно создавать напрямую из .NET, и этот процесс больше не требует наличия пользовательского компенсационного COM-компонента. Также компенсационные компоненты COM больше не нужны для создания окон инструментов .NET — возможности, позволяющей вам добавлять в интегрированную среду разработки собственные окна. В действительности, создание окон инструментов — настолько тривиальная задача, что я лучше отправлю вас к документации по Visual Studio по адресу http://msdn2.microsoft.com/en-us/library/envdte80.windows2.createtoolwindow2.aspx. Прочие усовершенствования включают лучшую поддержку локализации и немного улучшенные панели инструментов.

Трюки разработки встраиваемых модулей

Как и в любой сфере разработки, в документации рассказывается об основах, но не говорится о том, в какие неприятности вы можете попасть во время разработки собственных встраиваемых модулей. Чтобы вам не пришлось перегружать серверы Google поисковыми запросами в попытках найти настоящие трюки разработки встраиваемых модулей, я решил дать вам ответы на вопросы, с которыми сам столкнулся во время разработки встраиваемых модулей. Все примеры вы найдете в исходном коде для данной книги. В этом разделе я предполагаю, что вы уже прочитали документацию по встраиваемым модулям на веб-узле http://msdn2.microsoft.com/en-us/library/5abkeks7.aspx.

Для того чтобы сфокусироваться на процессе разработки встраиваемых модулей, я представлю вам первый из созданных мной модулей — `WhoAmI`. В разделе «Почему я всегда должен разрабатывать под учетной записью, не имеющей прав

администратора?» главы 4, «Часто задаваемые вопросы об отладке приложений на платформе .NET», я очень подробно рассказал, почему нет никакой причины когда-либо запускать программное обеспечение для разработки с правами администратора. Единственная ситуация, когда вам могут понадобиться права администратора, — это отладка с разными учетными записями пользователя. Но в подобных случаях вы всего лишь запускаете отладчик с администраторскими правами, а в системе продолжаете работать как член группы Users (Пользователи). Так как у вас выполняется несколько экземпляров Visual Studio, вы можете запутаться в учетных записях, с которыми работает каждый из экземпляров. Встраиваемый модуль `WhoAmI` всего лишь создает панель инструментов с кнопкой, которая отображает рисунок и сведения об учетной записи пользователя для текущего экземпляра IDE.

Единственная интересная деталь во встраиваемом модуле `WhoAmI` заключается в том, что единственная добавляемая в IDE команда, `WhoAmI.Rights`, заново создается каждый раз, когда IDE запускается, так как IDE не обрабатывает динамические кнопки на панелях инструментов. Если я не буду выполнять собственное обновление динамической кнопки, то встраиваемый модуль `WhoAmI` будет показывать неправильную картинку после изменения прав учетной записи. Вероятно, вы не будете вручную удалять и добавлять свою учетную запись в группу Administrators (Администраторы), но вы вполне можете использовать удивительный командный файл MakeMeAdmin Аарона Маргозиса (Aaron Margosis), чтобы временно поднимать уровень своей учетной записи из группы Users (Пользователи) до администраторского. Подробнее о файле MakeMeAdmin вы можете прочитать в блоге Аарона по адресу http://blogs.msdn.com/aaron_margosis/archive/2004/07/24/193721.aspx.

Несмотря на то, что `WhoAmI` не так интересен с точки зрения других встраиваемых модулей, его достаточно, чтобы продемонстрировать вам все необходимые составляющие встраиваемого модуля. Он создает панель инструментов и поддерживает полную локализацию. Он также достаточно мал, чтобы понять его с первого взгляда. Знакомясь с советами в этом разделе, следите за соответствующим кодом в файле .\WhoAmI\WhoAmI.sln.

После того как вы закончите формирование основы своего будущего шедевра в мастере создания встраиваемых модулей, вероятно, вы захотите изменить имя главного класса модуля. Код, генерируемый мастером, считает указанное вами имя именем пространства имен для всего, что включает в себя встраиваемый модуль. Он также присваивает имя `Connect` классу, в котором реализуется интерфейс `IDTExtensibility2` и необязательные интерфейсы `IDTCommandTarget`. Единственная проблема заключается в том, что для выгрузки класса из сборки встраиваемого модуля интегрированной среде разработки требуется его полное имя. Таким образом, у любых добавляемых вами команд полное имя выглядит как МойВстраиваемыйМодуль.`Connect`.МояКоманда.

Лично мне не нравится видеть эту вставку `Connect` в середине всех моих команд. Я предпочитаю, чтобы у всех команд, поддерживаемых моим встраиваемым модулем, имя включало только имя встраиваемого модуля и саму команду. По-моему, это выглядит более профессионально и значительно упрощает доступ

к командам из окна Command (Команда). Мои коллеги могут подтвердить, что иногда я бываю перфекционистом. Исправить это незначительное упущение в именах команд относительно просто.

Трюк заключается в том, чтобы присвоить классу, в котором реализуется интерфейс IDTExtensibility2 и необязательные интерфейсы IDTCommandTarget, имя вашего встраиваемого модуля и поместить его за пределами любых пространств имен. Так как тип, находящийся вне пространств имен, будет вызывать предупреждение Code Analysis «Declare types in namespaces» (Объявляйте типы в пространствах имен), вам необходимо подавить эту ошибку в своем коде. Если вы взглянете на содержимое файла .\WhoAmI\Connect.cs, то увидите пример использования атрибута SuppressMessage для решения этой задачи.

В случае более сложных встраиваемых модулей, для которых вы будете создавать несколько файлов кода, нужно будет менять имя пространства имен по умолчанию, совпадающее с именем встраиваемого модуля, на что-то другое, что будет приводить к конфликту с классом, открытым за пределами пространств имен. Для встраиваемого модуля SettingsMaster я присвоил пространству имен по умолчанию имя SettingsMasterWorker.

Хотя только вы принимаете решение, как именовать свои команды, обычно все исправляют способ регистрации команд и создания панелей инструментов в сгенерированном мастером коде. Регистрация осуществляется в унаследованном от IDTExtensibility.OnConnection методе, но вы должны принять за хорошую привычку разносить регистрацию команд и создание панелей инструментов в отдельные собственные методы. Это значительно упрощает добавление команд во встраиваемый модуль, так как они обрабатываются в коде в одном месте.

Все не прекращаются споры насчет того, нужно ли пытаться удалять свои команды перед их добавлением, когда при помощи значения ext_ConnectMode.ext_cm_UISetup вызывается унаследованный от IDTExtensibility.OnConnection метод. Я всегда удаляю и заново создаю мои команды, чтобы гарантировать, что на момент их добавления мне известно точное состояние команд. Что касается панелей инструментов, то их можно добавлять только один раз, поэтому ваш код инициализации сначала должен проверять наличие панели инструментов в коллекции CommandBars, и, если при этом создается исключение ArgumentException, то только в этом случае создавать саму панель инструментов. Вы можете увидеть пример того, как это делаю я, в методе AddButtonToCommandBar в файле .\WhoAmI\Connect.cs.

Так как сфера локализации претерпела значительные усовершенствования во встраиваемых модулях в Visual Studio 2005, я был удивлен, что в описании обработки сателлитных сборок в разделе документации «Walkthrough: Creating Managed Satellite DLLs» говорится, что вы должны открыть командную строку и вручную выполнить команду Resgen.exe, чтобы скомпилировать файл .resx в файл .resources, и применить компоновщик сборок AL.exe для создания сателлитной сборки. Хотя можно сжульничать и реализовать эти шаги в событиях сборки, определенно, это очень неэффективный и не масштабируемый подход.

Я подозреваю, что команда по расширению просто не разговаривала с остальными членами команды по реализации IDE, потому что если вам необходимы

локализованные сателлитные сборки в любых других проектах .NET, вы просто добавляете к проекту новый файл ресурсов и вносите название культуры в имя файла. Например, для того чтобы добавить ресурсы для канадского французского языка, нужно добавить к проекту файл ресурсов с именем Resources.fr-ca.resx. IDE соберет ресурсы культуры и автоматически поместит их в подходящий каталог, где собирается главная сборка. Игнорируйте в документации по встраиваемым модулям обсуждение локализации и пользуйтесь для локализации своих модулей стандартной поддержкой локализации .NET в IDE. Прочитать об этом можно на веб-узле http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vbcon/ html/vbwlkwalkthroughlocalizingwindowsforms.asp.

Разрабатывая модуль `WhoAmI`, я хотел продемонстрировать вам полную локализацию и наткнулся на три ошибки. Во-первых, в качестве имен любых битовых рисунков, которые вы хотите использовать на панелях инструментов, могут выступать только целые значения — это необходимо, когда вы передаете их методу `Commands2.AddNamedCommand2`, который устанавливает рисунок для панели инструментов. Однако редактор ресурсов в Visual Studio сообщает, что битовые рисунки с целочисленными именами, которые вы добавляете в файл .resx, обладают недопустимыми именами. К счастью, файл .resx все равно компилируется.

Вторая относящаяся к локализации проблема связана с тем, как ваш встраиваемый модуль отображается в поле About (О программе) IDE и в диалоговом окне Add-In Manager (Диспетчер встраиваемых модулей). В документации по встраиваемым модулям рассказывается, как локализовать элементы в файле .AddIn (регистрационном файле встраиваемого модуля), указав строку с путем к файлу ресурсов, к которой в начале добавлен символ `@`. Из документации неочевидно то, что элемент `<Assembly>` в файле .AddIn должен содержать полный путь к указанной сборке, или строки и значки будут загружаться в поле About (О программе) неправильно. Странно, однако, что в диалоговом окне Add-In Manager (Диспетчер встраиваемых модулей) локализованные данные отображаются правильно, даже если в узле `<Assembly>` не указан полный путь.

Во время разработки встраиваемых модулей можно без проблем указывать полный путь к сборке в файле .AddIn, но при установке все становится немного сложнее. Так как я использую превосходный набор инструментов Windows Installer XML (WiX), то решение оказывается довольно простым. Поскольку файл .AddIn — это XML-файл, я использую встроенные в WiX пользовательские действия XML для записи полного пути во время установки. Вы можете увидеть, как реализовано обновление, в файле .\Instal\WintellectTools\AddIns.wxs.

Последняя найденная мной ошибка локализации не демонстрируется во встраиваемом модуле `WhoAmI`, но вы встретитесь с ней в следующем встраиваемом модуле, `HiddenSettings`. Теперь IDE считывает страницы параметров вашего встраиваемого модуля из файла .AddIn, и это значительное усовершенствование по сравнению с предыдущими версиями. Вы указываете категорию верхнего уровня, которая выводится в корне дерева в диалоговом окне Options (Параметры), и перечисляете все остальные подкатегории, которые выводятся как дочерние узлы дерева страниц параметров встраиваемого модуля. К сожалению, не существует способа указать локализованные имена категории и подкатегорий. Во время

установки необходимо либо использовать другой файл .AddIn, выбираемый в зависимости от языка пользовательского интерфейса, либо записывать подходящие значения в ходе установки. Надеюсь, что Microsoft исправит эту ошибку в следующей версии Visual Studio.

Последний раздел этого списка трюков относится к панелям инструментов. Как и в предыдущих версиях Visual Studio, панели инструментов представляют собой самую проблемную область разработки встраиваемых модулей. Вероятно, вы надеетесь, что за пятнадцать лет существования Visual C++ и Visual Studio уже должно было быть изобретено практически идеальное решение для создания панелей инструментов. Но мы все так же ограничены базовыми кнопками и комбинированными полями на панелях инструментов из встраиваемых модулей, что заставляет ломать голову над тем, как обойти эти ограничения, вместо того чтобы продумывать наилучший пользовательский интерфейс для решения конкретной задачи. Хотя Microsoft, наконец-то, упростила удаление встраиваемого модуля до использования переключателя командной строки `/ResetAddin` с командой `Devenv.exe`, панели инструментов, созданные встраиваемым модулем, при этом не удаляются. Таким образом, пользователи все так же видят пустые инструменты, засоряющие среду разработки, удалить которые можно только при помощи макроса Visual Studio или файла сценария JavaScript/VBScript, который программным образом обращается к IDE.

Так как мне совершенно не нравится такое положение вещей, в первую очередь я попытался решить эту проблему, используя интересную возможность под названием «временные панели инструментов». Я надеялся, что IDE будет сохранять местоположение временной панели инструментов, позволяя восстанавливать ее в том же месте. После того как я закодировал поддержку временной панели инструментов во встраиваемом модуле `WhoAmI`, я обнаружил, что местоположение панели не сохранялось, и, что еще хуже, временная панель инструментов выводилась в случайных местах на экране.

Так как появление панелей инструментов в случайных точках — это крайне нежелательная практика разработки пользовательских интерфейсов, мне пришлось остановиться на решении с постоянной панелью инструментов. Это возложило на мой установочный код ответственность за удаление панели инструментов при удалении встраиваемого модуля. Если вы реализуете индивидуальную установку для каждого пользователя и помещаете встраиваемый модуль в каталог C:\Documents and Settings*имя_пользователя*\My Documents\Visual Studio 2005\Addins, то можете использовать для решения задачи удаления встраиваемого модуля файл JavaScript, основанный на модели автоматизации Visual Studio.

Однако такие встраиваемые модули, как `WhoAmI`, нужно устанавливать в глобальный каталог встраиваемых модулей (C:\Documents and Settings\All Users\Application Data\Microsoft\MSEnvShared\Addins), так как они должны присутствовать во всех экземплярах Visual Studio. Когда я поменял код, чтобы установка выполнялась в глобальный каталог, я осознал, что пользовательское действие по удалению панели инструментов работает только для учетной записи пользователя, с которой удаление и выполняется. Причина этой проблемы заключается в том, что местоположение панели инструментов сохраняется индивидуально для каждого пользователя.

В итоге мне не удалось найти хорошее решение для этой ошибки. Мои встраиваемые модули, создающие панели инструментов, также устанавливают файлы JavaScript, которые остаются в каталогах после удаление модулей и которые необходимо выполнить под каждой учетной записью пользователя, чтобы удалить панели инструментов. Вы найдете эти файлы в каталоге .\Install\WintellectTools.

Страницы параметров и встраиваемый модуль HiddenSettings

Как я упомянул во введении к этому разделу, команда Visual Studio значительно усовершенствовала поддержку страниц параметров в Visual Studio 2005. Теперь нам не нужно создавать отдельные неуправляемые модули DLL на C++ для страниц, которые должны отображаться в диалоговом окне Options (Параметры). Страницы параметров упростились практически до уровня стандартных элементов управления Windows Forms.

Вам даже не потребуется писать код — все происходит в файле .AddIn. Для каждой страницы параметров вы просто добавляете элемент `<ToolsOptionsPage>`, содержащий элементы `Category` и `SubCategory` для описания места, где эта страница должна выводиться в диалоговом окне Options (Параметры). В элементе `SubCategory` при помощи элемента `FullClassName` вы сообщаете Visual Studio, какой тип необходимо загрузить для создания страницы параметров. И, наконец, в элементе `Assembly` вы указываете, где найти этот тип. В следующем фрагменте из файла .AddIn демонстрируется настройка страницы параметров:

```
<ToolsOptionsPage>
  <Category Name="HiddenSettings">
      <SubCategory Name="Debugger">
          <Assembly>.\HiddenSettings\HiddenSettings.dll</Assembly>
          <FullClassName>HiddenSettings.DebuggerOptionsPage</FullClassName>
      </SubCategory>
  </Category>
</ToolsOptionsPage>
```

Для того чтобы показать вам пример страниц параметров, я создал целый встраиваемый модуль, `HiddenSettings`, который всего лишь добавляет в диалоговое окно Options (Параметры) две страницы параметров, Debugger и Text Editor Guidelines, помещая их в узел HiddenSettings, как показано на рис. 7.4 и 7.5 соответственно.

Как вы, вероятно, догадались, на странице параметров Debugger отображаются скрытые параметры, которые использует отладчик. Первый параметр позволяет вам установить каталог кэша для исходных файлов, извлекаемых благодаря реализованной в отладчике поддержке сервера-источника. Подробнее о сервере-источнике рассказывается в разделе «Настройте сервер-источник» в главе 2. Второй параметр на этой странице позволяет настроить каталог для загрузки исходного кода CCP (Code Center Premium), если у вас есть доступ к исходному коду Windows. Дополнительную информацию о CCP вы найдете на веб-

узле http://www.microsoft.com/resources/sharedsource/ccp/premium.mspx. Последний параметр на странице параметров Debugger отключает предупреждение безопасности, которое появляется каждый раз при присоединении к процессу, выполняющемуся под учетной записью другого пользователя.

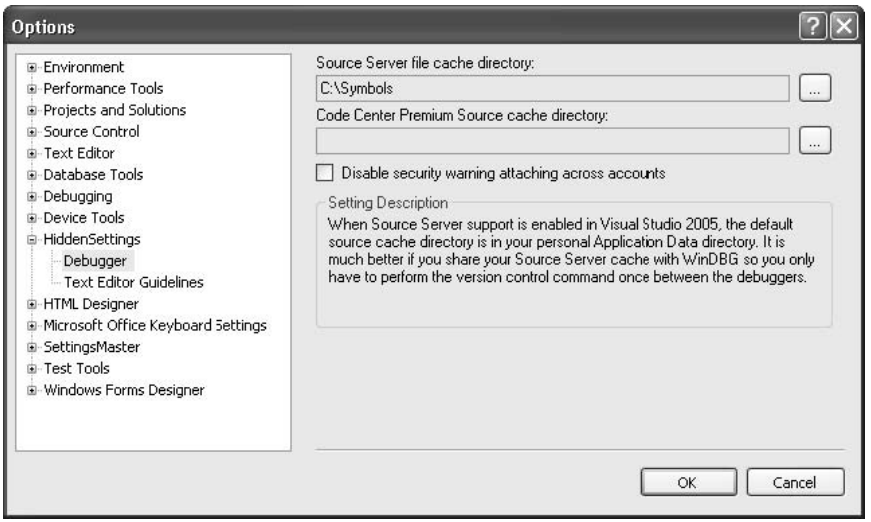

Рис. 7.4. Страница параметров Debugger узла HiddenSettings

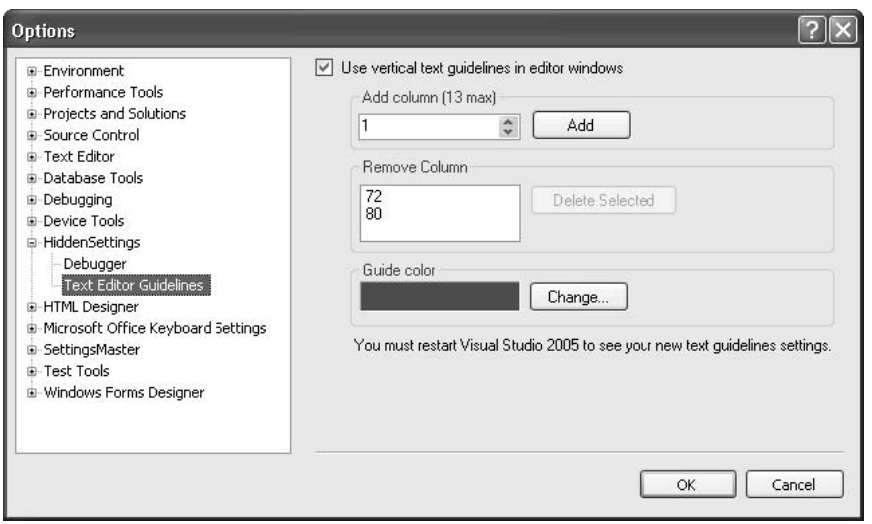

Рис. 7.5. Страница параметров Text Editor Guidelines узла HiddenSettings

Страница параметров Text Editor Guidelines отрывает великолепную не документированную возможность окон редакторов Visual Studio. Во многих редакторах программного кода, например в выдающемся Visual SlickEdit от SlickEdit,

есть очень простая, но крайне полезная вещь: вертикальная линия, идущая сверху вниз в определенных столбцах. Если вам приходится учитывать ограничение на длину строк, как, например, мне для исходного кода, показанного в этой книге, то очень удобно сразу же видеть, когда вы приближаетесь к этому лимиту. Сара Форд (Sara Ford), одна из разработчиков Visual Studio, написала об этом в своем блоге (http://blogs.msdn.com/saraford/archive/2004/11/15/257953.aspx), и я сразу же решил создать пользовательский интерфейс, упрощающий использование этого ключа. Значения ключа устанавливаются довольно хитро, но теперь вам не приходится беспокоиться об этом.

Как я упоминал в предыдущем разделе, существует связанная с локализацией ошибка, из-за которой вам приходится жестко кодировать значения категории и подкатегории страницы параметров в атрибуте `Name` соответствующих субэлементов `<ToolsOptionsPage>` `<Category>` и `<SubCategory>` в файле .AddIn. Еще одна относящаяся к страницам параметров ошибка заключается в том, что вы не получаете правильное уведомление, когда пользователь переходит к другому элементу управления. Это затрудняет проверку данных, поэтому приходится искать другие способы проверять, что вводимые пользователем данные допустимы. Например, на странице параметров Debugger я не разрешаю вводить каталог кэша файлов сервера-источника в поле, запрещая его редактирование и разрешая только чтение. Для того чтобы удостовериться, что пользователь выберет допустимый каталог, я использую класс `FolderBrowserDialog`.

Встраиваемый модуль SettingsMaster

Пока что я знакомил вас с относительно простыми встраиваемыми модулями, но настало время обратиться к более продвинутому модулю, который, по моему мнению, будет вам очень полезен, — `SettingsMaster`. Как я говорил во введении, назначение модуля `SettingsMaster` состоит в том, чтобы упростить для вас обновление настроек сборки в разных проектах и решениях. Теперь, если вам необходимо изменить элемент, например выходной каталог, или добавить новое определение в компиляцию, это можно сделать одним щелчком мыши, не доводя себя до безумия обновлением 37 различных проектов вручную.

Если вы приобрели предыдущее издание этой книги, то у вас есть версия `SettingsMaster`, которая работает с Visual Studio .NET 2003. Модуль `SettingsMaster` завоевал большую популярность, и, когда я изучал отзывы о той версии, наиболее часто я сталкивался с мнением, что его абсолютно необходимо обновить до Visual Studio 2005. То, что мне сначала казалось небольшим обновлением кода, превратилось в полное переписывание встраиваемого модуля с нуля. Причиной тому стали изменения в проектных моделях Visual Studio, а также мое желание сделать реализацию значительно проще.

Одним из крупнейших изменений в новой версии стало то, что `SettingsMaster` работает только с проектами на Visual Basic и C#. Предыдущая версия также поддерживала неуправляемые проекты на C++, но, учитывая наличие новой превосходной возможности наследования свойств Visual C++ через таблицы свойств .vsprops, нет никакой необходимости воспроизводить эту функцию IDE.

Использование встраиваемого модуля SettingsMaster

После того как вы установите SettingsMaster из файла WintellectToolsInstall.MSI, вы увидите, что в интегрированную среду разработки добавляется панель инструментов с двумя кнопками. Хотя пользовательский интерфейс очень прост, когда вы щелкаете одну из этих кнопок, за сценой происходит много важных событий. Независимо от того, работаете вы с обычным проектом .NET или с одним из немного странных проектов Microsoft ASP.NET 2.0, вы можете менять любые параметры, которые открывает модель автоматизации. Все они хранятся в файлах .SettingsMaster, представляющих собой всего лишь XML-файлы.

Первая кнопка на панели инструментов SettingsMaster связана с командой SettingsMaster.CorrectCurrentSolution. Эта команда применяет файл параметров по умолчанию ко всем проектам в решении, которые разрешают обновление. Обновляемые проекты — это проекты .NET, доступные не только для чтения. Если проект доступен только для чтения из системы управления версиями, то встраиваемый модуль SettingsMaster не проверяет его, так как мне показалось, что в подобной ситуации очень просто случайно поменять настройки сборки и привести главную сборку в полнейший беспорядок.

Для того чтобы сменить файл .SettingsMaster, который применяется к проектам по нажатию этой кнопки, откройте раздел SettingsMaster в диалоговом окне Options (Параметры). Первоначально по умолчанию там выбран файл с понятным названием SettingsMasterDefaults.SettingsMaster, который устанавливается вместе с встраиваемым модулем SettingsMaster и меняет все ваши параметры сборки для всех проектов и конфигураций на параметры, рекомендованный мной в главе 2.

Вторая кнопка связана с командой SettingsMaster.CustomProjectUpdate, которая позволяет вам выбирать несколько проектов для обновления, а потом запрашивает файл .SettingsMaster с настройками для этих проектов. Как и при использовании SettingsMaster.CorrectCurrentSolution, обновлять можно только проекты .NET, доступные не только для чтения. Если вы не знали, то в проводнике решений (Solution Explorer) для выбора нескольких проектов нужно щелкнуть их, удерживая клавишу Ctrl.

У команды SettingsMaster.CustomProjectUpdate есть дополнительная возможность: если вы выполняете ее из окна Command (Команда), то после имени команды можете передать полный путь к конфигурационному файлу. Это позволяет быстрее применять пользовательские настройки. В действительности, можно даже написать микрокоманды для всех пользовательских файлов .SettingsMaster, которые вам могут потребоваться. В следующем макросе показано, как просто это делается:

```
Public Sub ApplyFavorites()
    DTE.ExecuteCommand("SettingsMaster.CustomProjectUpdate", _
                       "C:\Settings\Favorites.SettingsMaster")
End Sub
```

Для того чтобы вы могли полностью насладиться возможностями обновления своих проектов, установка WintellectToolsInstall.msi включает еще три файла

.SettingsMaster. Файлы CodeAnalysisOnAndAllErrors.SettingsMaster и CodeAnalysisOff.SettingsMaster включают и выключают применение правил Code Analysis для всех конфигураций. Файл EnableDocComments.SettingsMaster включает создание XML-файла с документацией для всех конфигураций. В исходном коде для модуля SettingsMaster есть еще несколько файлов .SettingsMaster, которые я применял для настройки проекта, содержащего исходный код этой книги, а также для отладки встраиваемых модулей.

Создать собственный файл .SettingsMaster довольно просто. Создавая новый файл .SettingsMaster, выберите для свойства схемы XML-кода файл SettingsMaster.xsd либо из установочного каталога, либо из каталога .\SettingsMaster исходного кода. В файле SettingsMaster.xsd постоянно используются тэги <xs:documentation>, поэтому вы без проблем поймете, за что отвечает каждый элемент. В листинге 7.3 показан пример файла .SettingsMaster. Главное, что нужно запомнить, — что все индивидуальные элементы параметров проекта или конфигурации напрямую соответствуют имени автоматизации проекта или конфигурации. В разделе «Главное о реализации» далее в этой главе я подробнее поговорю о решениях, которые принимал при компоновке файлов .SettingsMaster. Обязательно прочитайте листинг 7.3, потому что там обсуждается особая обработка некоторых узлов, что должно помочь вам при создании собственных пользовательских файлов .SettingsMaster.

Листинг 7.3. Пример файла .SettingsMaster

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ************************************************************************
* Debugging Microsoft .NET 2.0 Applications
* Copyright © 1997-2006 John Robbins All rights reserved.
************************************************************************ -->
<!-- Пример файла параметров. -->
<SettingsMasterConfiguration
    xmlns="http://schemas.wintellect.com/SettingsMaster/2006/02">
  <NormalProject>

    <!-- Установка общих свойств проекта для всех языков. -->
    <CommonProjectProperties>
      <Company>Wintellect</Company>
    </CommonProjectProperties>

    <VBProjectProperties>
      <!-- Особые свойства для VB, 0 = выкл., 1 = вкл. -->
      <OptionExplicit>1</OptionExplicit>
      <OptionStrict>1</OptionStrict>
    </VBProjectProperties>

    <ConfigurationProperties Name="All" Platform="Any CPU">
      <!-- Установка для действия запуска значения 1, что означает запуск
           программы, указанной в элементе <StartProgram>. -->
```

```
      <StartAction>1</StartAction>
      <!-- Особая обработка элемента StartProgram использует переменные
      <!-- среды. В этом примере я устанавливаю в качестве программы для
      <!-- отладки Visual Studio 2005 и пользуюсь преимуществом того, что
      <!-- при установке Visual Studio на всех машинах устанавливается
      <!-- переменная %VS80COMNTOOLS%. Во время обработки не выполняется
      <!-- проверка на пустые переменные среды. -->
      <StartProgram>%VS80COMNTOOLS%..\IDE\devenv.exe</StartProgram>
      <!-- XML-файл документации. Если в пути есть символы $(OutputPath), то
      <!-- свойство OutputPath заменяется соответствующим образом. Если
      <!-- используется $(AssemblyName), подставляется имя текущей сборки.
      <!-- SettingsMaster игнорирует $(OutputPath) для проектов на Visual
      <!-- Basic, так как для них это свойство не имеет значения. -->

<DocumentationFile>$(OutputPath)\$(AssemblyName).xml</DocumentationFile>
      <!-- Считает все предупреждения ошибками. -->
      <TreatWarningsAsErrors>true</TreatWarningsAsErrors>
    </ConfigurationProperties>

    <!-- Конфигурация для отладочных сборок на C#. -->
    <CSharpConfigurationProperties Name="Debug" Platform="Any CPU">
      <!-- Константы компиляции отделяются друг от друга символами ';'.
      <!-- Обратите внимание, что свойства DefineDebug и DefineTrace
      <!-- игнорируются при конфигурации, но учитываются, если установить
      <!-- значение DEBUG;TRACE для свойства DefineConstants. -->
      <DefineConstants>DEBUG;TRACE</DefineConstants>
    </CSharpConfigurationProperties>

    <!-- Конфигурация для релизов на C#. -->
    <CSharpConfigurationProperties Name="Release" Platform="Any CPU">
      <DefineConstants>TRACE</DefineConstants>
    </CSharpConfigurationProperties>

    <!-- Конфигурация для отладочных сборок VB. -->
    <VBConfigurationProperties Name="Debug" Platform="Any CPU">
      <!-- Настройка условной компиляции Debug. -->
      <DefineDebug>true</DefineDebug>
      <!-- Настройка условной компиляции Trace. -->
      <DefineTrace>true</DefineTrace>
    </VBConfigurationProperties>

    <!-- Конфигурация для релизов на VB. -->
    <VBConfigurationProperties Name="Release" Platform="Any CPU">
      <!-- Настройка условной компиляции Trace. -->
      <DefineTrace>true</DefineTrace>
    </VBConfigurationProperties>
  </NormalProject>
```

продолжение

Листинг 7.3 (*продолжение*)

```
  <AspNetProject>
    <AspNetProjectProperties>
      <!-- Включение использования правил Code Analysis. -->
      <EnableFxCop>true</EnableFxCop>
      <!-- Включение неуправляемой отладки. -->
      <EnableUnmanagedDebugging>1</EnableUnmanagedDebugging>
    </AspNetProjectProperties>
  </AspNetProject>
</SettingsMasterConfiguration>
```

Первоначальное исследование

Когда я впервые взглянул на задачу обновления версии `SettingsMaster` для Visual Studio .NET 2003, я практически сразу же осознал, что это потребует чего-то большего, чем простой перекомпиляции. У меня была разработана система вокруг определения свойств в конфигурациях, которая в то время была абсолютно разумной. Я получал набор XML-узлов, в которых устанавливалось значение `Debug`, и знал, какую конфигурацию применяет этот набор. Теперь конфигурации также включают тип процессора, так как в .NET 2.0 можно указывать значения `Itanium`, `x64`, `x86` или независимое от платформы значение `Any CPU`. Хотя я не мог предвидеть, как Microsoft реализует схему поддержки процессоров, мне следовало бы обработать в предыдущей версии установку свойств для самих проектов. Это такие свойства, как в элементах `<CommonProjectProperties>` и `<VBProjectProperties>` в листинге 7.3.

Еще одной постоянной проблемой предыдущей версии было то, что я не добавлял файл схемы для выполнения проверки и помощи пользователям в создании собственных файлов. Если бы я реализовал хороший файл схемы, то позволил бы волшебству XML делать основную работу по проверке данных в файле, вместо того чтобы делать это вручную в своем коде.

Так как проекты ASP.NET 2.0 по умолчанию — это, в действительности, и не проекты вовсе, мне пришлось изучить, как они могут влиять на дизайн модуля `SettingsMaster`. Как я и подозревал, проекты ASP.NET совершенно не похожи на «обычные» проекты .NET. Я не имею в виду, что проекты ASP.NET ненормальные, но единственная общая черта между проектами ASP.NET и проектами библиотек классов заключается в том, что они позволяют редактировать содержимое в окне кода.

Изучив разнообразные типы проектов, я создал множество макросов для вывода всех свойств различных проектов и конфигураций. В то время как обычные проекты и конфигурации отдавали все значения, которые можно менять на страницах свойств, проекты ASP.NET показывали лишь несколько значений.

Я был разочарован, так как надеялся, что значения параметров MSBuild, доступные в диалоговом окне Web Site Properties (Свойства веб-узла), будут открыты, чтобы можно было устанавливать такие элементы, как ключ строгого имени и сборки с фиксированными именами. Фактически, устанавливать разрешается только типы отладки (.NET, SQL или неуправляемая), а также указывать, долж-

ны ли работать правила Code Analysis. Это лучше, чем ничего, но ограничивает применение встраиваемого модуля SettingsMaster для проектов ASP.NET. Исследуя проекты ASP.NET, я заметил, что разнообразные параметры сохраняются и в файле решения, и в скрытом файле .suo.

Возвращаясь к обычным проектам .NET, могу сказать, что больше всего я волновался о том, насколько сильно будут отличаться друг от друга проекты Smart Device и проекты Windows. Хорошие новости — настоящие отличия касаются только свойств, относящихся к конкретным языкам, а не к типам платформ. Отличия, зависящие от языков, не так важны, поэтому справиться с ними было несложно.

В предыдущей версии SettingsMaster основной частью работы была обработка огромных различий между проектами .NET и C++. Как я уже упомянул, в систему C++ было внесено огромное изменение в виде таблиц свойств. Идея заключается в том, что вы помещаете общие макросы и определения сборок в файл .vsprops и добавляете этот файл в свой проект, чтобы наследовать из него значения. В проектах C++ есть пользовательский интерфейс для управления таблицами свойств — окно Property Manager (Диспетчер свойств). Чтобы открыть это окно в проекте C++, выберите команду Property Manager (Диспетчер свойств) в меню View (Вид). Благодаря этой возможности теперь вы можете настраивать общие параметры сборок в файлах, добавлять эти файлы в окно Property Manager (Диспетчер свойств) и менять параметры проектов на C++, не пробираясь через сотни страниц свойств. Питер Хене (Peter Huene) представил отличное обсуждение использования окна Property Manager (Диспетчер свойств) в статье по адресу http://blogs.msdn.com/peterhu/archive/2004/06/07/150488.aspx. Так как в Visual Studio уже есть возможность добавления параметров проектов к проектам C++, я не стал повторно реализовывать это в модуле SettingsMaster.

Главное о реализации

После того как я завершил исследование, моей главной задачей стал выбор формата файла .SettingsMaster. Так как установка свойств для проекта или конфигурации выполняется довольно просто, я сразу же решил, что элементы свойств должны соответствовать именам свойств. В этом случае для установки любого свойства конфигурации можно использовать подобную строку:

```
configuration.Properties.Item ( currProperty.Name ).Value = value;
```

Свойство Value – это объект, поэтому я обрабатываю значение элемента как строку, а .NET выполняет для меня все необходимые преобразования.

Благодаря сопоставлению имен элементов именам свойств определять элементы в файле .SettingsMaster можно индивидуально, а в сочетании с аккуратной работой со схемой это позволило мне добавлять свойства, не меняя код. Оказалось, что я потратил треть времени разработки на файл SettingsMaster.xsd, чтобы гарантировать, что продумал его абсолютно правильно.

Очевидно, что в исходном коде модуля SettingsMaster должно быть заключено какое-то априорное знание о схеме, и в табл. 7.1 перечисляются основные узлы для обычного проекта, которые выводятся под элементами <NormalProject>.

Так как для проектов ASP.NET доступно лишь несколько свойств, в табл. 7.2 перечисляются все значения свойств, которые разрешается устанавливать внутри элементов <AspNetProjectProperties>.

Таблица 7.1. Дочерние элементы NormalProject

Имя узла	Описание
CommonProjectProperties	Содержит все свойства проекта, общие для проектов на Visual Basic и C#, такие как ApplicationIcon, Company и PostBuildEvent
VBProjectProperties	Содержит свойства проекта, относящиеся только к Visual Basic: OptionCompare, OptionExplicit и OptionStrict
ConfigurationProperties	Основной узел, в котором определяются все главные свойства для конфигурации, такие как CheckForOverflowUnderflow, DocumentationFile и TreatWarningsAsErrors. У этого узла два обязательных атрибута. Атрибут Platform должен содержать строку, идентифицирующую тип процессора, и может принимать значения x86, x64, Itanium и Any CPU. Атрибут Name представляет имя конфигурации. Особый случай атрибута Name — это значение All, которое заставляет модуль SettingsMaster применять все настройки узлов ко всем конфигурациям указанной в атрибуте Platform платформы. Это позволяет вам устанавливать общие параметры для отладочных сборок и релизов, такие как DocumentationFile и RunCodeAnalysis
CSharpConfigurationProperties	Содержит параметр, относящийся только к C#, DefineConstants. Атрибуты Name и Platform также можно использовать здесь
VBConfigurationProperties	Содержит параметры, относящиеся только к Visual Basic: DefineConstants, DefineDebug и DefineTrace. Атрибуты Name и Platform также можно использовать здесь

Таблица 7.2. Дочерние элементы AspNetProjectProperties

Имя узла	Описание
EnableFxCop	Присвойте значение 1, чтобы разрешить применение к проекту правил Code Analysis
EnableNTLMAuthentication	Присвойте значение 1, чтобы разрешить аутентификацию NT Lan Manager Authentication
EnableSQLServerDebugging	Присвойте значение 1, чтобы разрешить отладку Microsoft SQL Server. Для того чтобы отлаживать хранимые процедуры, вы должны быть системным администратором SQL Server

Имя узла	Описание
EnableUnmanagedDebugging	Присвойте значение 1, чтобы разрешить неуправляемую отладку
FxCopRules	Строка, содержащая правила Code Analysis, которые вы хотите разрешить, запретить и обрабатывать как предупреждения. Пример определения этой строки см. в файле CodeAnalysisOnAndAllErrors.SettingsMaster

После того как определил ключевую логику для файлов .SettingsMaster и их обработки, я обратился к вопросу трактовки включенного состояния для команд. Основное беспокойство вызывало правильное применение команд, чтобы не допустить изменения параметров в проектах, доступных только для чтения, или во время отладки. Логика команды `SettingsMaster.CorrectCurrentSolution` проста: соответствующая ей кнопка становится доступной, только если решение открыто, содержит, по крайней мере, один проект Visual Basic или C#, и этот проект доступен не только для чтения. Что касается команды `SettingsMaster.CustomProjectUpdate`, то по крайней мере один из выбранных проектов должен допускать запись.

Во время разработки я столкнулся с двумя проблемами, о которых стоит упомянуть. Первоначально я разрабатывал модуль так, чтобы файл SettingsMaster.dll загружался из локального каталога C:\Documents and Settings*имя_пользователя*\My Documents\Visual Studio 2005\Addins. Все шло превосходно до тех пор, пока не настало время протестировать загрузку SettingsMaster.dll из каталога C:\Documents and Settings\All Users\Application Data\Microsoft\MSEnvShared\Addins, то есть глобального местоположения встраиваемых модулей. У модуля `SettingsMaster` два постоянных параметра: булево значение, указывающее, нужно ли автоматически сохранять любые измененные проекты, и строковое значение, содержащее полный путь к файлу .SettingsMaster по умолчанию, который используется для выполнения команды `SettingsMaster.CorrectCurrentSolution`. Первые запуски из глобального местоположения приводили к исключениям сериализации, которые создавались, как только я делал попытку считать данные из файла параметров в каталоге C:\Documents and Settings*имя_пользователя*\Local Settings\Application Data\Wintellect\SettingsMaster. В данных исключения говорилось, что коду модуля SettingsMaster.dll не удавалось загрузить SettingsMaster.dll.

Это заставило меня чесать затылок до тех пор, пока через несколько секунд меня не осенило, что Visual Studio, вероятно, не проверяет при разрешении сборок глобальное местоположение. К счастью, это очень легко исправить, всего лишь обработав дескриптор событий `CurrentDomain.AssemblyResolve` и вернув по запросу правильную сборку SettingsMaster.dll.

Вторая небольшая проблема случилась, когда я захотел немного исправить `SettingsMaster` и переместил сборку, загружающуюся из глобального местоположения, обратно в каталог C:\Documents and Settings*имя_пользователя*\My Documents\Visual Studio 2005\Addins. Совершенно неважно, что я исправлял, но я не мог получить унаследованный от `IDTExtensibility.OnConnection` метод, который вызывается со значением `ext_ConnectMode.ext_cm_UISetup`, чтобы сбросить панели инструментов и команды. Оказывается, что когда вы используете превосходный ключ

/ResetAddIn команды Devenv.exe, IDE забывает удалить еще кое-что, помимо панели инструментов. В разделе реестра HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\8.0\PreloadAddinStateManaged среда Visual Studio хранит ключ с именем, составленным из имени встраиваемого модуля и пути к файлу .AddIn. Если значение этого ключа равно 0x2, то Visual Studio предполагает, что пользовательский интерфейс уже инициализирован, и не передает значение ext_ConnectMode.ext_cm_UISetup повторно. Если удалить этот ключ или присвоить ему значение 0x1, то у вас появляется шанс снова инициализировать встраиваемый модуль.

Возможные будущие усовершенствования

Встраиваемый модуль SettingsMaster крайне полезен и в том виде, в каком он существует сейчас, но если вы ищете идею для проекта, то для того, чтобы сделать SettingsMaster еще лучше, в него можно внести много усовершенствований:

- редактора конфигурации для конфигурационных XML-файлов нет. Так как таблицы свойств программировать относительно просто, вы могли бы написать редактор конфигурационных файлов, чтобы не приходилось править файлы вручную. Этот конфигурационный редактор должен открываться с панели инструментов SettingsMaster, а также со страницы свойств в диалоговом окне Options (Параметры);
- еще одна возможность, которую было бы относительно просто добавить — это обработчик событий, который отслеживал бы загрузку проекта и автоматически обновлял параметры проекта;
- также было бы удобно иметь команду для вывода и записи в конфигурационный файл .SettingsMaster текущих параметров проекта, чтобы их можно было применять к другим проектам;
- для того чтобы обеспечить дополнительное взаимодействие с пользователем, можно также выводить изменения, вносимые модулем SettingsMaster, в окне Output (Вывод);
- встраиваемый модуль SettingsMaster поддерживает только проекты ASP.NET, стандартные для Visual Studio 2005. Если вы преобразовываете проект ASP.NET 1.1 в проект ASP.NET, то новая модель отличается настолько сильно, что перенос становится очень сложным. Кроме того, если вам необходима реальная сборка для проектов ASP.NET 2.0, то вам не повезло. К счастью, Скотт Гутри (Scott Guthrie), руководитель подразделения ASP.NET, Internet Information Server и Windows Forms в Microsoft, попросил свою команду разработать модель проектов веб-приложений Visual Studio 2005 Web Application Project Model (http://webproject.scottgu.com/), для того чтобы решить эти проблемы. Если вам нравится эта модель, то вы можете реализовать ее поддержку в SettingsMaster.

Резюме

С новыми возможностями макросов и встраиваемых модулей в IDE Visual Studio 2005 у разработчиков теперь есть неограниченная власть над средой. Теперь ее можно делать именно такой, какой она должна быть для того, чтобы вы могли

быстрее решать любые проблемы. В этой главе я хотел продемонстрировать вам некоторые хитрости построения реальных и полезных макросов и встраиваемых модулей. Хотя в IDE все еще остается достаточно странностей, имеющиеся возможности более чем компенсируют их. Надеюсь, что сложности, с которыми я столкнулся и о которых рассказал вам, сэкономят вам время при разработке собственных расширений для IDE Visual Studio.

Разработчики уже давно жаждут получить всю мощь Visual Studio 2005 в свои руки. Теперь она у нас есть, и я настоятельно рекомендую вам, не откладывая в долгий ящик, реализовать все инструменты, о которых вы так давно мечтали. А остальные смогут воспользоваться ими!